# Multi-valued Logic Programming Semantics: An Algebraic Approach [1]

**Bamshad Mobasher**
Dept. of Computer Science
University of Minnesota
mobasher@cs.umn.edu

**Don Pigozzi**
Dept. of Mathematics
Iowa State University
dpigozzi@iastate.edu

**Giora Slutzki**
Dept. of Computer Science
Iowa State University
slutzki@cs.iastate.edu

## Abstract

In this paper we introduce the notion of join-irreducibility in the context of bilattices and present a procedural semantics for bilattice based logic programs which uses, as its basis, the join-irreducible elements of the knowledge part of the bilattice. The join-irreducible elements in a bilattice represent the "primitive bits" of information present within the system. In bilattices which have the descending chain property in their knowledge ordering, these elements provide a small representative set completely characterizing the bilattice. The overall complexity of the inference systems based on such bilattices can thus be reduced by restricting attention to the join-irreducible elements.

## 1 Introduction

Many AI practitioners have shied away from using standard logic programming languages as the knowledge representation language in AI systems, partly due to the difficulty associated with representing uncertain, incomplete, or conflicting information in such languages. The root of these difficulties is the inherent limitations of first-order logic as the basis of the standard logic programming systems. One such limitation is the monotonicity of first-order logic which makes it unsuitable as a mechanism for revisable reasoning. Another important limitation stems from the all-or-nothing nature of classical first-order logic: statements can be evaluated to be completely true or completely false. Intelligent agents, however, must often deal with information which is uncertain, or incomplete.

It is therefore desirable to construct logic programming systems that can overcome the difficulties mentioned above. The work presented here is an attempt to provide a general framework for an efficient procedural semantics of such logic programming languages. The above brief discussion suggests that such systems must have two common characteristics: they must rely on the expressive power of an underlying multi-valued logic which can deal with contradictory as well as incomplete or uncertain information, and secondly, such systems should be able to interpret statements not only based on their truth or falsity, but also based on some measure of the knowledge or information contained within those statements.

Our attention is focused on those logics that have a knowledge dimension as well as a truth dimension and thus can be used to model the connection between truth and knowledge

[1] A preliminary version of this paper appeared in the proceedings of the Workshop on Uncertainty in Databases and Deductive Systems (WUDDS '94), Ithaca, NY, November 1994.

in a particular logic program or deductive database. The first logic of this kind originated with Belnap [3]. It is based on the idea that information in a database can have both a positive and a negative content with regard to the truth of a particular event. The two situations in which only positive or only negative information is available give rise to two truth values that can be identified with classical `true` and `false`, respectively. But there are two other situations: when the information has both a positive and a negative content, and where there is no information of either kind. These lead to a third and fourth "truth value" that are denoted respectively by $\top$ and $\bot$. Part of the motivation here is that, in a distributed database, information about a given event is collected from various sources at various times and some of it might be contradictory. So the truth value of the event can be viewed as representing our state of knowledge about the classical truth or falsity of the event rather than its actual truth or falsity.

Ginsberg [14] has suggested using *bilattices* as the underlying framework for various AI inference systems including those based on default logics, truth maintenance systems, probabilistic logics, and others. Bilattices are mathematical structures with two separate orderings, called knowledge and truth orderings, that provide a framework for the study of knowledge-truth interaction. These ideas were pursued by Fitting [11, 12] in the context of logic programming semantics. More recently, bilattices and their extensions have been used in the literature to model a variety of reasoning mechanisms about uncertainty in the presence of incomplete or contradictory information. For example, in [27], a variant of Fittings extension of logic programming to bilattices was used to deal with a form of negation as failure as well as a second explicit negation in logic programs. In [17] bilattices were extended to include a third ordering (called the *precision* ordering) in order to effectively deal with varying degrees of belief and doubt in probabilistic deductive database.

In [20, 21] we developed a knowledge-based procedural semantics based on the 4-valued Belnap bilattice and proved soundness and completeness theorems with respect to Fitting's declarative fixpoint semantics. A novel feature of this procedural semantics is the introduction of completely symmetric notions of *proof* and *refutation*. Intuitively, the existence of a proof, respectively refutation, for a given goal corresponds to having positive, respectively negative, information about it.

This paper introduces two new features into logic programming over multiple-valued logics. The first is a procedural semantics of great generality and conceptual simplicity that applies to any bilattice. This procedural semantics proves to be both sound and complete with respect to the natural fixpoint semantics over the bilattice. We introduce the notion of a *b-derivation* for each element of the bilattice except $\top$ and $\bot$. (In the 4-element case `true`-derivations coincide with proofs and a `false`-derivations with refutations.) We prove the soundness and completeness theorems for this procedural semantics, again with respect to Fitting's declarative fixpoint semantics.

As might be expected, the simplicity and generality of this approach comes at a high cost in computational complexity. For a given truth value $b$, the search for a $b$-derivation of a complex goal $G$ may entail searches for $c$-derivations of the subformulas of $G$ for a large number of truth values $c$ that are only remotely related to $b$; moreover, this complexity ramifies as we pass down the parse tree of $G$. It turns out that for finite distributive bilattices (and, more generally, bilattices with the *descending chain property*), we can restrict our attention to derivations that range over a relatively small subset of special truth-values.

These special truth values turn out to be the so-called *join irreducible* elements of the knowledge part of the bilattice. Ginsberg [14] has discussed the ramifications of reducing the complexity of bilattice based inference systems by focusing on a smaller set of representative elements called *grounded* elements. As we will see, join-irreducible elements provide an even smaller set of representative elements which represent the most "primitive" bits of information. In fact, this difference could be exponential for certain classes of bilattices. More recently, the notion of join-irreduciblity has been used in connection with a proof theory for bilattice-based logics [1].

The most novel feature of this paper is the introduction of a much more efficient procedural semantics that proves to be equivalent to the general semantics for a special class of bilattices that includes all finite distributive bilattices. The formulation of the *join-irreducible* procedural semantics, as it is called, and the proof of its equivalence with the general semantics, is based on a fundamental result of lattice theory, due to Birkhoff, that establishes a duality between finite distributive lattices and arbitrary finite partially ordered sets. The main result of the paper is the Completeness Theorem for join-irreducible procedural semantics (Theorem 4.8). It is obtained from the Completeness Theorem for the general procedural semantics (Theorem 3.21) by directly reducing the join-irreducible semantics to the latter. The join-irreducible semantics can provide the basis for effective implementation of a family of logic programming languages which, depending on the choice of the underlying logic, can be used for a variety of reasoning tasks in intelligent systems.

## 2 Bilattices and Join-irreducible Elements

A lattice is a partially ordered set $\langle L, \leq \rangle$ in which each pair $a, b$ of elements has a least upper bound $(a \vee b)$ and a greatest lower bound $(a \wedge b)$. The elements $a \vee b$ and $a \wedge b$ are respectively called the *meet* and *join* of $a$ and $b$. The largest and smallest elements of $L$, if they exist, are called "top" and "bottom," denoted respectively by $\top$ and $\bot$. $L$ is *complete* if every subset $X$ of $L$ has a least upper bound $(\bigvee X)$ and greatest lower bound $(\bigwedge X)$. $L$ has the *descending chain property*, or DCP, if all of its strictly descending chains are finite. $L$ is *distributive* if it satisfies the distributive law $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ or (equivalently) $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$. An element $a$ of $L$ is *join-irreducible* if $a \neq \bot$ and $a = b \vee c$ implies that $b = a$ or $c = a$, for all $b, c \in L$. We denote the set of join-irreducible elements of $L$ by $JIR(L)$. In the context of distributive lattices, we will find the following property of join-irreducible elements useful [7].

**Lemma 2.1** *Let $L = \langle L, \leq \rangle$ be a distributive lattice, $c_1, c_2, \cdots, c_n \in L$, and let $c \in JIR(L)$. Then $c \leq c_1 \vee c_2 \vee \cdots \vee c_n$ if and only if $c \leq c_j$ for some $j$ $(1 \leq j \leq n)$.*

The following well-known theorem, due to Birkhoff, shows that for distributive lattices with the DCP, the join-irreducible elements provide a representative set from which all other elements can be obtained. This fact will play a critical role in reducing the complexity of the operational semantics for bilattice based logic programming languages.

**Theorem 2.2** (Birkhoff [4]) *Let $L$ be a distributive lattice satisfying the DCP. Then for every $a \in L$, there exists an irredundant decomposition of $a$ as a finite join of join-irreducible*

*elements in L, that is, $a = b_1 \vee \cdots \vee b_n$, where $b_i \in JIR(L)$ and none of the $b_i$ can be removed. Furthermore, if $b_1 \vee \cdots \vee b_n = c_1 \vee \cdots \vee c_m$ are two irredundant decompositions of $a$ as joins of join-irreducibles, then $n = m$ and $b_i = c_i$ $(1 \leq i \leq n = m)$, up to renumbering.*

Let $\langle \mathcal{B}, \leq_t, \leq_k \rangle$ be a structure consisting of a nonempty set $\mathcal{B}$ and two partial orderings, $\leq_t$ and $\leq_k$ on $\mathcal{B}$. If $\leq_t$ is a lattice ordering, let `true` and `false` denote the top and bottom elements (if defined), $\wedge$ and $\vee$ the meet and join, and $\bigwedge$ and $\bigvee$ the infinitary meet and join (if defined). Similarly, if $\leq_k$ is a lattice ordering, the corresponding notions are denoted respectively by $\top$, $\bot$, $\otimes$, $\oplus$, $\prod$, and $\sum$.

A *bilattice* is an algebraic structure which we shall view as a space of generalized truth values with two lattice orderings, one measuring degrees of truth, and the other measuring degrees of knowledge. A negation operator provides the connection between the two orderings. The formal definition is as follows.

**Definition 2.3** (Ginsberg [14]) A *bilattice* is a structure $\langle \mathcal{B}, \leq_t, \leq_k, \neg \rangle$ consisting of a nonempty set $\mathcal{B}$, partial orderings $\leq_t$ and $\leq_k$, and a mapping $\neg : \mathcal{B} \to \mathcal{B}$ such that:

1. $\langle \mathcal{B}, \leq_t \rangle$ and $\langle \mathcal{B}, \leq_k \rangle$ are complete lattices;
2. $x \leq_t y$ implies $\neg y \leq_t \neg x$, for all $x, y \in \mathcal{B}$;
3. $x \leq_k y$ implies $\neg x \leq_k \neg y$, for all $x, y \in \mathcal{B}$;
4. $\neg\neg x = x$, for all $x \in \mathcal{B}$.

Note that $\neg$ reverses the $\leq_t$-ordering, like classical negation, but preserves the $\leq_k$-ordering. Thus, by part 4 of the above definition, it is an automorphism of the lattice $\langle \mathcal{B}, \leq_k \rangle$.

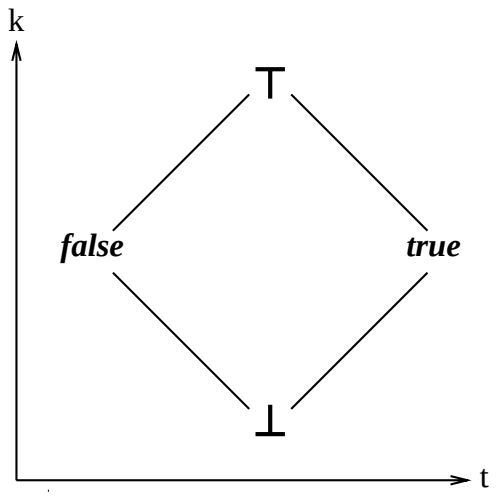
In a bilattice, $\leq_t$ represents the truth ordering and $\leq_k$ the knowledge ordering. Informally, $p \leq_k q$ means that the evidence underlying an assignment of the truth value $p$ is subsumed by the evidence underlying an assignment of $q$. In other words, more is known about the truth or falsity of a statement whose truth value is $q$ than is known about one whose truth value is $p$. The lattice operations for the $\leq_t$-ordering are natural generalizations of the familiar classical ones.

A bilattice satisfies the *interlacing conditions* [11, 12] if:

1. $x \leq_t y \Longrightarrow x \oplus z \leq_t y \oplus z$ and $x \otimes z \leq_t y \otimes z$, for all $x, y, z \in \mathcal{B}$;
2. $x \leq_k y \Longrightarrow x \vee z \leq_k y \vee z$ and $x \wedge z \leq_k y \wedge z$, for all $x, y, z \in \mathcal{B}$.

In other words, the interlacing conditions say that the lattice operations in each ordering of the bilattice are monotonic with respect to the other ordering. There are twelve distributive laws associated with the four operations $\wedge$, $\vee$, $\oplus$, and $\otimes$. A bilattice is *distributive* if all twelve distributivity laws hold. A bilattice satisfies the *infinite distributivity condition* if all of the infinitary distributive laws, such as $a \otimes \bigvee_i b_i = \bigvee_i (a \otimes b_i)$ and $a \wedge \prod_i b_i = \prod_i (a \wedge b_i)$, hold. It is easy to show that the distributive laws, in fact, imply the interlacing conditions. For the remainder of this paper we assume that all bilattices under consideration satisfy the distributive laws, though, some of the results presented hold under less restrictive conditions.

Figure 1: The bilattice $\mathcal{FOUR}$




In [20] we gave a natural procedural semantics for logic programs based on Belnap's four-valued logic [3], which is called $\mathcal{FOUR}$. It is the simplest example of a nontrivial bilattice. This bilattice is depicted in Figure 1. There are many other interesting nonclassical logics that can be represented using bilattices. Some examples are Reiter's default logic [26], fuzzy logics [29], Kripke's intuitionistic logic model [9], and modal logics based on the many-worlds semantics [14]. For a more detailed discussion see [12, 14].

In this paper, based on the notion of join-irreducibility in a bilattice, we introduce a new algebraic procedural semantics for logic programming over arbitrary distributive bilattices that satisfy certain finiteness conditions. Ginsberg [14] showed that every distributive bilattice can be represented as a sublattice of the direct product of two lattices. We later use this representation to characterize the join-irreducible elements of a distributive bilattice. One of our main results is that, by imposing certain finiteness conditions on distributive bilattices, we can restrict our attention to the join-irreducible elements in the bilattice, thus reducing the overall complexity of the procedural semantics. Let us now make these notions more precise.

**Definition 2.4** Let $\langle L_1, \leq_1 \rangle$ and $\langle L_2, \leq_2 \rangle$ be lattices and $h : L_1 \to L_2$ a lattice isomorphism. Define binary relations $\leq_t$ and $\leq_k$ on $L_1 \times L_2$ and a unary operation $\neg : L_1 \times L_2 \to L_1 \times L_2$ by:

1. $\langle x_1, x_2 \rangle \leq_t \langle y_1, y_2 \rangle$ if $x_1 \leq_1 y_1$ and $y_2 \leq_2 x_2$,
2. $\langle x_1, x_2 \rangle \leq_k \langle y_1, y_2 \rangle$ if $x_1 \leq_1 y_1$ and $x_2 \leq_2 y_2$,
3. $\neg\langle x, y \rangle = \langle h^{-1}(y), h(x) \rangle$.

The structure $\langle L_1 \times L_2, \leq_t, \leq_k, \neg \rangle$ is denoted by $\mathcal{B}_h(L_1, L_2)$. If $\langle L_1, \leq_1 \rangle = \langle L_2, \leq_2 \rangle = \langle L, \leq \rangle$ and $h$ is the identity automorphism, we denote $\mathcal{B}_h(L_1, L_2)$ by $\mathcal{B}(L)$. Note that in this special case (3) becomes

$3'$ $\neg\langle x, y\rangle = \langle y, x\rangle$.

