Algebraic Semantics for Knowledge-based Logic Programs

Bamshad Mobasher[1]
Dept. of Computer Science
Univ. of Minnesota, Morris
mobasher@cda.mrs.umn.edu

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

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

1 Introduction

In this paper we present an operational 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 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.

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 monotoniciy 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 operational 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 in a particular logic program or deductive database. The first logic of this kind originated with Belnap [2]. 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 [11] 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 that seem to best model the knowledge-truth interaction. These ideas were pursued by Fitting [9, 10] in the context of logic programming semantics. In [15] we developed a knowledge-based operational semantics based on the 4-valued Belnap bilattice and proved a soundness and completeness theorem for it with respect to Fitting's declarative fixpoint semantics. A novel feature of the operational 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.

[1]Direct all correspondence to Bamshad Mobasher, Dept. of Computer Science, University of Minnesota, Morris, MN 56267. Email: mobasher@cda.mrs.umn.edu.

In this paper the operational semantics in [15] is first generalized to an arbitrary distributive bilattice. We introduce the notion of a *b-proof* for each element of the bilattice except $\top$ and $\bot$. (In the 4-element case `true`-proofs coincide with proofs and a `false`-proofs with refutations.) We prove a soundness and completeness theorems for this procedural semantics, again with respect to Fitting's declarative fixpoint semantics.

Although the resulting logic programming system is quite satisfactory in some respects, for example the symmetry between truth and refutation in the 4-element case is carried over and the mathematical theory is quite smooth, it has some unpleasant defects. For a given truth value b, the search for a b-proof of a complex goal G may entail searches for c-proofs 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 essentially restrict our attention to b-proofs where b ranges over a relatively small subset of special truth-values. Moreover, in the search for a b-proof for G, we need only look for b-proofs of subformulas of G. These special truth values turn out to be the so-called *join irreducible* elements of the knowledge part of the bilattice. Ginsberg [11] 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 class of bilattices.

We present a *join-irreducible* operational semantics as an alternative to the standard one. The join-irreducible semantics, which represents the most novel feature of this paper, 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 Background and Preliminaries

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 largest and smallest elements of L, if they exist, are 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).

The following well-known theorem, due to Birkhoff, shows that for distributive lattices with the DCP, 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 2.1 (Birkhoff [3]). *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, $\wedge$ and $\vee$ the meet and join, and $\bigwedge$ and $\bigvee$ the infinitary meet and join (if they exist). 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 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.

Definition 2.2. (Ginsberg [11]) A *bilattice* is a structure $\langle \mathcal{B}, \leq_t, \leq_k, \neg \rangle$ consisting of a non-empty 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 $x, y \in \mathcal{B}$;

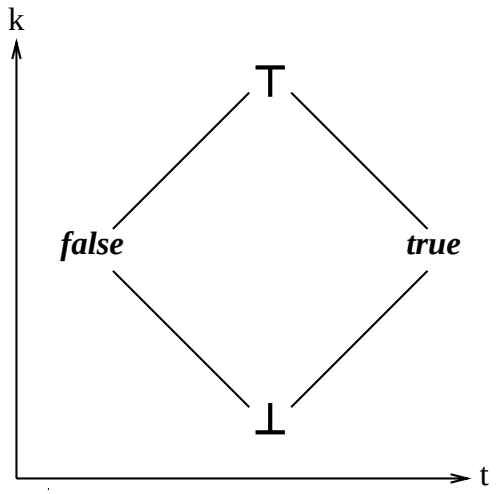

Figure 1: The bilattice $\mathcal{FOUR}$

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* [9, 10] 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. For the remainder of this paper we assume that all bilattices under consideration satisfy the interlacing conditions. 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.

In [15] we gave a natural procedural semantics for logic programs based on Belnap's four-valued logic [2], 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 [18], probabilistic logics [20], Kripke's intuitionistic logic model [7], and modal logics based on the many-worlds semantics [11]. For a more detailed discussion see [10, 11].

In this paper, based on the notion of join-irreducibility in a bilattice, we introduce a new algebraic operational semantics for logic programming over arbitrary distributive bilattices that satisfy certain finiteness conditions. Ginsberg [11] showed that every distributive bilattice can be represented as the special kind of 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 2.3. Let $\langle L, \leq \rangle$ be a lattice. Define binary relations $\leq_t$ and $\leq_k$ on $L \times L$ and a unary operation $\neg : L \times L \rightarrow L \times L$ by:

1. $\langle x_1, x_2 \rangle \leq_t \langle y_1, y_2 \rangle$ if $x_1 \leq y_1$ and $y_2 \leq x_2$,

2. $\langle x_1, x_2\rangle \leq_k \langle y_1, y_2\rangle$ if $x_1 \leq y_1$ and $x_2 \leq y_2$.

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

The structure $\langle L \times L, \leq_t, \leq_k, \neg\rangle$ is denoted by $\mathcal{B}(L)$.

Theorem 2.4 (Ginsberg [11],Fitting [9]). *Let L be a complete lattice. Then $\mathcal{B}(L)$ is a bilattice and is distributive if L is distributive. Conversely, every distributive bilattice is isomorphic to some $\mathcal{B}(L)$ for some complete, distributive lattice L.*

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 $\{0, 1\}$. The probabilistic bilattice can be formed by taking for L the interval [0,1]. 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 in a $\neg$ is an automorphism of the knowledge lattice, $\neg b$ will be k-join-irreducible whenever b is k-join-irreducible.

In [11], Ginsberg discussed the ramifications of reducing the complexity of bilattice based inference systems by reducing bilattices to a smaller set of representative 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 2.5. (Ginsberg [11]) An element x of a bilattice $\mathcal{B}$ is *t-grounded* if, for any $y \in \mathcal{B}$, $y \geq_t x \Rightarrow y \geq_k x$; and it is *f-grounded* if, for any $y \in \mathcal{B}$, $y \leq_t x \Rightarrow y \geq_k x$. 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 $\mathcal{B}$. A bilattice is called *grounded* if it is grounded at every point.

Given Ginsberg's observation mentioned above, we can also characterize grounded elements of a bilattice as those that either have a positive or a negative component, but not both. More specifically, 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 $x_1 = \bot$. 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 2.6. 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 2.7. *Let $\mathcal{B} = \mathcal{B}(L) = \langle L \times L, \leq_t, \leq_k, \neg\rangle$ be a bilattice, where $\langle L, \leq\rangle$ is a complete lattice. Then*

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

It is easy to verify that, in fact, $JIR_k(\mathcal{B}) = JIR_k^+(\mathcal{B}) \cup JIR_k^-(\mathcal{B})$.

To illustrate the above 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 $0 \leq b \leq 1$, and obtaining the structure $\mathcal{B}(P, P)$. Then

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

and

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

Note that in this bilattice, $\langle 0, 1\rangle$ represents `false` while $\langle 1, 0\rangle$ represents `true`. In fact, in any bilattice $\mathcal{B}$, the elements `false` and `true` are among the k-join-irreducible elements (`false` $\in JIR_k^-(\mathcal{B})$ and `true` $\in JIR_k^+(\mathcal{B})$), if the top element