The following theorem is a slight sharpening of results due to Fitting and Ginsberg.

**Theorem 2.5** (Ginsberg [14], Fitting [11]) *Let $L_1$ and $L_2$ be complete lattices and let $h : L_1 \to L_2$ be a lattice isomorphism. Then $\mathcal{B}_h(L_1, L_2)$ is bilattice and is distributive if $L_1$ and $L_2$ are both distributive. In particular, for any complete lattice $L$, $\mathcal{B}(L)$ is a bilattice and is distributive if $L$ is distributive. Conversely, every distributive bilattice is isomorphic to $\mathcal{B}(L)$ for some complete distributive lattice $L$.*

**Proof:** If $L_1$ and $L_2$ are complete lattices and $h : L_1 \to L_2$ is a isomorphism, it is routine to check that $\mathcal{B}_h(L_1, L_2)$ is a bilattice and is distributive if $L_1$ (equivalently $L_2$) is distributive. In particular, $\mathcal{B}(L)$ is a bilattice and distributive if $L$ is distributive.

Let $\mathcal{B} = \langle \mathcal{B}, \leq_t, \leq_k, \neg\rangle$ be a distributive bilattice, and recall that $\vee$, $\wedge$, `true`, and `false` are the lattice operations under $\leq_t$, and $\oplus$, $\otimes$, $\top$, and $\bot$ are the operations under $\leq_k$. Define $L_1 = \{\, x \in \mathcal{B} \mid \bot \leq_t x \,\}$ and $L_2 = \{\, x \in \mathcal{B} \mid \bot \geq_t x \,\}$, and consider the lattices $L_1 = \langle L_1, \leq_t \rangle$ and $L_2 = \langle L_2, \geq_t \rangle$, where $\leq_t$ is the truth ordering inherited from $\mathcal{B}$ and $\geq_t$ is its dual. $L_1$ is clearly a sublattice of $\langle \mathcal{B}, \leq_t \rangle$ and $L_2$ is the dual of a sublattice of $\langle \mathcal{B}, \leq_t \rangle$. It is also easy to see that $\neg$ (restricted to $\text{Ł}_1$) is lattice isomorphism between $L_1$ and $L_2$ whose inverse is also $\neg$. Thus $\mathcal{B}_\neg(L_1, L_2)$ is defined and is a distributive bilattice. In order to prove it is isomorphic to $\mathcal{B}$ we first prove that $x \leq_k y$ iff $x \leq_t y$ for all $x, y \in L_1$, and $x \leq_k y$ iff $x \geq_t y$ for all $x, y \in L_2$, and hence

$$\langle L_1, \leq_k \rangle = \langle L_1, \leq_t \rangle \quad \text{and} \quad \langle L_2, \leq_k \rangle = \langle L_2, \geq_t \rangle.$$

Let $x, y \in L_1$, i.e., $x \vee \bot = x$ and $y \vee \bot = y$. Then by the distributivity of $\mathcal{B}$,

$$x \oplus y = x \oplus (y \vee \bot) = (x \oplus y) \vee (x \oplus \bot) = (x \oplus y) \vee x, \tag{1}$$

$$x \vee y = x \vee (y \oplus \bot) = (x \vee y) \oplus (x \vee \bot) = (x \vee y) \oplus x \tag{2}$$

If $x \leq_k y$, then, by (1), $y = x \oplus y = (x \oplus y) \vee x = y \vee x$, and thus $x \leq_t y$. Conversely, if $x \leq_t y$, then, by (2), $y = x \vee y = (x \vee y) \oplus x = x \oplus y$, and thus $y \leq_t x$. This gives the first equality $\langle L_1, \leq_k \rangle = \langle L_1, \leq_t \rangle$ The second $\langle L_2, \leq_k \rangle = \langle L_2, \geq_t \rangle$ follows directly from the first together with the fact that $\neg$ is an isomorphism between $\langle L_1, \leq_t \rangle$ and $\langle L_2, \geq_t \rangle$ and is also an isomorphism (as is easily verified) between $\langle L_1, \leq_k \rangle$ and $\langle L_2, \leq_k \rangle$.

Define $g : \mathcal{B} \to \mathcal{B}_\neg(L_1, L_2)$ by setting $g(x) = \langle x \vee \bot, x \wedge \bot\rangle$ for all $x \in \mathcal{B}$. By distributivity of $\mathcal{B}$ we have

$$\begin{aligned}
(x \vee \bot) \oplus (x \wedge \bot) &= ((x \vee \bot) \oplus x) \wedge ((x \vee \bot) \oplus \bot)) \\
&= (x \oplus x) \vee (\bot \oplus x)) \wedge ((x \oplus) \vee (\bot \oplus \bot)) \\
&= (x \vee x) \wedge (x \vee \bot) \\
&= x \wedge x \\
&= x. \qquad (3)
\end{aligned}$$

Hence $g$ is injective. Again using the distributivity of $\mathcal{B}$ we can show by a similar argument that, for all $x \in L_1$ and $y \in L_2$, $g(x \oplus y) = \langle x, y\rangle$. So $g$ is a bijective, and to show it is

a bilattice isomorphism it suffices to show that it is an order isomorphism with respect to both $\leq_t$ and $\leq_k$ and preserves $\neg$. Let $x, y \in \mathcal{B}$. Using (3) and the interlacing conditions we get directly from the definition of $\mathcal{B}_\neg(L_1, L_2)$ that

$$\begin{aligned} x \leq_t y \quad &\text{iff} \quad x \vee \bot \leq_t y \vee \bot \text{ and } x \wedge \bot \leq_t y \wedge \bot \\ &\text{iff} \quad x \vee \bot \leq_t y \vee \bot \text{ and } y \wedge \bot \geq_t x \wedge \bot \\ &\text{iff} \quad g(x) \leq_t g(y). \end{aligned}$$

Again using (3) and the interlacing conditions we get

$$\begin{aligned} x \leq_k y \quad &\text{iff} \quad x \vee \bot \leq_k y \vee \bot \text{ and } x \wedge \bot \leq_k y \wedge \bot \\ &\text{iff} \quad x \vee \bot \leq_t y \vee \bot \text{ and } x \wedge \bot \geq_t y \wedge \bot \\ &\text{iff} \quad g(x) \leq_k g(y). \end{aligned}$$

Finally, let $x \in \mathcal{B}$. Then $g(\neg x) = \langle (\neg x) \vee \bot, (\neg x) \wedge \bot \rangle = \langle (\neg(x \wedge \bot), \neg(x \vee \bot) \rangle = \neg\langle x \vee \bot, x \wedge \bot \rangle = \neg g(x)$. Thus $\mathcal{B}$ is isomorphic to $\mathcal{B}_\neg(L_1, L_2)$ as a bilattice.

Let $L = L_1$, and define $f : \mathcal{B}_\neg(L_1, L_2) \to \mathcal{B}(L)$ by $f(\langle x, y \rangle) = \langle x, \neg y \rangle$ for all $x \in L_1$ and $y \in L_2$. Let $x_1, y_1 \in L_1$ and $x_2, y_2 \in L_2$ and assume $\langle x_1, x_2 \rangle \leq_t \langle y_1, y_2 \rangle$ in $\mathcal{B}(L_1, L_2)$. Then by Definition 2.4 (1), $x_1 \leq y_1$ in $L_1$ and $y_2 \leq x_2$ in $L_2$. This means $x_1 \leq_t y_1$ and $x_2 \leq_t y_2$ in $\mathcal{B}$. Hence $x_1 \leq y_1$ and $\neg y_2 \leq \neg x_2$ in $L$. So we get finally $\langle x_1, \neg x_2 \rangle \leq_t \langle y_1, \neg y_2 \rangle$ in $\mathcal{B}$(L), i.e., $f(\langle x_1, y_1 \rangle) \leq_t f(\langle x_2, y_2 \rangle)$. So $f$ preserves the truth ordering and in a similar way in can be shown to preserve the knowledge ordering. Since it is clearly bijective, it is an isomorphism of both the truth and knowledge lattices. Finally, $f(\neg\langle x, y \rangle) = f(\langle \neg y, \neg x \rangle) = \langle \neg y, \neg\neg x \rangle = \langle \neg y, x \rangle = \neg\langle x, \neg y \rangle = \neg f(\langle x, y \rangle)$. So $f$ is a bilattice isomorphism. Thus $\mathcal{B}_\neg(L_1, L_2) \cong \mathcal{B}(L)$ and hence also $\mathcal{B} \cong \mathcal{B}(L)$. ∎

Given the structure of $\mathcal{B}(L)$, it is easy to verify that the basic bilattice operations are defined as follows (we will denote the join and meet operations in the underlying lattice $L$ by $+$ and $\cdot$, respectively). Let $c_1, c_2, d_1, d_2 \in L$.

$$\begin{aligned} \langle c_1, d_1 \rangle \otimes \langle c_2, d_2 \rangle &= \langle c_1 \cdot c_2, d_1 \cdot d_2 \rangle \\ \langle c_1, d_1 \rangle \oplus \langle c_2, d_2 \rangle &= \langle c_1 + c_2, d_1 + d_2 \rangle \\ \langle c_1, d_1 \rangle \wedge \langle c_2, d_2 \rangle &= \langle c_1 \cdot c_2, d_1 + d_2 \rangle \\ \langle c_1, d_1 \rangle \vee \langle c_2, d_2 \rangle &= \langle c_1 + c_2, d_1 \cdot d_2 \rangle. \end{aligned}$$

Intuitively, one can think of the components $x$ and $y$ of a pair $\langle x, y \rangle$ as summarizing the evidence for and the evidence against an assertion, respectively. Belnap's four-valued logic, for instance, can be represented by the above construction if one takes $L$ to be the 2-element lattice $\{0, 1\}$ with $0 \leq 1$. A probabilistic bilattice, based on the logic introduced in [29], can be formed by taking for $L$ the interval [0,1] with the natural ordering. In the latter logic, each truth value can represent the degrees of belief and doubt.

The set of all elements of a bilattice $\mathcal{B}$ that are join-irreducible in the knowledge ordering (*k-join-irreducible*) is denoted by $JIR_k(\mathcal{B})$, and the corresponding set for the truth ordering is denoted by $JIR_t(\mathcal{B})$. Since $\neg$ is an automorphism of the knowledge lattice, $\neg b$ will be k-join-irreducible whenever $b$ is k-join-irreducible.

In [14], Ginsberg discussed the ramifications of reducing the complexity of bilattice based inference systems by representing bilattices using a smaller set of "basis" elements. He captured this idea in the notion of *groundedness*. Grounded elements of a bilattice are those representing "primitive" bits of information. More formally:

**Definition 2.6** (Ginsberg [14]) An element $x$ of a bilattice $\mathcal{B}$ is *t-grounded* if, for any $y \in \mathcal{B}$, $x \leq_t y \Rightarrow x \leq_k y$; and it is *f-grounded* if, for any $y \in \mathcal{B}$, $x \leq_t y \Rightarrow x \geq_k y$. The element $x$ is *grounded*, if it is either t-grounded or f-grounded. Furthermore, a bilattice $\mathcal{B}$ is called *grounded at* $x$ if $x$ can be written as the join (in the knowledge ordering) of grounded elements of $\mathcal{B}$.

Given Ginsberg's observation that every element of a distributive bilattice can be viewed as representing the evidence both for and against an assertion, grounded elements are those elements for which one of these two components is empty. More precisely, it is easy to show that for a bilattice $\mathcal{B} = \mathcal{B}(L)$, an element $x = \langle x_1, x_2 \rangle$ is t-grounded iff $x_2 = \bot$ and it is f-grounded iff $x_1 = \bot$.

**Lemma 2.7** *Let $\mathcal{B} = \mathcal{B}(L)$ be a distributive bilattice, where $L = \langle L, \leq \rangle$ is a complete distributive lattice. Let $x \in \mathcal{B}$. Then*

1. *$x$ is t-grounded if and only if $x = \langle x_1, \bot \rangle$, for some $x_1 \in L$;*
2. *$x$ is f-grounded if and only if $x = \langle \bot, x_2 \rangle$, for some $x_2 \in L$.*

**Proof:** We prove part (1); part (2) is proved similarly.

Let $x = \langle x_1, x_2 \rangle \in \mathcal{B}$ be t-grounded and $y = \langle y_1, \bot \rangle \in \mathcal{B}$ with $y_1 \geq x_1$. Then, $y \geq_t x$, and since $x$ is t-grounded, we have $\langle y_1, \bot \rangle \geq_k \langle x_1, x_2 \rangle$. Hence, $y_1 \geq x_1$ and $\bot \geq x_2$, implying that $x_2 = \bot$. On the other hand, suppose that $y = \langle y_1, y_2 \rangle \geq_t \langle x_1, \bot \rangle = x$. Then, $y_1 \geq x_1$ and $y_2 \leq \bot$, implying that $y_2 = \bot$ and hence, $y \geq_k x$. ∎

Thus, intuitively, the grounded elements of a bilattice are those truth values which do not encode any conflicting information about a particular sentence. It is interesting to note that for a bilattice $\mathcal{B} = \mathcal{B}(L)$, each of the sets of t-grounded and f-grounded elements, with the $\leq_k$-ordering, forms a sublattice that is isomorphic to the underlying lattice $L$

Using the lattice theoretic notion of join-irreducibility, we can reduce the set of grounded elements of a bilattice to an even smaller set of representative elements.

**Definition 2.8** Let $\mathcal{B}$ be a bilattice. An element $x \in \mathcal{B}$ is a *positive* k-join-irreducible element if $x \in JIR_k(\mathcal{B})$ and $x$ is t-grounded. It is a *negative* k-join-irreducible element if $x \in JIR_k(\mathcal{B})$ and it is f-grounded. The sets of positive and negative k-join-irreducibles are denoted respectively by $JIR_k^+(\mathcal{B})$ and $JIR_k^-(\mathcal{B})$.

We can further characterize these elements by the following lemma.

**Lemma 2.9** *Let $\mathcal{B} = \mathcal{B}(L) = \langle L \times L, \leq_t, \leq_k, \neg \rangle$ be a distributive bilattice, where $\langle L, \leq, +, \cdot \rangle$ is a complete distributive lattice. Then*

1. $JIR_k^+(\mathcal{B}) = \{\langle a, \bot\rangle \mid a \in JIR(L)\}$;

2. $JIR_k^-(\mathcal{B}) = \{\langle \bot, a\rangle \mid a \in JIR(L)\}$;

3. $JIR_k(\mathcal{B}) = JIR_k^+(\mathcal{B}) \cup JIR_k^-(\mathcal{B})$.

**Proof:** The proof of parts 1 and 2 is an easy consequence of the definitions, Lemma 2.7, and the fact that, if $b = \langle b_1, b_2\rangle$ and $c = \langle c_1, c_2\rangle$, then $b \oplus c = \langle b_1 + c_1, b_2 + c_2\rangle$. Part 3 is a direct consequence of parts 1 and 2, and the definition of join-irreducibility. ∎

To illustrate these concepts, consider the bilattice $\mathcal{NINE}$, depicted in Figure 2. This bilattice can be constructed by taking the set $P = \{0, b, 1\}$ with the ordering $0 < b < 1$ and then forming the structure $\mathcal{B}(P)$. Then

$$JIR_k^+(\mathcal{NINE}) = \{\langle b, 0\rangle, \langle 1, 0\rangle\} \text{ and } JIR_k^-(\mathcal{NINE}) = \{\langle 0, b\rangle, \langle 0, 1\rangle\}.$$

The k-join-irreducible elements in the bilattice $\mathcal{NINE}$ are $\langle 1, 0\rangle$, $\langle 0, 1\rangle$, $\langle b, 0\rangle$, and $\langle 0, b\rangle$. Intuitively we can think of $\langle b, 0\rangle$, and $\langle 0, b\rangle$ as "maybe true", and "maybe false," respectively. The truth value $\langle b, 0\rangle$, for example, encodes the existence of partial evidence for a statement and no evidence against it. The element $\langle 0, 1\rangle$ represents `false` while $\langle 1, 0\rangle$ represents `true`. In fact, using the characterization of join-irreducibles given in Lemma 2.9, it is easy to see that, in any distributive bilattice $\mathcal{B}$, if the top element in the underlying lattice is join-irreducible, then elements `false` and `true` are among the k-join-irreducible elements (`false` $\in JIR_k^-(\mathcal{B})$ and `true` $\in JIR_k^+(\mathcal{B})$). Furthermore, note that in $\mathcal{NINE}$ and in fact in any distributive bilattice $\mathcal{B}$ we have:

$$\neg(JIR_k^+(\mathcal{B})) = JIR_k^-(\mathcal{B}) \text{ and } \neg(JIR_k^-(\mathcal{B})) = JIR_k^+(\mathcal{B}).$$

By Lemma 2.1 every element in $\mathcal{NINE}$ (except the bottom element) can be uniquely represented as an irredundant join of k-join-irreducible elements. For example, the element $\langle b, 1\rangle$ can be represented as the join of the two k-join-irreducible elements $\langle b, 0\rangle$ and $\langle 0, 1\rangle$.

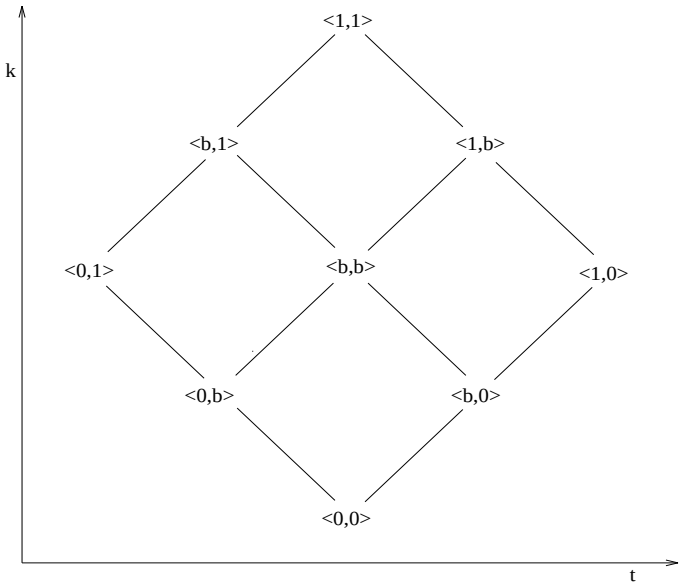
The bilattice $\mathcal{FOUR}$ contains four elements, `true`, `false`, $\bot$, and $\top$. The two k-join-irreducible elements are `true` and `false`, with `true` positive and `false` negative. More generally, let $L$ be any finite, linearly ordered lattice with $n$ elements. Then $\mathcal{B}(L)$ has $n^2$ elements of which $2n - 2$ are k-join-irreducible. If the underlying lattice is a powerset lattice, then the difference between the number of grounded and k-join-irreducible elements is exponential. For example, let $X$ be any set and let $L$ be the lattice $\langle \mathcal{P}(X), \subseteq \rangle$. If $X$ has $n$ elements then the cardinality of $\mathcal{B}(L)$ is $2^{2n}$. It is easy to see that $\mathcal{B}(L)$ has $2^{n+1} - 1$ grounded elements, but only $2n$ k-join-irreducible elements.

The following lemmas provide the basis for the join-irreducible procedural semantics defined in the next section.

**Lemma 2.10** *Let $\mathcal{B} = \mathcal{B}(L) = \langle L \times L, \leq_t, \leq_k, \neg\rangle$ be a distributive bilattice, where $\langle L, \leq, +, \cdot\rangle$ is a complete distributive lattice. Let $a, b, c \in \mathcal{B}$.*

1. *If $c \in JIR_k(\mathcal{B})$, then $c \leq_k a \oplus b$ if and only if $c \leq_k a$ or $c \leq_k b$;*

Figure 2: The bilattice $\mathcal{NINE}$




2. *If* $c \in JIR_k^+(\mathcal{B})$, *then*

   (a) $c \leq_k a \vee b$ *if and only if* $c \leq_k a$ *or* $c \leq_k b$;

   (b) $c \leq_k a \wedge b$ *if and only if* $c \leq_k a$ *and* $c \leq_k b$;

3. *If* $c \in JIR_k^-(\mathcal{B})$, *then*

   (a) $c \leq_k a \vee b$ *if and only if* $c \leq_k a$ *and* $c \leq_k b$;

   (b) $c \leq_k a \wedge b$ *if and only if* $c \leq_k a$ *or* $c \leq_k b$.

**Proof:** Part 1 is a special case of Lemma 2.1 We prove part 3; part 2 is proved similarly. Since $c \in JIR_k^-(\mathcal{B})$, $c = \langle \bot, c_2 \rangle$, where $c_2 \in JIR(L)$. Let $a = \langle a_1, a_2 \rangle$ and $b = \langle b_1, b_2 \rangle$. To prove part (a) we first note that $a \vee b = \langle a_1, a_2 \rangle \vee \langle b_1, b_2 \rangle = \langle a_1 + b_1, a_2 \cdot b_2 \rangle$. Now we have:

$$\begin{aligned} c \leq_k a \vee b \quad \text{iff} \quad & \langle \bot, c_2 \rangle \leq_k \langle a_1 + b_1, a_2 \cdot b_2 \rangle \\ \text{iff} \quad & \bot \leq a_1 + b_1 \text{ and } c_2 \leq a_2 \cdot b_2 \\ \text{iff} \quad & c_2 \leq a_2 \cdot b_2 \\ \text{iff} \quad & c_2 \leq a_2 \text{ and } c_2 \leq b_2 \\ \text{iff} \quad & \langle \bot, c_2 \rangle \leq_k \langle a_1, a_2 \rangle \text{ and } \langle \bot, c_2 \rangle \leq_k \langle b_1, b_2 \rangle \\ \text{iff} \quad & c \leq_k a \text{ and } c \leq_k b. \end{aligned}$$

To prove part (b) we note that $a \wedge b = \langle a_1, a_2 \rangle \wedge \langle b_1, b_2 \rangle = \langle a_1 \cdot b_1, a_2 + b_2 \rangle$. Now we have:

$$c \leq_k a \wedge b \quad \text{iff} \quad \langle \bot, c_2 \rangle \leq_k \langle a_1 \cdot b_1, a_2 + b_2 \rangle$$

iff $\bot \leq a_1 \cdot b_1$ and $c_2 \leq a_2 + b_2$
iff $c_2 \leq a_2 + b_2$
iff $c_2 \leq a_2$ or $c_2 \leq b_2$ (by Lemma 2.1)
iff $\langle \bot, c_2 \rangle \leq_k \langle a_1, a_2 \rangle$ or $\langle \bot, c_2 \rangle \leq_k \langle b_1, b_2 \rangle$
iff $c \leq_k a$ or $c \leq_k b$. ∎

**Lemma 2.11** *Let $\mathcal{B} = \mathcal{B}(L)$ be a distributive bilattice, where $\langle L, \leq, +, \cdot \rangle$ is a complete distributive lattice. Let $b, c \in JIR_k(\mathcal{B})$.*

1. *If $c \in JIR_k^+(\mathcal{B})$ and $b \leq_k c$, then $b \in JIR_k^+(\mathcal{B})$.*
2. *If $c \in JIR_k^-(\mathcal{B})$ and $b \leq_k c$, then $b \in JIR_k^-(\mathcal{B})$.*

**Proof:** Suppose that $c \in JIR_k^+(\mathcal{B})$ and $b \leq_k c$. Then $c = \langle c_1, \bot \rangle$, where $c_1 \in JIR(L)$. Now, $b \in JIR_k(\mathcal{B})$, hence $b = \langle b_1, \bot \rangle$ or $b = \langle \bot, b_2 \rangle$, where $b_i \in JIR(L)$, for $i = 1, 2$. But, $b \neq \langle \bot, b_2 \rangle$, since otherwise $b \leq_k c$ implies that $\langle \bot, b_2 \rangle \leq_k \langle c_1, \bot \rangle$, which is impossible because $b_2 \neq \bot$. Hence $b = \langle b_1, \bot \rangle \in JIR_k^+(\mathcal{B})$.

The case when $c \in JIR_k^-(\mathcal{B})$ is proved similarly. ∎

# 3 Logic Programming over Distributive Bilattices

## 3.1 Logic Programming Syntax

Our logic programming language, denoted by $\mathcal{L}$, will have a distributive bilattice $\mathcal{B}$ as the underlying space of truth values. The alphabet of $\mathcal{L}$ consists of the usual sets of variables, constants, predicate symbols, and function symbols, similar to conventional logic programming. As is commomplace in first-order logic we assume that there is also an infinite number of new constants, called *generic constants*, that are distinct from the regular constants in the sense that they may not appear in any clause of a program over $\mathcal{L}$. Intuitively, these generic constants play the role of "arbitrary but fixed" objects in the language. The extension of the language by these constants is done for technical reasons which will become more clear in the sequel. In addition, $\mathcal{L}$ includes the connectives $\leftarrow$, $\neg$, $\wedge$, $\vee$, $\otimes$, and $\oplus$. The connectives $\wedge$ and $\vee$ represent the meet and join operations of the bilattice in the truth ordering and $\otimes$ and $\oplus$ represent the meet and join in the knowledge ordering. The "quantifiers" $\prod, \sum$ represent the infinitary meet and join operations of the bilattice in the knowledge ordering.

The notions of *term* and *ground term* are defined in the usual way. The set $U_{\mathcal{L}}$ of all ground terms in a language $\mathcal{L}$ is called the *Herbrand universe* of $\mathcal{L}$ [18]. An *atom* is either a special nullary predicate $b \in \mathcal{B} \setminus \{\top, \bot\}$, or an expression of the form $p(t_1, \cdots, t_n)$, where $p$ is an $n$-ary predicate symbol and $t_1, \cdots, t_n$ are terms. An atom in which there are no occurrences of variables is called a *ground atom*. *Formulas* are either atoms or expressions of the form $\neg A$, $A \oplus B$, $A \otimes B$, $A \wedge B$, or $A \vee B$, where $A$ and $B$ are formulas. A *complex*

*formula* is a formula which is not an atom. The set of variables occurring in a formula $A$ is denoted by $vars(A)$. A *clause* is an expression of the form:

$$\prod_{x_1 \cdots x_n} (A \leftarrow \sum_{y_1 \cdots y_m} (G)),$$

where $A$ is an atom such that $A \notin \mathcal{B}$ , $G$ is a formula, $x_1, \cdots, x_n$ are the variables occurring in $A$, and $y_1, \cdots, y_m$ are the variables occurring in $G$, but not in $A$. $A$ is called the *head* and $G$ is called the *body* of the clause. Normally, we drop the quantifiers from the clauses and simply write $A \leftarrow G$, where the variables occurring in the head of the clause are implicitly quantified by $\prod$, and the variables occurring in the clause body and not in the clause head are quantified by $\sum$. This convention is a standard practice in logic programming. Of course, in classical logic programming the quantifiers are the truth quantifiers $\forall$ and $\exists$ which are assumed to implicitly quantify a clause. The choice of quantifiers $\prod$ and $\sum$ is motivated by our interest in the knowledge content of statements rather than their truth content.

As usual, a *program* is a finite set of clauses that contain no generic constants. A *goal* is simply a formula that contains no generic constants. The *Herbrand Base* of a program $P$, denoted $B_P$, is the set of all ground atoms using only constants and function or predicate symbols occurring in $P$, and generic constants.

We also assume that in any program clause $A \leftarrow G$, either $vars(A) \subseteq vars(G)$ or $G \in \mathcal{B}$. Any program that does not satisfy this property can easily be transformed to an equivalent program which does. This is done by replacing each clause $A \leftarrow G$ not satisfying the above property by a clause $A \leftarrow G \otimes E(x_1, \cdots, x_n)$, where $\{x_1, \cdots, x_n\} = vars(A) - vars(G)$ and $E$ is a new predicate symbol, and adding the clause $E(x_1, \cdots, x_n) \leftarrow \top$. It is easy to verify that the new program will be semantically equivalent to the original program in the sense of the fixpoint semantics defined in Section 3.3. For more details see [19, 21].

Before describing our logic programming semantics, we need to present some basic concepts in unification theory. A closer examination of these concepts is necessary since we provide a full treatment of variables in our procedural semantics.

## 3.2 Unification and Substitution Unifiers

Substitutions, renamings, composition of substitutions, and basic unification concepts are defined in the standard manner as detailed in [18]. We have attempted to be rather precise in our statements regarding the properties of substitutions and the conditions they must satisfy at various stages of the derivation process. This issue is often ignored or treated rather summarily in the literature, leading to subtle errors or conflicting definitions (see [16]).

**Definition 3.1** A *substitution* $\theta$ is a mapping from variables to terms such that $v\theta \neq v$ for only finitely many $v$. Every substitution is uniquely represented by a finite set of the form $\theta = \{v_1/t_1, \cdots, v_n/t_n\}$, where each $v_i$ is a variable, each $t_i$ is a term distinct from $v_i$, and the variables $v_1, \cdots, v_n$ are distinct. Each element $v_i/t_i$ is called a *binding* for $v_i$. The empty substitution is called the *identity* substitution and is denoted by $\varepsilon$. $\theta$ is *ground* if each $t_i$ is a ground term, and it is *variable-pure* if each $t_i$ is a variable. $\theta$ in *injective* if $t_i \neq t_j$ whenever $i \neq j$. The *domain* of $\theta$ is $\{v_1, \cdots, v_n\}$ and is denoted by $dom(\theta)$. The set of variables occurring in the range of $\theta$, i.e., $\cup_{i=1}^{n} vars(t_i)$, is denoted by $vrange(\theta)$.

A substitution $\theta$ is extended to expressions in the following way: Let $\theta = \{v_1/t_1, \cdots, v_n/t_n\}$ and let $E$ be an expression. Then $E\theta$, the *instance of $E$ by $\theta$*, is the expression obtained from $E$ by simultaneously replacing each occurrence of the variable $v_i$ in $E$ by the term $t_i$, $i = 1, \cdots, n$. For a set $S$ of expressions, $S\theta = \{E\theta \mid E \in S\}$. For two expressions $E_1$ and $E_2$, we write $E_1 \leq E_2$, if $E_2 = E_1\sigma$ for some substitution $\sigma$. If $E\theta$ is ground, then $E\theta$ is called a *ground instance* of $E$.

Let $\sigma = \{v_1/t_1, \ldots, v_n/t_n\}$ and $\tau$ be substitutions, $E$ an expression, and $U$ a set of variables. Then $\sigma_U = \{v_{i_1}/t_{i_1}, \ldots, v_{i_k}/t_{i_k}\}$, where $\{v_{i_1}, \ldots, v_{i_k}\} = U \cap dom(\sigma)$; and $\sigma_E = \sigma_{vars(E)}$.

**Definition 3.2** Let $\theta$ and $\sigma$ be substitutions. By the *composition* $\theta\sigma$ of $\theta$ and $\sigma$ we mean their composition as transformations of the set of expressions in the usual sense of functional composition. If $\theta = \{u_1/s_1, \cdots, u_m/s_m\}$ and $\sigma = \{v_1/t_1, \cdots, v_n/t_n\}$, then the representation of $\theta\sigma$ is obtained from the set

$$\{u_1/s_1\sigma, \cdots, u_m/s_m\sigma, v_1/t_1, \cdots, v_n/t_n\}$$

by deleting any binding $u_i/s_i\sigma$ for which $u_i = s_i\sigma$ and deleting any binding $v_j/t_j$ for each $v_j \in \{u_1, \cdots, u_m\}$.

Let $\theta$ and $\eta$ be substitutions.

1. $\theta$ is an *instance of* $\eta$, in symbols $\eta \leq \theta$, if $\theta = \eta\alpha$ for some substitution $\alpha$. In this case we say that $\theta$ is an *instance of $\eta$ by $\alpha$*.

2. Let $U$ be a set of variables. We say that $\theta$ is a *variant of $\eta$ with respect to $U$* if there are instances $\theta'$ and $\eta'$ of $\theta$ and $\eta$, respectively such that $\theta_U = \eta'_U$ and $\eta_U = \theta'_U$.

3. We say that $\theta$ is a *variant* of $\eta$, in symbols $\theta \equiv \eta$, if it is a variant of $\eta$ with respect to the set of all variables, i.e., if each is an instance of the other ($\eta \leq \theta$ and $\theta \leq \eta$).

It is easy to see that $\leq$ is a preordering on substitutions and $\equiv$ is an equivalence relation. In particular it is symmetric, so that $\theta$ is a variant of $\eta$ w.r.t $U$ iff $\eta$ is a variant of $\theta$ w.r.t. $U$. We say that $\theta$ is a *variant of $\eta$ w.r.t.* a given expression $E$ if $\theta$ is a variant of $\eta$ w.r.t. $vars(E)$, i.e., if $E\theta = E\eta'$ and $E\eta = E\theta'$ for some $\theta'$ and $\eta'$ such that $\theta \leq \theta'$ and $\eta \leq \eta'$. The key feature of the variant relation is the fact that, given any expression $E$, any substitution $\theta$, and any finite set $V$ of variables, there is a variant $\eta$ of $\theta$ w.r.t. $E$ such that $vars(E\eta) \cap V = \emptyset$.

In the sequel we often use the expression "$\theta$ is a renaming of $\eta$" as a synonym for "$\theta$ is a variant of $\eta$." Also, when we say that a substitution $\alpha$ is "unique up to renaming," we mean that $\alpha$ can be any member of a $\equiv$-equivalence class.

A substitution $\theta$ is *idempotent* if $\theta\theta = \theta$. The class of idempotent substitutions exhibits some interesting properties which have been extensively studied [19, 25, 8]. In particular, it can be easily verified that $\sigma$ is an idempotent substitution if and only if $dom(\sigma) \cap vrange(\sigma) = \emptyset$.

**Definition 3.3** Let $S$ be a finite set of expressions. A substitution $\theta$ is called a *unifier* for $S$ if $S\theta$ is a singleton. A unifier $\theta$ of $S$ is called a *most general unifier* (*mgu*) for $S$ if $\theta \leq \sigma$ for each unifier $\sigma$ of $S$. The set $S$ is called *unifiable* if it has a unifier.

As a technical tool, we will also need to utilize a special kind of substitution which replaces variables of an expression with new constants not appearing in any program clause, i.e., with generic constants.

**Definition 3.4** A substitution $\delta$ is a *generic constant substitution* or simply a *gc-substitution* if it is of the form

$$\{x_1/a_1, x_2/a_2, \cdots, x_n/a_n\},$$

where $a_1, \cdots, a_n$ are distinct generic constants. If E is an expression, then $\delta$ is a *gc-substitution for* $E$, if $dom(\delta) = vars(E)$.

The following technical lemma connecting notions of a gc-substitutions and the most general unifier will be used in the proof of the Completeness Theorem.

**Lemma 3.5** *Let $A$ and $B$ be atoms such that $vars(A) \cap vars(B) = \emptyset$ and neither $A$ nor $B$ contains a generic constant. Let $\delta$ be a gc-substitution for $A$. Assume $A\delta$ and $B$ are unifiable and $\eta = mgu(A\delta, B)$. Then $A$ and $B$ are unifiable, and if $\mu = mgu(A, B)$, then*

1. *$\delta\eta = \mu\tau$, where $\tau$ is a gc-substitution for $B\mu$, and*
2. *$\mu_A$ is injective and variable-pure.*

**Proof:** $vars(A) \cap vars(B) = \emptyset$ implies $B\delta = B$. So $\eta = mgu(A\delta, B\delta)$, and hence $\delta\eta$ unifies $A$ and $B$. Let $\mu = mgu(A, B)$. Then $\delta\eta = \mu\tau$ for some $\tau$. Since $\delta$ is a gc-substitution for $A$, we have $A\delta = A\delta\eta$. Thus $A\delta = A\delta\eta = A\mu\tau$. Now $dom(\delta) = vars(A)$ and all constants of $\delta$ are generic. Thus, since $A$ and $B$ do not contain any generic constants and $\mu$ is their $mgu$, $\mu$ cannot contain generic constants. But $x\mu\tau = x\delta$, which is generic, for every $x \in vars(A)$. This implies that $\mu_A$ must be variable-pure, and it must also be injective since $\delta$ is injective. It then follows from the equality $A\delta = A\mu\tau$ that $\tau$ is a gc-substitution for $A\mu$ and hence also for $B\mu$. ∎

In our procedural semantics presented below we employ a parallel computation model (see [28, 6]). During the evaluation of a query, even when subgoals share variables, they are solved independently. After termination, however, answer substitutions obtained for shared variables are tested for consistency. We use the notion of substitution unification and substitution unifiers, studied in [19] (see [20]), to ensure the consistency of bindings obtained for shared variables during the computation. A theory of substitution unification based on the solution of equations is investigated in [8, 25]. The notion of unifiable substitutions has been used in concurrent logic programming systems which use parallelism [15].

**Definition 3.6** Let $S$ be a set of substitutions. Then a substitution $\gamma$ is called a *substitution unifier* (*s-unifier*) of $S$, if $S\gamma = \{ \sigma\gamma : \sigma \in S \}$ is a singleton. If such a substitution $\gamma$ exists, then we say that $S$ is *unifiable*. $\gamma$ is a *most general substitution unifier* of $S$, if for every s-unifier $\delta$ of $S$, there is a substitution $\eta$ such that $\delta = \gamma\eta$. We denote the set of all most general s-unifiers of $S$ by $mgsu(S)$.

**Definition 3.7** Let $S$ be a unifiable set of substitutions. A substitution $\delta$ is a *substitution unification* of $S$ if $\delta = \sigma\gamma$, for some $\gamma \in mgsu(S)$ and some $\sigma \in S$. The set of all substitution unifications of $S$ is denoted by $\odot S$. Clearly, for any $\sigma \in S$,

$$\odot S \quad = \quad \{\sigma\tau \mid \tau \in mgsu(S)\} = \sigma\ mgsu(S).$$

When dealing with a pair of substitutions $\sigma$ and $\tau$, we often use a shorthand notation and denote the set of mgsu's of $\sigma$ and $\tau$ by $mgsu(\sigma, \tau)$. Similarly, we denote the set of all substitution unifications of $\sigma$ and $\tau$ by $\sigma \odot \tau$.

It is well-known that mgu's and mgsu's are unique up to renaming [18, 8, 25, 19]. In the sequel we sometimes abuse the notation and interpret $mgsu(S)$ and $\odot S$ as functions returning unique values.

We now present some of the properties of substitution unifiers that will be important in proving subsequent results. The proofs and a more detailed study of these properties can be found in [19, 21].

**Lemma 3.8 (Weak Distributivity of $\odot$)** *Let $\theta$ and $\eta_1, \cdots, \eta_n$ be substitutions, where the $\eta_i$ are unifiable. Then $\theta\eta_1, \cdots, \theta\eta_n$ are unifiable, and*

$$\odot\{\theta\eta_1, \cdots, \theta\eta_n\} \leq \theta(\odot\{\eta_1, \cdots, \eta_n\}).$$

**Lemma 3.9 (Associativity of $\odot$)** *Let $\sigma, \tau, \rho$ be unifiable and idempotent substitutions. Then*

$$(\sigma \odot \tau) \odot \rho \equiv \odot\{\sigma, \tau, \rho\} \equiv \sigma \odot (\tau \odot \rho).$$

The following results are technical and are needed in the proof of the Completeness Theorem. However, the conditions stated in these lemmas are actually general and occur naturally in standard logic programming systems.

**Lemma 3.10** *Let $\sigma_1, \cdots, \sigma_n$, and $\eta_1, \cdots, \eta_n$ be substitutions such that:*

1. *$\sigma_i \leq \eta_i$, for $i = 1, \cdots, n$, and*
2. *$\sigma_i$ is idempotent ($i = 1, \cdots, n$), and*
3. *$\eta_1, \cdots, \eta_n$ are unifiable.*

*Then $\sigma_1, \cdots, \sigma_n$ are unifiable and $\odot\{\sigma_1, \cdots, \sigma_n\} \leq \odot\{\eta_1, \cdots, \eta_n\}$.*

**Lemma 3.11** *Let $G_1$ and $G_2$ be expressions and $\sigma_i$, $\eta_i$, and $\theta$ be substitutions, for $i = 1, 2$ such that:*

1. *$dom(\sigma_i) \subseteq vars(G_i)$, $i = 1, 2$;*
2. *$vrange(\sigma_i) \cap vars(G_i) = \emptyset$, $i = 1, 2$;*
3. *$vrange(\sigma_i) \cap dom(\theta\eta_i) = \emptyset$, $i = 1, 2$;*
4. *$G_i\sigma_i \leq G_i\theta\eta_i$, $i = 1, 2$; and*
5. *$\eta_1$ and $\eta_2$ are unifiable.*

*Then $\sigma_1$ and $\sigma_2$ are unifiable and $\sigma_1 \odot \sigma_2 \leq \theta(\eta_1 \odot \eta_2)$.*

### 3.3 Fixpoint Semantics

In the classical two-valued logic programming, a single step operator on interpretations, denoted $T_P$, is associated with a program. In the absence of negation, this operator is monotonic and has a natural least fixpoint. It is this fixpoint which serves as the denotational meaning of the program. Unfortunately, in the presence of negation in the clause bodies, the $T_P$ operator is no longer monotonic and may not have a fixpoint. The idea of associating such an operator with programs carries over in a natural way to logic programming languages with a distributive bilattice as the space of truth values. However, the ordering in which the least fixpoint is evaluated is the knowledge ordering ($\leq_k$) and not the truth ordering ($\leq_t$). Since negation is monotonic with respect to the knowledge ordering, and thus the knowledge operators are self-dual under negation, presence of negation in the body of program clauses does not pose any of the problems associated with classical logic programming. The fixpoint semantics presented in this section is essentially due to Fitting [11].

An *interpretation* for a program $P$ is a mapping $I : B_P \to \mathcal{B}$. $I$ is extended in a natural way to ground formulas as follows: $I(b) = b$, for every $b \in \mathcal{B} \setminus \{\bot, \top\}$, $I(\neg A) = \neg I(A)$ and $I(A_1 \sqcap A_2) = I(A_1) \sqcap I(A_2)$ for $\sqcap \in \{\wedge, \vee, \otimes, \oplus\}$. We further extend the interpretation $I$ to non-ground formulas:

$$I(G) = \prod\{I(G\sigma) \mid \sigma \text{ is a ground substitution for the variables of } G\}.$$

The following lemma is an easy consequence of the definitions involved.

**Lemma 3.12** *Let $I_1$ and $I_2$ be two interpretations for a program $P$ and let $\sigma$ and $\tau$ be substitutions.*

1. *$I_1(G\sigma) \leq_k I_2(G\tau)$ for every formula $G$ if and only if $I_1(A\sigma) \leq_k I_2(A\tau)$ for every atom $A$.*
2. *$I_1(G\sigma) = I_2(G\tau)$ for every formula $G$ if and only if $I_1(A\sigma) = I_2(A\tau)$ for every atom $A$.*

**Proof:** The implication from left to right in part 1 is trivial. The opposite implication is proved by an easy induction on the structure of $G$, using the fact that the operations $\neg$, $\oplus$, $\otimes$, $\wedge$, and $\vee$ are monotone with respect to $\leq_k$.

Part 2 is an immediate consequence of part 1. ∎

The *semantic operator* $\Phi_P$ is the function from interpretations to interpretations defined as follows:

$$\Phi_P(I)(A) = \begin{cases} A & \text{if } A \in \mathcal{B} \setminus \{\bot, \top\} \\ \sum\{\, I(G\sigma) \mid A' \leftarrow G \in P \text{ and } A = A'\sigma, \text{ with } \sigma \text{ ground} \,\} & \text{otherwise.} \end{cases}$$

The knowledge ordering of $\mathcal{B}$ induces a pointwise partial ordering of interpretations. $\Phi_P$ is monotonic with respect to this ordering since all operations on $\mathcal{B}$ (including the

negation) are monotonic with respect to the knowledge ordering. Hence, by the Knaster-Tarski theorem, $\Phi_P$ has a least fixpoint, which provides the denotational meaning of the program $P$. As is customary we will denote the $n$th iteration of the $\Phi_P$ operator by $\Phi_P \uparrow n$. We formally define these notions in the following.

**Definition 3.13** The *initial interpretation* $I_0$ of a program $P$ is defined as follows. For any atom $A \in B_P$:

$$I_0(A) = \begin{cases} A & \text{if } A \in \mathcal{B} \setminus \{\bot, \top\} \\ \bot & \text{otherwise.} \end{cases}$$

Note that for any atomic formula $G$ and any $c \in \mathcal{B} \setminus \{\bot, \top\}$, if $I_0(G) \geq_k c$, then $G = d$, for some $d \in \mathcal{B}$ with $d \geq_k c$.

**Definition 3.14** The *upward iteration* of $\Phi_P$ is defined as follows:

$$\Phi_P \uparrow \alpha = \begin{cases} I_0 & \text{if } \alpha = 0 \\ \Phi_P(\Phi_P \uparrow (\alpha - 1)) & \text{if } \alpha \text{ is a successor ordinal} \\ \bigsqcup\{\Phi_P \uparrow \beta \mid \beta < \alpha\} & \text{if } \alpha \text{ is a limit ordinal} \end{cases}$$

The smallest ordinal at which this sequence gives the least fixpoint of $\Phi_P$ is called the *closure ordinal*. In $\mathcal{FOUR}$ and in fact in any bilattice which satisfies the infinitary distributivity conditions, $\Phi_P$ is continuous and its closure ordinal is at most $\omega$ [11].

Generic constants behave semantically like variables. The reason for this is that, since generic constants do not occur in any program clause, the program does not provide any information about them. Thus, we can replace generic substitutions in a program with variables without changing the meaning of that program. These ideas are formalized in the following lemma.

**Lemma 3.15** *For every formula $G$ and any gc-substitution $\sigma$ for $G$,*

$$(\Phi_P \uparrow n)(G\sigma) = (\Phi_P \uparrow n)(G), \quad \textit{for every } n < \omega$$

.

**Proof:** Consider the following property of an interpretation $I$.

$$I(G\sigma) = I(G), \quad \text{for every formula } G \text{ and gc-substitution } \sigma \text{ for } G. \tag{4}$$

We first prove that, for every interpretation $I$, if $I$ has the property (4), then so does $\Phi_P(I)$. Note that, to show $\Phi_P(I)$ has the property, it suffices to prove that $\Phi_P(I)(G\sigma) \leq_k \Phi_P(I)(G\delta)$, for every ground substitution $\delta$. Furthermore, by Lemma 3.12, we can assume $G$ is an atom.

Assume $I$ is an arbitrary interpretation such that (4) holds and $G$ is an atom $A$. $\Phi_P(I)(A\sigma)$ is the join of all elements of $\mathcal{B}$ of the form $I(G'\tau)$ such that $A\sigma = B\tau$ for some clause $B \leftarrow G'$ of $P$ and some ground substitution $\tau$. Let $x_1, \ldots, x_n$ be the variables of $A$ so that $A\sigma = A(x_1\sigma, \ldots, x_n\sigma) = B\tau$. Since the constants $x_1\sigma, \ldots, x_n\sigma$ are distinct and do not occur in $B$, there is a substitution $\tau'$ such that $A = B\tau'$. Since $A\sigma = B\tau'\sigma$, we can

assume $\tau'\sigma = \tau$. We also have $A\delta = B\tau'\delta$ and hence $I(G'\tau'\delta)$ is one of the set of elements of the bilattice whose join defines $\Phi_P(I)(A\delta)$. But $I(G'\tau) = I(G'\tau'\sigma) = I(G'\tau') \leq_k I(G'\tau'\delta)$; the second equality holds because of our assumption that $I$ has the property (4). Thus (4) is preserved under $\Phi_P$. The conclusion of the lemma now follows by an easy induction on $n$ and the fact that $I_0$ clearly has the property (4). ■

Interpretations over distributive bilattices exhibit some interesting algebraic properties. In particular, we have found the following results useful in the proofs of our Soundness and Completeness Theorems. The proofs are straightforward (for more details see [19, 21]).

**Lemma 3.16** *Let $\alpha$ and $\beta$ be substitutions, let $I$ be an interpretation, and let $F, G_1, G_2$ be formulas.*

1. *If $\alpha \leq \beta$, then $I(F\alpha) \leq_k I(F\beta)$.*
2. *$I(F\alpha) \leq_k I(F\alpha\beta)$.*
3. *If $\alpha$ and $\beta$ are variants w.r.t. $F$, then $I(F\alpha) = I(F\beta)$.*
4. *$I(G_1) \sqcap I(G_2) \leq_k I(G_1 \sqcap G_2)$, where $\sqcap \in \{\otimes, \oplus, \wedge, \vee\}$.*
5. *If $\alpha$ and $\beta$ are unifiable, then $I(F\alpha) \leq_k I(F(\alpha \odot \beta))$ and $I(F\beta) \leq_k I(F(\alpha \odot \beta))$.*

**Lemma 3.17** *Let $A \leftarrow G \in P$. Then for any substitution $\theta$,*

$$(\Phi_P \uparrow \omega)(A\theta) \geq_k (\Phi_P \uparrow \omega)(G\theta).$$

Since the programs in the logic programming system presented here are interpreted with respect to the knowledge ordering, clearly, the intended use of the system is in situations where we want to focus on the amount or the quality of information available about a goal statement. Let us consider a simple example to motivate the focus on the knowledge ordering.

**Example:** Suppose that we would like a program to determine whether an individual suspected of a crime should be charged with that crime. Our criteria are based upon consistent information regarding three separate conditions. The suspect should be charged if he or she has a motive, does not have an alibi, and has been placed at the scene of the crime. Furthermore, we consider the suspect "placed" at the scene, if there is convincing DNA evidence *or* there are reliable witnesses testifying to that effect. Of course, the information obtained from all of these sources could be incomplete, inconclusive, or even contradictory, thus we would like our program to take into consideration the "certainty" of various pieces of information gathered.

Suppose that our current suspect, `bob` has a motive, and the DNA evidence suggests that he was at the scene (however the evidence is not conclusive). There are also two witnesses, one who was close to the scene during the time period in question and did not see anyone, and another who claims to have seen someone at the scene looking like `bob` (clearly, the latter claim can only be considered as incomplete or inconclusive information).

Finally, there are two sources providing information about bob's alibi: an unreliable source who claims to have seen bob at a bar during the time period of the crime, and a more reliable source who disputes this alibi.

For simplicity we will use the bilattice $\mathcal{NINE}$ (see Figure 2, with $b = \frac{1}{2}$) for our example and express the bilattice elements as pairs $\langle x, y\rangle$, where $x, y \in \{0, \frac{1}{2}, 1\}$ represent the evidence for and the evidence against a statement, respectively (clearly, in a realistic situation we may want to use a wider range of truth values to represent various degrees of uncertainty). In this context $\frac{1}{2}$ represents "partial" evidence. Now, our program representing the above situation could look something like the following:

1. $charged(x) \leftarrow hasmotive(x) \otimes placed(x) \otimes \neg hasalibi(x)$
2. $placed(x) \leftarrow dna(x) \vee witnessed(x)$
3. $hasmotive(\texttt{bob}) \leftarrow \langle 1, 0\rangle$
4. $hasalibi(\texttt{bob}) \leftarrow \langle \frac{1}{2}, 0\rangle$
5. $hasalibi(\texttt{bob}) \leftarrow \langle 0, 1\rangle$
6. $dna(\texttt{bob}) \leftarrow \langle \frac{1}{2}, 0\rangle$
7. $witnessed(\texttt{bob}) \leftarrow \langle \frac{1}{2}, 0\rangle$
8. $witnessed(\texttt{bob}) \leftarrow \langle 0, 1\rangle$

According to the fixpoint semantics, clauses with the same head are combined using the knowledge join operator, and thus, the final evaluation of $hasalibi(\texttt{bob})$ and $witnessed(\texttt{bob})$, under the intended interpretation, are (in both cases) $\langle 0, 1\rangle \oplus \langle \frac{1}{2}, 0\rangle = \langle \frac{1}{2}, 1\rangle$. Furthermore, $placed(\texttt{bob})$ is evaluated, using the truth operator $\vee$, to $\langle \frac{1}{2}, 0\rangle \vee \langle \frac{1}{2}, 1\rangle = \langle \frac{1}{2}, 0\rangle$. Hence, if $I$ is the intended interpretation (fixpoint), we have the following:

$$\begin{aligned} I(charged(\texttt{bob})) &= \langle 1, 0\rangle \otimes \langle \tfrac{1}{2}, 0\rangle \otimes \neg\langle \tfrac{1}{2}, 1\rangle \\ &= \langle 1, 0\rangle \otimes \langle \tfrac{1}{2}, 0\rangle \otimes \langle 1, \tfrac{1}{2}\rangle \\ &= \langle \tfrac{1}{2}, 0\rangle \otimes \langle 1, \tfrac{1}{2}\rangle \\ &= \langle \tfrac{1}{2}, 0\rangle. \end{aligned}$$

The above example illustrates the utility of knowledge-based logic programming in the presence of incomplete or uncertain information. It also illustrates the use of both knowledge and truth operators in the same program. Of course, the interpretation of the final truth value obtained for a given goal is usually domain or application dependent. For instance, in the above example, we may interpret the truth value $\langle \frac{1}{2}, 0\rangle$ as insufficient evidence to charge bob with the crime.

### 3.4 General Procedural Semantics

Fitting originally introduced a procedural model for logic programs based on a four-valued bilattice which used a version of Smullyan style semantic tableaux [10]. This was later extended to a larger family of the so-called linear bilattices [11, 13]. In contrast, we use a resolution-based procedural semantics which will allow us to start with any formula as a goal and within a uniform framework derive both negative and positive information about that goal representing evidence for or against its truth. In the context of the bilattice $\mathcal{FOUR}$ this means that, if the derivation from a goal $A$ leads to success, then $A$ is at least true,

and if it leads to failure, then $A$ is at least `false`. More generally, if a derivation from $A$ leads to a particular truth-value $b$ in the underlying bilattice, then the evidence supporting the truth or falsity of $A$ is judged to be at least $b$. We say that $A$ has a *$b$-derivation* in this event.

Our procedural model is essentially an extension of SLDNF-resolution. In our derivation trees, each branch of a subtree with the root node $A$, where $A$ is an atom, corresponds to a clause whose head unifies with $A$. Each such clause contributes to the knowledge the system contains about the truth or falsity of $A$. The treatment of $\neg$ under SLDNF-resolution is extended to the operators $\wedge$, $\vee$, $\oplus$, and $\otimes$. More precisely, if during the derivation a subgoal is reached which contains one of these operators, then attempts are made to establish, in parallel, appropriate derivations for the two operands. Substitution unification provides a means of consistently combining the answers obtained for the operands to obtain an answer for the formula itself.

It is easy to see how the following definition can be reformulated to apply to any partially ordered algebra as described in the Introduction. Parts (1) and (2) remain essentially the same, and the symbol $\Box$ of part (3) is allowed to range over all fundamental operations of the algebra, which can be of any finite arity.

**Definition 3.18** Let $\mathcal{B}$ be a distributive bilattice and $b \in \mathcal{B}$. Let $P$ be a program and $G$ a goal. Then $G$ has a *$b$-derivation with answer $\theta$* if

1. $G = c$, where $c \in \mathcal{B} \setminus \{\bot, \top\}$, $b \leq_k c$, and $\theta$ is the identity substitution $\varepsilon$; or
2. $G$ is an atom $A$, and there is a clause $B \leftarrow G' \in P$, with $\sigma = mgu(A, B)$ such that $G'\sigma$ has a $b$-derivation with answer $\theta'$, $\theta = (\sigma\theta')_G$; or
3. $G$ is $\neg G'$, and $G'$ has a $\neg b$-derivation with answer $\theta$; or
4. $G$ is $G_1 \Box G_2$, where $\Box \in \{\oplus, \otimes, \vee, \wedge\}$, and $G_1$ has a $c$-derivation and $G_2$ has a $d$-derivation with answers $\theta_1$ and $\theta_2$, respectively, $\theta = (\theta_1' \odot \theta_2')_G$, where $\theta_i'$ is a variant of $\theta_i$ w.r.t. $G_i$ $(i = 1, 2)$, and $b = c \Box d$.

In part 4 of Definition 3.18, the reason for allowing variants of answers before taking their substitution unification (i.e., $\theta_1' \odot \theta_2'$), is to ensure that variables do not conflict in independent derivations associated with complex subgoals. In order to select the appropriate variant, we can compose the answers with special renaming substitutions which replace the variables in the range of answers by variables which have not occurred in the derivation up to that point. The additional bindings which result from these compositions ensure that the relationships between variables of independent derivations are preserved.

We also adopt the standard process of using suitable variants of program clauses at each step of a derivation. This is so that the variables used for the derivation do not already occur in the derivation up to that point. We will refer to this assumption as the *unique renaming assumption*. One consequence of this assumption is that the answer substitutions obtained are idempotent. Furthermore, by convention, we assume that any goal has a $\bot$-derivation with answer $\varepsilon$.

We can now present the soundness and completeness results which establish the correspondence between the procedural and the fixpoint semantics for logic programs based on arbitrary distributive bilattices.

**Theorem 3.19 (Soundness)** *Let $\mathcal{B}$ be a distributive bilattice and $b \in \mathcal{B}$. Let $P$ be a program, $G$ a goal, and $\theta$ a substitution for the variables of $G$. If $G$ has a $b$-derivation with answer $\theta$, then $(\Phi_P \uparrow \omega)(G\theta) \geq_k b$.*

**Proof:** (By induction on the length of derivations)

1. $G = c \in \mathcal{B}$: Then $c \geq_k b$, and $\theta = \varepsilon$. Now, $I_0(G\theta) = I_0(c) = c \geq_k b$. Since $\Phi_P$ is monotonic, $(\Phi_P \uparrow \omega)(G\theta) = (\Phi_P \uparrow \omega)(c) \geq_k b$.

2. *$G$ is an atom:* Then $P$ must have a clause $A \leftarrow G'$ such that $\sigma = mgu(G, A)$ and $G'\sigma$ has a $b$-derivation with answer $\theta'$ such that $\theta = (\sigma\theta')_G$. By the induction hypothesis, $(\Phi_P \uparrow \omega)(G'\sigma\theta') \geq_k b$. Then,

$$\begin{array}{rcll} (\Phi_P \uparrow \omega)(G\theta) & = & (\Phi_P \uparrow \omega)(G\sigma\theta') & \text{since } G\theta = G\sigma\theta' \\ & = & (\Phi_P \uparrow \omega)(A\sigma\theta') & \text{since } \sigma = mgu(G, A) \\ & \geq_k & (\Phi_P \uparrow \omega)(G'\sigma\theta') & \text{by Lemma 3.17} \\ & \geq_k & b. & \end{array}$$

3. *$G$ is $\neg G'$:* Then $G'$ must have a $\neg b$-derivation with answer $\theta$. By the inductive hypothesis, $(\Phi_P \uparrow \omega)\ (G'\theta) \geq_k \neg b$. Hence,

$$\begin{array}{rcl} (\Phi_P \uparrow \omega)\ (G\theta) & = & (\Phi_P \uparrow \omega)\ (\neg G'\theta) \\ & = & \neg(\Phi_P \uparrow \omega)\ (G'\theta) \\ & \geq_k & b. \end{array}$$

4. *$G$ is $G_1 \sqcap G_2$, where $\sqcap \in \{\oplus, \otimes, \vee, \wedge\}$:* Then there exist $c, d \in \mathcal{B}$ such that $G_1$ has a $c$-derivation and $G_2$ has a $d$-derivation with answers $\theta_1$ and $\theta_2$, respectively such that $\theta = (\theta_1' \odot \theta_2')_G$, and $c \sqcap d = b$, where $\theta_i'$ are variants of $\theta_i$ w.r.t. $G$ $(i = 1, 2)$. Now

$$\begin{array}{rcll} (\Phi_P \uparrow \omega)\ ([G_1 \sqcap G_2]\theta) = (\Phi_P \uparrow \omega)\ (G_1\theta \sqcap G_2\theta) & & & \\ \geq_k & (\Phi_P \uparrow \omega)\ (G_1\theta) \sqcap (\Phi_P \uparrow \omega)\ (G_2\theta), & & \text{by Lemma 3.16(4)} \\ = & (\Phi_P \uparrow \omega)\ (G_1(\theta_1' \odot \theta_2')) \sqcap (\Phi_P \uparrow \omega)\ (G_2(\theta_1' \odot \theta_2')) & & \\ \geq_k & (\Phi_P \uparrow \omega)\ (G_1\theta_1') \sqcap (\Phi_P \uparrow \omega)\ (G_2\theta_2'), & & \text{by Lemma 3.16(5)} \\ \geq_k & (\Phi_P \uparrow \omega)\ (G_1\theta_1) \sqcap (\Phi_P \uparrow \omega)\ (G_2\theta_2), & & \text{by Lemma 3.16(3)} \\ \geq_k & c \sqcap d, & \text{by inductive hypothesis and properties of } \sqcap & \\ = & b. \ \blacksquare & & \end{array}$$

We next present the completeness results for the general semantics. The key to the proof is the following Lifting Lemma.

**Lemma 3.20 (Lifting Lemma)** *Let $\mathcal{B}$ be a distributive bilattice and $b \in \mathcal{B}$. Suppose that $P$ is a program, $G$ a goal, and $\theta$ a substitution for the variables of $G$. Also, suppose $G\theta$ has a $b$-derivation with answer $\eta$, with respect to a program $P$. Then $G$ has a $b$-derivation with answer $\sigma$ such that $G\theta\eta = G\sigma\gamma$, for some substitution $\gamma$.*

**Proof:** (By induction on the length of derivations)

1. $G = c \in \mathcal{B}$: Then $G\theta = c \in \mathcal{B} - \{\bot, \top\}$, and $b \leq_k c$. Hence, $\eta = \varepsilon$ and $G = c$ has a $b$-derivation with answer $\varepsilon$. Clearly, $\theta\eta = \theta\varepsilon = \theta = \varepsilon\theta$. Now, take $\sigma = \varepsilon$ and $\gamma = \theta$.

2. *$G$ is an atom $A$*: Then there exists a clause $B \leftarrow G' \in P$ such that $\sigma' = mgu(A\theta, B)$ and $G'\sigma'$ has a $b$-derivation with answer $\rho$ and $\eta = (\sigma'\rho)_{A\theta}$.

   We consider two cases. If $vars(B) \not\subseteq vars(G')$, then it must be the case that $G' = c \geq_k b$, and thus $G'$ has a $c$-derivation with answer $\rho = \varepsilon$, and $\eta = (\sigma')_{A\theta}$. Since $A\theta$ and $B$ are unifiable via $\sigma'$, by the unique renaming assumption, $A$ and $B$ are unifiable via $\theta\sigma'$. Let $\tau = mgu(A, B)$. Then $\theta\sigma' = \tau\gamma$ for some substitution $\gamma$. But, $G'\tau = c$ has a $c$-derivation with answer $\varepsilon$. Hence, $A$ has a $b$-derivation with answer $\sigma$, where $\sigma = \tau_A$. Furthermore, we have $A\theta\eta = A\theta\sigma' = A\tau\gamma = A\sigma\gamma$.

   On the other hand, suppose $vars(B) \subseteq vars(G')$. Again, since $A\theta$ and $B$ are unifiable via $\sigma'$, by the unique renaming assumption, $A$ and $B$ are unifiable via $\theta\sigma'$. Let $\tau = mgu(A, B)$ and let $\delta$ be a substitution such that $\theta\sigma' = \tau\delta$. Using the unique renaming assumption again, we have $G'\sigma' = G'\theta\sigma' = G'\tau\delta$. Thus, $G'\tau\delta$ has a $b$-derivation with answer $\rho$. By the inductive hypothesis, $G'\tau$ has a $b$-derivation with answer $\alpha$ such that $G'\tau\delta\rho = G'\tau\alpha\gamma$ for some substitution $\gamma$. Then, by the definition of derivation, $A$ has a $b$-derivation with answer $\sigma = (\tau\alpha)_A$. Furthermore, we have:

$$\begin{aligned} A\theta\eta &= A\theta\sigma'\rho \\ &= A\tau\delta\rho \\ &= B\tau\delta\rho \\ &= B\tau\alpha\gamma \quad (\text{since } vars(B) \subseteq vars(G')) \\ &= A\tau\alpha\gamma \\ &= A\sigma\gamma. \end{aligned}$$

   Thus, we have shown that $A$ has a $b$-derivation with answer $\sigma$ such that $A\theta\eta = A\sigma\gamma$.

3. *$G$ is $\neg G'$*: Then $G'\theta$ has a $c$-derivation with answer $\eta$, where $c = \neg b$. By the inductive hypothesis, $G'$ has a $c$-derivation with answer $\sigma$ such that $G'\theta\eta = G'\sigma\gamma$, for some substitution $\gamma$. Hence, $G = \neg G'$ has a $b$-derivation with answer $\sigma$, and $G\theta\eta = G\sigma\gamma$.

4. *$G$ is $G_1 \sqcap G_2$, where $\sqcap \in \{\oplus, \otimes, \vee, \wedge\}$* : Then $G_1\theta \sqcap G_2\theta$ has a $b$-derivation. Hence, $G_1\theta$ has a $c$-derivation with answer $\eta_1$, and $G_2\theta$ has a $d$-derivation with answer $\eta_2$, for some $c, d \in \mathcal{B}$ such that $b = c \sqcap d$, and $\eta = (\eta_1' \odot \eta_2')_{G\theta}$, where $\eta_i'$ is a variant of $\eta_i$ w.r.t. $G_i$ $(i = 1, 2)$. By the inductive hypothesis, $G_1$ has a $c$-derivation with answer $\sigma_1$ such that $G_1\theta\eta_1 = G_1\sigma_1\gamma_1$, for some substitution $\gamma_1$, and $G_2$ has a $d$-derivation with answer $\sigma_2$ such that $G_2\theta\eta_2 = G_2\sigma_2\gamma_2$, for some substitution $\gamma_2$. Let $\sigma_i'$ be a variant of $\sigma_i$ w.r.t. $G_i$ $(i = 1, 2)$ such that $vrange(\sigma_i)$ is disjoint from $vars(G)$ and from $dom(\theta\eta_i')$, and $dom(\sigma_i') \subseteq vars(G_i)$. Hence, all the conditions of Lemma 3.11 are satisfied. It follows that $\sigma_1' \odot \sigma_2'$ exists and there is a substitution $\gamma$ such that

$\theta(\eta_1' \odot \eta_2') = (\sigma_1' \odot \sigma_2')\gamma$. Let $\sigma = (\sigma_1' \odot \sigma_2')_G$. Then $G$ has a $b$-derivation with answer $\sigma$ such that $G\theta\eta = G\sigma\gamma$. ∎

**Theorem 3.21 (Completeness)** *Let $\mathcal{B}$ be a distributive bilattice and $b \in \mathcal{B}$. Let $P$ be a program and $G$ a goal. Suppose $\theta$ is a substitution for the variables of $G$ (without generic constants). If $(\Phi_P \uparrow \omega)\ (G\theta) \geq_k b$, then $G$ has a $b$-derivation with answer $\sigma$ such that $G\theta = G\sigma\gamma$, for some substitution $\gamma$.*

**Proof:** For $b = \bot$, the result is trivial. We prove the result by induction on $n < \omega$, with $b \neq \bot$, assuming $(\Phi_P \uparrow n)(G\theta) \geq_k b$.

**Basis:** $(n = 0)$ First suppose that $(\Phi_P \uparrow 0)(G\theta) = I_0(G\theta) \geq_k b$. We prove the result by a secondary induction on the structure of $G$.

1. *$G$ is an atom*: Then since $I_0(G\theta) \geq_k b$, for any ground substitution $\delta$, $I_0(G\theta\delta) \geq_k b$. By the definition of $I_0$, and since $b \neq \bot$, $G\theta\delta = G = c \geq_k b$, for some $c \in \mathcal{B}$. Then $G$ has a $b$-derivation with answer $\sigma = \varepsilon$. Clearly, $G\theta = G\sigma\theta$.

2. *$G$ is $\neg G'$*: Then $I_0(G'\theta) \geq_k \neg b$. So by the secondary inductive hypothesis, $G'$ has a $\neg b$-derivation with answer $\sigma$ such that $G'\theta = G'\sigma\gamma$, for some substitution $\gamma$. Hence, $G = \neg G'$ has a $b$-derivation with answer $\sigma$ and $G\theta = G\sigma\gamma$.

3. *$G$ is $G_1 \sqsquare G_2$, where $\sqsquare \in \{\oplus, \otimes, \vee, \wedge\}$* : Since $I_0(G_1 \sqsquare G_2)\theta \geq_k b$, for every ground substitution $\delta$, we have that $I_0(G_1\theta\delta \sqsquare G_2\theta\delta) \geq_k b$. Therefore, $I_0(G_1\theta\delta) \geq_k b_1$, and $I_0(G_2\theta\delta) \geq_k b_2$, for some $b_1, b_2 \in \mathcal{B}$ such that $b_1 \sqsquare b_2 = b$. By the secondary induction hypothesis, for $i = 1, 2$, $G_i$ has a $b_i$-derivation with answer $\sigma_i$ such that $G_i\theta\delta = G_i\sigma_i\gamma_i$, for some substitution $\gamma_i$. Let $\sigma_i'$ be a variant of $\sigma_i$ w.r.t $G_i$ $(i = 1, 2)$ such that $vars(G_i\sigma_i')$ and hence also $vrange(\sigma_i')$ are disjoint from $vars(G)$ and from $dom(\theta)$. Hence, by Lemma 3.11, $\sigma_1' \odot \sigma_2'$ exists, and there is a substitution $\gamma$ such that $\theta = (\sigma_1' \odot \sigma_2')\gamma$. Let $\sigma = (\sigma_1' \odot \sigma_2')_G$. It follows that $G$ has a $b$-derivation with answer $\sigma$ such that $G\theta = G\sigma\gamma$.

**Induction:** Assume that the implication holds for the $n$th iteration of $\Phi_P$. Now, suppose that $(\Phi_P \uparrow n + 1)\ (G\theta) \geq_k b$. The result is proved by a secondary induction on the structure of $G$:

1. *$G$ is an atom $A$:* The assumption $(\Phi_P \uparrow n + 1)\ (A\theta) = [\Phi_P(\Phi_P \uparrow n)\ ]\ (A\theta) \geq_k b$, implies that for every ground substitution $\delta$, $[\Phi_P(\Phi_P \uparrow n)\ ]\ (A\theta\delta) \geq_k b$. Then, by the definition of $\Phi_P$ we have:

$$\sum\{(\Phi_P \uparrow n)\ (F\eta) \mid A' \leftarrow F \in P \text{ and } \eta = mgu(A\theta\delta, A')\} \geq_k b.$$

Let $\delta$ be a gc-substitution for $A\theta$. It follows that $P$ must contain a clause $A' \leftarrow F$ such that $\eta = mgu(A\theta\delta, A')$ and $(\Phi_P \uparrow n)\ (F\eta) \geq_k b$. Neither $A\theta$ nor $A'$ contains a generic constant, and, by the unique renaming assumption, $vars(A\theta) \cap (vars(A') \cup vars(F)) = \emptyset$. Thus, by Lemma 3.5, $A\theta$ and $A'$ are unifiable, and, if $\mu = mgu(A\theta, A')$, then $\mu_{A\theta}$ is

injective and variable-pure, and $\delta\eta = \mu\tau$, where $\tau$ is a gc-substitution for $A'\mu$. Thus, since $dom(\delta) \cap vars(F) = \emptyset$ (because $dom(\delta) = vars(A\theta)$), we get $F\mu\tau = F\delta\eta = F\eta$. So $(\Phi_P \uparrow n)(F\mu\tau) \geq_k b$, and hence $(\Phi_P \uparrow n)(F\mu) \geq_k b$ by Lemma 3.15. By the induction hypothesis, $F\mu$ has a $b$-derivation with answer $\varepsilon'$, where $\varepsilon'$ is injective and variable-pure. Hence $A\theta$ has a $b$-derivation with answer $(\mu\varepsilon')_{A\theta}$ which is injective and variable-pure since $\mu_{A\theta}$ and $\varepsilon'$ both are. By the Lifting Lemma we conclude that $A$ has a $b$-derivation with answer $\sigma$ such that $A\theta\mu\varepsilon' = A\sigma\rho$ for some substitution $\rho$. Since $(\mu\varepsilon')_{A\theta}$ is injective and variable-pure, there is a $\mu'$ such that $A\theta\mu\varepsilon'\mu' = A\theta$. Let $\gamma = \rho\mu'$. Then $A\theta = A\sigma\gamma$ as required.

2. *$G$ is $\neg G'$:* Then $(\Phi_P \uparrow n+1)\ (\neg G'\theta) = \neg(\Phi_P \uparrow n+1)\ (G'\theta) \geq_k b$, implying that $(\Phi_P \uparrow n+1)\ (G'\theta) \geq_k \neg b$. Now, by the secondary inductive hypothesis, $G'$ has a $\neg b$-derivation with answer $\sigma$ such that $G'\theta = G'\sigma\gamma$, for some substitution $\gamma$. Hence, $G = \neg G'$ has a $b$-derivation with answer $\sigma$, and $G\theta = G\sigma\gamma$.

3. *$G$ is $G_1 \Box G_2$, where $\Box \in \{\oplus, \otimes, \wedge, \vee\}$:* The derivation of this case is similar to that of the basis case (where $n = 0$), except we replace $I_0$ with $(\Phi_P \uparrow n+1)$ .

Finally, since the induction establishes the result for all $n < \omega$, it also holds for $\omega$. ∎

# 4 Join-irreducible Procedural Semantics

Although the definition of a $b$-derivation results in a sound and complete procedural semantics for logic programming over arbitrary distributive bilattices, it has a drawback. For a given truth value $b \in \mathcal{B}$, the search for a $b$-derivation of a complex goal $G$ may entail searches for $c$-derivations of the subformulas of $G$ for a large number of truth values $c \leq_k b$ that are only remotely related to $b$; moreover, this complexity ramifies as we pass down the parse tree of $G$. It turns out that for finite distributive bilattices (and, more generally, bilattices with the *descending chain property*), we can restrict our attention to $b$-derivations where $b$ ranges over the relatively small subset of $k$-join-irreducible truth-values. The resulting simplification is quite dramatic and may, as we have seen, give an exponential decrease in the search space.

We now present a *join-irreducible* procedural semantics as an alternative to the standard one presented in Section 3.

**Definition 4.1** Let $\mathcal{B}$ be a distributive bilattice, $G$ a formula, and $P$ a program. Suppose that $b \in JIR_k(\mathcal{B})$. Then $G$ has a *$b$-JIR-derivation* with answer $\theta$, if

1. $G = c$, where $c \in \mathcal{B} - \{\bot, \top\}$, $b \leq_k c$, and $\theta = \varepsilon$; or
2. $G$ is an atom that unifies with the head of clause $B \leftarrow G' \in P$, with $\sigma = mgu(G, B)$, and $G'\sigma$ has a $b$-*JIR*-derivation with answer $\theta'$ such that $\theta = (\sigma\theta')_G$; or
3. $G$ is $\neg G'$, and $G'$ has a $\neg b$-*JIR*-derivation with answer $\theta$; or
4. $b \in JIR_k^+(\mathcal{B})$ and

(a) $G = G_1 \vee G_2$ or $G = G_1 \oplus G_2$, and at least one of $G_1$ or $G_2$ has a $b$-*JIR*-derivation with answer $\theta$; or

(b) $G = G_1 \wedge G_2$ or $G = G_1 \otimes G_2$ , and $G_i$ has a $b$-*JIR*-derivation with answer $\theta_i$ $(i = 1, 2)$ such that $\theta = (\theta_1' \odot \theta_2')_G$, where $\theta_i'$ are variants of $\theta_i$ w.r.t. $G_i$; or

5. $b \in JIR_k^-(\mathcal{B})$ and

(a) $G = G_1 \vee G_2$ or $G = G_1 \otimes G_2$ , and $G_i$ has a $b$-*JIR*-derivation with answer $\theta_i$ $(i = 1, 2)$ such that $\theta = (\theta_1' \odot \theta_2')_G$, where $\theta_i'$ are variants of $\theta_i$ w.r.t. $G_i$; or

(b) $G = G_1 \wedge G_2$ or $G = G_1 \oplus G_2$, and at least one of $G_1$ or $G_2$ has a $b$-*JIR*-derivation with answer $\theta$.

Note that one of the consequences of distinguishing between positive and negative k-join-irreducible elements in the above definition is that we have recaptured the duality between the truth operators $\vee$ and $\wedge$. For instance, when searching for a $b$-*JIR*-derivation of a goal $G_1 \vee G_2$, where $b \in JIR_k^+(\mathcal{B})$, we need only look for a $b$-*JIR*-derivation in one of the subgoals. This corresponds to the concept of proof in the more familiar context of four-valued logic programs. On the other hand, when searching for $b$-*JIR*-derivation of $G_1 \vee G_2$, with $b \in JIR_k^-(\mathcal{B})$, we have to look for $b$-*JIR*-derivation for both subgoals. In the four-valued case, this corresponds to the notion of refutation (essentially amounting to disproving the goal). Mathematically, this distinction is explained by Lemma 2.10. Thus, the procedural semantics presented here is a generalization of the more familiar situation in the four-valued case where the truth operators are considered in connection with the canonical join-irreducible elements `true` and `false` [20].

In order to study the connections between the join-irreducible and the general procedural semantics, we need a model of computation that can deal with an arbitrary element $b$ in a distributive bilattice, and not just when $b$ is join-irreducible. This idea is captured in the following definition.

**Definition 4.2** Let $\mathcal{B}$ be a distributive bilattice. Let $a \in \mathcal{B}$, and suppose that $a = b_1 \oplus \cdots \oplus b_n$, where $b_1 \oplus \cdots \oplus b_n$ is a decomposition of $a$ as a join of join-irreducible elements of $\mathcal{B}$ in the knowledge ordering. Let $G$ be a goal and $P$ a program. Then $G$ has an $a$-*JIR*-proof with answer $\theta$, if $G$ has a $b_i$-*JIR*-derivation with answer $\theta_i$ $(1 \leq i \leq n)$ such that $\theta = (\odot\{\theta_1, \cdots, \theta_n\})_G$ (in other words $\theta$ is a representative of the equivalence class of the substitution unifications for $\{\theta_1, \cdots, \theta_n\}$).

Of course, if $\mathcal{B}$ has the Descending Chain Property in the knowledge ordering ($DCP_k$), then, by Theorem 2.2, we can obtain an irredundant decomposition for $a$ in the previous definition.

A system based on the above semantics would, presumably, work as follows. The system is presented with a goal $G$ and a truth value $b$ in the underlying bilattice. The first task, then, would be to obtain a decomposition of $b$ as a join of join-irreducible elements of the bilattice. Assuming that $b_1, b_2, \cdots, b_n$ are the join-irreducible elements thus obtained, the system would then attempt to construct $b_i$-*JIR*-derivations for $G$, for each $i = 1, 2, \cdots, n$. For finite bilattices, using standard representation techniques such as a tabular representation

of bilattice operations (in particular $\oplus$), the join-irreducible decomposition can be obtained in polynomial time in the number of bilattice elements. In this case, the decomposition basically amounts to searching the table for the element to be decomposed and then testing the join-irreducibility of the constituent elements whose join is the original bilattice element. The test for join-irreducibility is a simple linear test to detemine whether the set of all elements strictly less than the element to be tested has a largest member.

If, however, one is interested in obtaining the *maximum* truth value $b$ of a goal $G$ that is implied by the program, then the system must attempt to establish $c$-*JIR*-derivations for each $c \in JIR_k(\mathcal{B})$. The maximum truth value desired is the join of all such join-irreducible elements with successful derivations.

**Example:** Consider the following program, $P$, based on the bilattice $\mathcal{NINE}$ (see Figure 2):

1. $p \leftarrow q \otimes r$
2. $q \leftarrow t \vee s$
3. $r \leftarrow u \wedge s$
4. $s \leftarrow \langle b, 0\rangle$
5. $t \leftarrow \langle 1, b\rangle$
6. $u \leftarrow \langle 0, b\rangle$.

Suppose that the query consists of the goal $p$ and a truth value $\langle 1, b\rangle$, i.e., we would like to obtain a $\langle 1, b\rangle$-*JIR*-derivation for $p$. First, recall that the k-join-irreducible elements in the bilattice $\mathcal{NINE}$ are $\langle 1, 0\rangle$, $\langle 0, 1\rangle$, $\langle b, 0\rangle$, and $\langle 0, b\rangle$. The system will obtain a decomposition of $\langle 1, b\rangle$ as a join of join-irreducibles: $\langle 1, b\rangle = \langle 1, 0\rangle \oplus \langle 0, b\rangle$. It must then obtain a $\langle 1, 0\rangle$-*JIR*-derivation and a $\langle 0, b\rangle$-*JIR*-derivation for $p$. The complete computation tree for the goal $p$ is depicted in Figure 3. The dotted lines in the figure represent recursive steps in the definition of *JIR*-derivation. Let us consider the steps involved in the construction of a $\langle 1, 0\rangle$-*JIR*-derivation. To obtain such a derivation for the subgoal $q \otimes r$ entails obtaining $\langle 1, 0\rangle$-*JIR*-derivations for both $q$ and $r$. The derivation for $q$ in turn, requires a $\langle 1, 0\rangle$-*JIR*-derivation for $t \vee s$. However, since $\langle 1, 0\rangle \in JIR_k^+(\mathcal{NINE})$, the system now needs to construct a $\langle 1, 0\rangle$-*JIR*-derivation for only one of the subgoals. Indeed such a derivation is obtained, as $\langle 1, b\rangle \geq_k \langle 1, 0\rangle$. In this example we can also obtain a $\langle 0, b\rangle$-*JIR*-derivation for $r$. This would again require the construction of a similar derivation for only one of $u$ or $s$ in the subgoal $u \wedge s$; this time because $\langle 0, b\rangle \in JIR_k^-(\mathcal{NINE})$. ∎

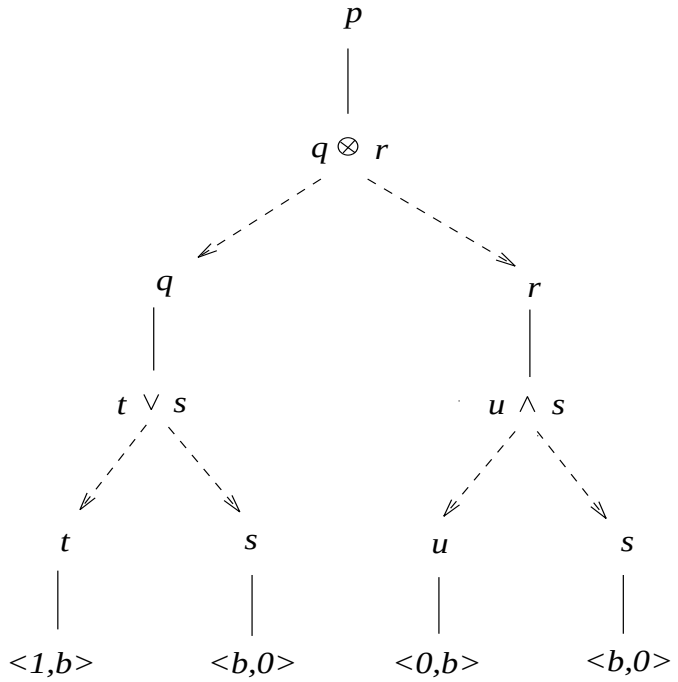
In the rest of this section we formally describe the relationship between the two bilattice-based procedural semantics. The following two technical lemmas will be useful in the sequel.

**Lemma 4.3** *Let $\mathcal{B}$ be a distributive bilattice, $P$ a program, and $G$ a goal. Suppose that $c \in JIR_k(\mathcal{B})$. If $G$ has a $c$-JIR-derivation with answer $\theta$, then $G$ has a $b$-JIR-derivation with answer $\theta$ for every $b \leq_k c$, where $b \in JIR_k(\mathcal{B})$.*

**Proof:** Let $b$ be an arbitrary element of $JIR_k(\mathcal{B})$ such that $b \leq_k c$. The result is proved by induction on the length of derivations.

1. If $G = d$, for some $d \in \mathcal{B} - \{\bot, \top\}$ such that $c \leq_k d$, and $\theta = \varepsilon$, then since, $b \leq_k c \leq_k d$, $G$ has a $b$-*JIR*-derivation with answer $\theta = \varepsilon$.

Figure 3: The Complete Computation Tree for the goal $p$




2. *G is an atom:* Since $G$ has a $c$-*JIR*-derivation with answer $\theta$, there is a clause $B \leftarrow G' \in P$, with $\sigma = mgu(G, B)$ such that $G'\sigma$ has a $c$-*JIR*-derivation with answer $\theta'$, and $\theta = (\sigma\theta')_G$. By the inductive hypothesis, $G'\sigma$ has a $b$-*JIR*-derivation with answer $\theta'$. By definition, $G$ has a $b$-*JIR*-derivation with answer $\theta$.

3. *G is* $\neg G'$*:* Then $G'$ has a $\neg c$-*JIR*-derivation with answer $\theta$. Since $b \leq_k c$, we also have $\neg b \leq_k \neg c$ and $\neg b, \neg c \in JIR_k(\mathcal{B})$. By the inductive hypothesis, $G'$ has a $\neg b$-*JIR*-derivation with answer $\theta$. Thus, by the definition of *JIR*-derivation $G$ has a $b$-*JIR*-derivation with answer $\theta$.

4. *G is* $G_1 \wedge G_2$ *:* There are two subcases.

   (a) $c \in JIR_k^+(\mathcal{B})$ : Then $G_i$ has a $c$-*JIR*-derivation with answer $\theta_i$ such that $\theta = (\theta_1' \odot \theta_2')_G$, where $\theta_i'$ is a variant of $\theta_i$ w.r.t. $G_i$ $(i = 1, 2)$. By the inductive hypothesis, $G_i$ has a $b$-*JIR*-derivation with answer $\theta_i$. By Lemma 2.11, $b \in JIR_k^+(\mathcal{B})$ . By the definition of *JIR*-derivation, $G = G_1 \wedge G_2$ has a $b$-*JIR*-derivation with answer $\theta$.

   (b) $c \in JIR_k^-(\mathcal{B})$ : Then $G_1$ or $G_2$ (say $G_1$) has a $c$-*JIR*-derivation with answer $\theta$. By the inductive hypothesis, $G_1$ has a $b$-*JIR*-derivation with answer $\theta$, and by the definition of *JIR*-derivation, $G = G_1 \wedge G_2$ has a $b$-*JIR*-derivation with answer $\theta$.

5. *G is* $G_1 \vee G_2$ *:* This case is dual to the case when $G = G_1 \wedge G_2$ .

6. *G is* $G_1 \otimes G_2$ *:* This case is similar to 4(a).

7. *$G$ is $G_1 \oplus G_2$:* This case is similar to 4(b). ∎

**Lemma 4.4** *Let $\mathcal{B}$ be a distributive bilattice with the $DCP_k$. Let $P$ be a program and $G$ a goal. Suppose that $c \in \mathcal{B}$, and $b \in JIR_k(\mathcal{B})$, $b \leq_k c$. If $G$ has a $c$-JIR-proof with answer $\theta$, then $G$ has a $b$-JIR-derivation with answer $\sigma$ such that $G\theta = G\sigma\gamma$, for some substitution $\gamma$.*

**Proof:** Suppose that $G$ has a $c$-*JIR*-proof with answer $\theta$. Since $\mathcal{B}$ satisfies the $DCP_k$, we can write $c = c_1 \oplus \cdots \oplus c_n$ such that $c_i \in JIR_k(\mathcal{B})$, for $i = 1, 2, \cdots, n$. Furthermore, $G$ has a $c_i$-*JIR*-derivation with answer $\theta_i$ such that $\theta = (\odot\{\theta_1, \cdots, \theta_n\})_G$.

Now, $b \leq_k c = c_1 \oplus \cdots \oplus c_n$ . By Lemma 2.1, for some $j$ $(1 \leq j \leq n)$, $b \leq_k c_j$, and, by Lemma 4.3, $G$ has a $b$-*JIR*-derivation with answer $\theta_j$. Furthermore, note that $\odot\{\theta_1, \cdots, \theta_n\} = \theta_j\alpha$, where $\alpha = mgsu(\{\theta_1, \cdots, \theta_n\})$. Putting things together, taking $\sigma = \theta_j$ and $\gamma = \alpha$, we have $G\theta = G(\odot\{\theta_1, \cdots, \theta_n\}) = G\theta_j\alpha = G\sigma\gamma$. ∎

We would like to show that the new join-irreducible procedural semantics is indeed complete with respect to the bilattice-based fixpoint semantics. We will obtain such a completeness result as a corollary of the Completeness Theorem for the general semantics presented in the previous section. In order to accomplish this task we will need the next lemmas which describes the precise connection between the two semantics in the context of bilattices with the $DCP_k$.

**Lemma 4.5** *Let $\mathcal{B}$ be a distributive bilattice with the $DCP_k$. Let $P$ be a program, $G$ a goal, and $c \in \mathcal{B}$. If $G$ has a $c$-derivation with answer $\theta$, then $G$ has a $c$-JIR-proof with answer $\theta'$ such that $G\theta = G\theta'\gamma$, for some substitution $\gamma$.*

**Proof:** (By induction on the length of derivations)

1. If $G = b$, for some $b \in \mathcal{B} \setminus \{\bot, \top\}$ such that $c \leq_k b$, and $\theta = \varepsilon$, then since $\mathcal{B}$ has the $DCP_k$, we can write $c = c_1 \oplus \cdots \oplus c_n$, the decomposition of $c$ as join of join-irreducibles, i.e., $c_i \in JIR_k(\mathcal{B})$. Then $c_i \leq_k b$, and by the definition of *JIR*-derivation, $G$ has a $c_i$-*JIR*-derivation with answer $\theta = \varepsilon$ $(1 \leq i \leq n)$. Hence, $G$ has a $c$-*JIR*-proof with answer $\theta' = \odot\{\varepsilon, \cdots, \varepsilon\} = \varepsilon$.

2. *$G$ is an atom*: Then there is a clause $B \leftarrow G' \in P$, with $\sigma = mgu(G, B)$ such that $G'\sigma$ has a $c$-derivation with answer $\delta$, and $\theta = (\sigma\delta)_G$. By the inductive hypothesis, $G'\sigma$ has a $c$-*JIR*-proof with answer $\delta'$, such that $G'\sigma\delta = G'\sigma\delta'\tau$, for some substitution $\tau$. Since $\mathcal{B}$ has the $DCP_k$, $c$ has an irredundant decomposition $b_1 \oplus \cdots \oplus b_n$ , where $b_i \in JIR_k(\mathcal{B})$. Furthermore, for each $i$, $G'\sigma$ has a $b_i$-*JIR*-derivation with answer $\delta'_i$ such that $\delta' = (\odot\{\delta'_1, \cdots, \delta'_n\})_{G'\sigma}$. By the definition of *JIR*-derivation, $G$ has a $b_i$-*JIR*-derivation with answer $\alpha_i = (\sigma\delta'_i)_G$; by the unique renaming assumption, $\sigma\delta'_i$ and $\alpha_i$ are idempotent, implying that $\alpha_i \leq \sigma\delta'_i$ $(1 \leq i \leq n)$. By Lemma 3.10, the $\alpha_i$ are unifiable and therefore $G$ has a $c$-*JIR*-proof with answer $\theta' = (\odot\{\alpha_1, \cdots, \alpha_n\})_G$. Also by Lemma 3.10, there exists a substitution $\beta$ such that

$$\odot\{\sigma\delta'_1, \cdots, \sigma\delta'_n\} = \odot\{\alpha_1, \cdots, \alpha_n\}\beta.$$

Finally, we must show that $G\theta = G\theta'\gamma$, for some substitution $\gamma$. By Lemma 3.8, there exists a substitution $\rho$ such that $\odot\{\sigma\delta'_1, \cdots, \sigma\delta'_n\}\rho = \sigma(\odot\{\delta'_1, \cdots, \delta'_n\})$.

Then, letting $\gamma = \beta\rho\tau$ we have:

$$\begin{aligned} G\theta'\gamma &= G[\odot\{\alpha_1, \cdots, \alpha_n\}]\beta\rho\tau \\ &= G[\odot\{\sigma\delta'_1, \cdots, \sigma\delta'_n\}]\rho\tau \\ &= G\sigma[\odot\{\delta'_1, \cdots, \delta'_n\}]\tau \\ &= G\sigma\delta'\tau \qquad\qquad (\text{since } vars(G\sigma) \subseteq\ vars(G'\sigma)) \\ &= G\sigma\delta = G\theta. \end{aligned}$$

3. *$G$ is $G_1 \wedge G_2$* : Then, $G_j$ has a $c_j$-derivation with answer $\theta_j$ such that $c = c_1 \wedge c_2$ and $\theta = (\eta_1 \odot \eta_2)_G$, where $\eta_j$ is a variant of $\theta_j$ w.r.t. $G_j$ $(j = 1, 2)$. By the inductive hypothesis, $G_j$ has a $c_j$-*JIR*-proof with answer $\theta'_j$ such that $G_j\theta_j = G_j\theta'_j\gamma_j$, for some substitution $\gamma_j$. Applying Lemma 3.11 (with $\theta = \varepsilon$), we conclude that $\theta'_1$ and $\theta'_2$ are unifiable and that $\theta'_1 \odot \theta'_2 \leq \eta_1 \odot \eta_2$. Note that the first 3 conditions of Lemma 3.11 hold by appropriate use of the unique renaming assumption. Condition 4 holds because $G_j\theta'_j \leq G_j\theta_j \leq G_j\eta_j$ $(j = 1, 2)$. Condition 5 follows since $\eta_1$ and $\eta_2$ are unifiable. Thus we have $G\theta = G(\eta_1 \odot \eta_2) = G(\theta'_1 \odot \theta'_2)\gamma'$, for some substitution $\gamma'$. Since $\mathcal{B}$ has the $DCP_k$, we can write $c = b_1 \oplus \cdots \oplus b_n$, where $b_i \in\ JIR_k(\mathcal{B})$, for $1 \leq i \leq n$. Hence, $b_i \leq_k c_1 \wedge c_2$. Now, for each $b_i$ we consider two cases.

   (a) $b_i \in JIR_k^+(\mathcal{B})$ : In this case by Lemma 2.10, $b_i \leq_k c_j$, and by Lemma 4.4, $G_j$ has a $b_i$-*JIR*-derivation with answer $\theta_j^i$ $(j = 1, 2)$ such that $G_j\theta'_j = G_j\theta_j^i\gamma_j^i$. Without loss of generality assume that $\theta_j^i$ is idempotent such that the conditions of Lemma 3.11 are satisfied, i.e., $\theta_1^i$ and $\theta_2^i$ are unifiable and $\theta_1^i \odot \theta_2^i \leq \theta'_1 \odot \theta'_2$. By the definition of *JIR*-derivation, $G = G_1 \wedge G_2$ has a $b_i$-*JIR*-derivation with answer $\phi_i = (\tau_1^i \odot \tau_2^i)_G$, where $\tau_j^i$ is a variant of $\theta_j^i$ w.r.t. $G_j$ $(j = 1, 2)$. By another application of Lemma 3.11, $\tau_1^i \odot \tau_2^i \leq \theta'_1 \odot \theta'_2$, and hence $G(\theta'_1 \odot \theta'_2) = G(\theta_1^i \odot \theta_2^i)\beta^i = G(\tau_1^i \odot \tau_2^i)\gamma^i = G\phi_i\gamma^i$, for some substitutions $\beta^i$ and $\gamma^i$.

   (b) $b_i \in JIR_k^-(\mathcal{B})$ : In this case by Lemma 2.10, $b_i \leq_k c_j$, for $j = 1$ or $j = 2$ (assume $b_i \leq_k c_1$), and by Lemma 4.4, $G_1$ has a $b_i$-*JIR*-derivation with answer $\theta_1^i$ such that $G_1\theta'_1 = G_1\theta_1^i\gamma^i$, for some substitution $\gamma^i$. Hence, by the definition of *JIR*-derivation, $G = G_1 \wedge G_2$ has a $b_i$-*JIR*-derivation with answer $\phi_i = \theta_1^i$, and $G\theta'_1 = G\phi_i\gamma^i$.

   From the unique renaming assumption we can deduce that $\tau_1^i \odot \tau_2^i$ is idempotent (as is $\theta_1^i \odot \theta_2^i$). Since $\theta_1^i, \theta_2^i, \tau_1^i \odot \tau_2^i \leq \theta'_1 \odot \theta'_2$, it follows by Lemma 3.10 that $\phi_1, \cdots, \phi_n$ are unifiable and that $\odot\{\phi_1, \cdots, \phi_n\}\rho = \theta'_1 \odot \theta'_2$, for some substitution $\rho$.

   Hence, $G$ has a $c$-*JIR*-proof with answer $\theta' = (\odot\{\phi_1, \cdots, \phi_n\})_G$. Now, letting $\gamma = \rho\gamma'$, we have $G\theta = G(\eta_1 \odot \eta_2) = G(\theta'_1 \odot \theta'_2)\gamma' = G\theta'\rho\gamma' = G\theta'\gamma$.

4. *$G$ is $G_1 \otimes G_2$* : This case is similar to 3(a).

5. *$G$ is $G_1 \vee G_2$* : This case is the dual of 3.

6. *G is* $G_1 \oplus G_2$: This case is similar to 3(b).

7. *G is* $\neg G'$: Straightforward from the definitions. ∎

The join-irreducible procedural semantics, like the general procedural semantics, is sound with respect to the fixpoint semantics over bilattices. We first state this result for the join-irreducible elements of the bilattice in the following lemma and then generalize this result to the arbitrary elements of the bilattice. The proof of this lemma is similar to that of the Soundness Theorem for the general semantics (Theorem 3.19) and is omitted.

**Lemma 4.6** *Suppose that* $\mathcal{B}$ *is a distributive bilattice. Let* $P$ *be a program,* $G$ *a goal, and* $b \in JIR_k(\mathcal{B})$*. If* $G$ *has a* $b$*-JIR-derivation with answer* $\theta$*, then* $(\Phi_P \uparrow \omega)(G\theta) \geq_k b$*.*

The Join-irreducible Soundness Theorem generalizes the above lemma to arbitrary elements of the bilattice.

**Theorem 4.7 (Join-Irreducible Soundness)** *Suppose that* $\mathcal{B}$ *is a distributive bilattice with the* $DCP_k$*. Let* $P$ *be a program,* $G$ *a goal, and* $b \in \mathcal{B}$*. If* $G$ *has a* $b$*-JIR-proof with answer* $\theta$*, then* $(\Phi_P \uparrow \omega)(G\theta) \geq_k b$*.*

**Proof:** Let $b = b_1 \oplus \cdots \oplus b_n$ be a decomposition of $b$ as join of join-irreducibles. By definition, $G$ has a $b_i$-*JIR*-derivation with answer $\theta_i$ such that $\theta = (\odot\{\theta_1, \theta_2, \cdots, \theta_n\})_G$. By Lemma 4.6, $(\Phi_P \uparrow \omega)(G\theta_i) \geq_k b_i$, and by Lemma 3.16(1), $(\Phi_P \uparrow \omega)(G\theta) \geq_k b_i$ $(1 \leq i \leq n)$. Then, $(\Phi_P \uparrow \omega)(G\theta) \geq_k b_1 \oplus \cdots \oplus b_n = b$. ∎

The next major result of this section is the completeness theorem for the join-irreducible semantics. It is obtained as a corollary of the Completeness Theorem for the general procedural semantics (Theorem 3.21) and Lemma 4.5.

**Theorem 4.8 (Join-Irreducible Completeness)** *Let* $\mathcal{B}$ *be a distributive bilattice with* $DCP_k$*, and* $b \in \mathcal{B}$*. Let* $P$ *be a program,* $G$ *a goal, and* $\theta$ *a substitution for the variables of* $G$*. If* $(\Phi_P \uparrow \omega)\ (G\theta) \geq_k b$*, then* $G$ *has a* $b$*-JIR-proof with answer* $\sigma$*, such that* $G\theta = G\sigma\gamma$*, for some substitution* $\gamma$*.*

**Proof:** Suppose that $(\Phi_P \uparrow \omega)(G\theta) \geq_k b$, then by Theorem 3.21 $G$ has a $b$-derivation with answer $\sigma'$ such that $G\theta = G\sigma'\gamma'$, for some substitution $\gamma'$. Now, by Lemma 4.5, $G$ has a $b$-*JIR*-proof with answer $\sigma$ such that $G\sigma' = G\sigma\gamma''$, for some substitution $\gamma''$. Now: $G\theta = G\sigma'\gamma' = G\sigma\gamma''\gamma' = G\sigma\gamma$, where $\gamma = \gamma''\gamma'$. ∎

Finally, we summarize, in the following theorem, the precise connection between the join-irreducible semantics and the general semantics presented in the previous section. The proof is a straightforward consequence of Lemma 4.5, the Join-irreducible Soundness Theorem, and Theorem 3.21.

**Theorem 4.9** *Let $\mathcal{B}$ be a distributive bilattice with the $DCP_k$. Let $P$ be a program and $G$ a goal. Suppose that $c \in \mathcal{B}$ .*

1. *If $G$ has a c-derivation with answer $\theta$, then $G$ has a c-JIR-proof with answer $\theta'$ such that $G\theta = G\theta'\gamma$, for some substitution $\gamma$; and*
2. *If $G$ has a c-JIR-proof with answer $\theta$, then $G$ has a c-derivation with answer $\theta'$ such that $G\theta = G\theta'\gamma$, for some substitution $\gamma$.* ∎

# 5 Summary and Conclusions

In this paper we have introduced a new procedural semantics for a broad family of multi-valued logic programming languages, based on the join-irreducible elements in the knowledge component of the underlying bilattice. The join-irreducible elements are significant in two ways. First, they help clarify the underlying relationship between the truth values in bilattices and the operational behavior of bilattice-based logic programs. This underlying relationship is explained by Lemma 2.10. Secondly, by concentrating on this smaller set of representative bilattice elements, we can reduce the overall complexity of the logic programming semantics. We have shown that, in fact, the join-irreducible elements of a bilattice can constitute an even smaller representative set compared to other representation mechanisms previously studied in the literature (see Lemmas 2.7 and 2.9). Other main results of the paper are the Soundness and Completeness Theorems for the join-irreducible logic programming semantics.

The procedural semantics presented in the paper makes use of a parallel computation model for evaluation of queries. We have used the notion of substitution unification to deal with variable sharing among independent subgoals. Substitution unification provides a non-equational and algebraic approach to the query evaluation process in parallel logic programming languages. A study of the properties of substitution unification and their relevance will be presented elsewhere [23].

Closer examination of the fixpoint and the procedural semantics presented in this paper, suggests that the only properties of the distributive bilattice used in our results are that (a) it forms a complete lattice under the knowledge ordering and (b) each of its operations distributes over an ifinite join in each argument. There is a standard construction from lattice theory by which every partially ordered algebra whose operations are monotone in each argument can be naturally embedded in another algebra of this kind that has exactly these crucial properties of a bilattice. We believe that the main results presented in Section 3, and possibly also many of the results involving join-irreducible elements in Section 4, can be extended to more general truth-value algebras based on the above observations. These ideas will be pursued elsewhere [24].

# References

[1] O. Arieli and A. Avron, Reasoning with logical bilattices, *Journal of Logic, Language, and Information*, to appear (1995).

[2] K. R. Apt and M. H. van Emden, Contribution to the theory of logic programming, *JACM*, **29** (1982), pp. 841-862.

[3] N. D. Belnap, Jr. A useful four-valued logic, in *Modern Uses of Multiple-Valued Logic*, J. Michael Dunn and G. Epstein editors, D. Reidel, Boston (1977), pp. 8-37.

[4] G. Birkhoff, *Lattice Theory*, third edition, Amer. Math. Soc., Providence, Rhode Island, (1967).

[5] K. L. Clark, Negation as failure, in *Logic and Data Bases*, H. Gallaire and J. Minker editors, Plenum Press, New York (1978), pp. 293-322.

[6] J. S. Conery and D. F. Kibler, AND parallelism and nondeterminism in logic programs, *New Generation Computing*, **3** (1985), pp. 43-70.

[7] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge University Press, Cambridge, Mass. (1990).

[8] E. Eder, Properties of substitutions and unification, *Journal of Symbolic Computation*, **1** (1985), pp. 31-46.

[9] M. C. Fitting, Logic Programming on a Topological Bilattice, *Fundamenta Informatica* **11** (1988), pp. 209-218.

[10] M. C. Fitting, Negation as refutation, in *Proceedings of the Fourth Annual Symposium on Logic in Computer Science*, R. Parikh editor, IEEE (1978), pp. 63-70.

[11] M. C. Fitting, Bilattices in logic programming, in *The Twentieth International Symposium on Multiple-Valued Logic*, G. Epstein editor, IEEE (1990), pp. 63-70.

[12] M. C. Fitting, Bilattices and semantics of logic programming, *Journal of Logic Programming*, **11** (1991), pp. 91-116.

[13] M. C. Fitting, Tableaux for logic programming, *Journal of Logic Programming*, **3** (1994), pp. 175-188.

[14] M. L. Ginsberg, Multi-valued logics: a uniform approach to reasoning in artificial intelligence, *Computational Intelligence*, **4** (1988), pp. 265-316.

[15] J. M. Jacquet, *Conclog: A Methodological Approach to Concurrent Logic Programming*, Springer-Verlag, Berlin (1991).

[16] J. L. Lassez and M. J. Maher and K. Marriott, Unification revisited, in *Foundations of Deductive Databases and Logic Programming*, J. Minker editor, Morgan Kaufmann, Los Altos, CA (1988), pp. 587-625.

[17] L. V. S. Lakshmanan and F. Sadri, Probabilistic deductive database, in *Proceedings of 1994 International Logic Programming Symposium*, M. Bruynooghe editor, MIT Press (1994), pp. 254-268.

[18] J. W. Lloyd, *Foundations of Logic Programming*, second edition, Springer, Berlin (1987).

[19] B. Mobasher, *Generalized Knowledge-based Semantics for Multivalued Logic Programs*, Ph.D. Dissertation, Iowa State University, Ames, Iowa (1994).

[20] B. Mobasher, J. Leszczylowski, G. Slutzki, and D. Pigozzi, Negation as Partial Failure, in *Proceedings of the Second International Workshop on Logic Programming and Non-monotonic Reasoning*, L. M. Pereira and A. Nerode editors, MIT Press (1993), pp. 244-262.

[21] B. Mobasher, J. Leszczylowski, D. Pigozzi, and G. Slutzki, A knowledge-based procedural semantics for logic programming, Technical Report TR#93-15, Department of Computer Science, Iowa State University, May 1993.

[22] B. Mobasher, D. Pigozzi, and G. Slutzki, Algebraic semantics for knowledge-based logic programs, in *Proceedings of the Workshop on Uncertainty in Databases and Deductive Systems*, L. V. S. Lakshmanan editor, Ithaca, New York (1994), pp. 13-24.

[23] B. Mobasher, D. Pigozzi, and G. Slutzki, Properties of Substitution Unification: A Non-equational Approach to Parallel Query Evaluation, *in preparation.*

[24] B. Mobasher, D. Pigozzi, and G. Slutzki, Reasoning with partially ordered truth-value algebras, *in preparation.*

[25] C. Palamidessi, Algebraic properties of idempotent substitutions, in *Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, M. S. Paterson editor, Springer-Verlag, Berlin (1990), pp. 386-399.

[26] R. Reiter, A Logic for Default Reasoning, *Artificial Intelligence* **13** (1980), pp. 81-132.

[27] P. Ruet and F. Fages, Combining explicit negation and negation as failure via Belnap's logic, in *Proceedings of the Workshop on Uncertainty in Databases and Deductive Systems*, L. V. S. Lakshmanan editor Ithaca, New York (1994), pp. 1-12.

[28] A. Takeuchi, *Parallel Logic Programming*, John Wiley and Sons, Inc. (1992).

[29] M. van Emden, Quantitative deduction and its fixpoint theory, *Journal of Logic Programming* **3** (1986), pp. 37-53.

[30] M. van Emden and R. A. Kowalski, The semantics of predicate logic as a programming language, *JACM*, **23** (1976), pp. 733-742.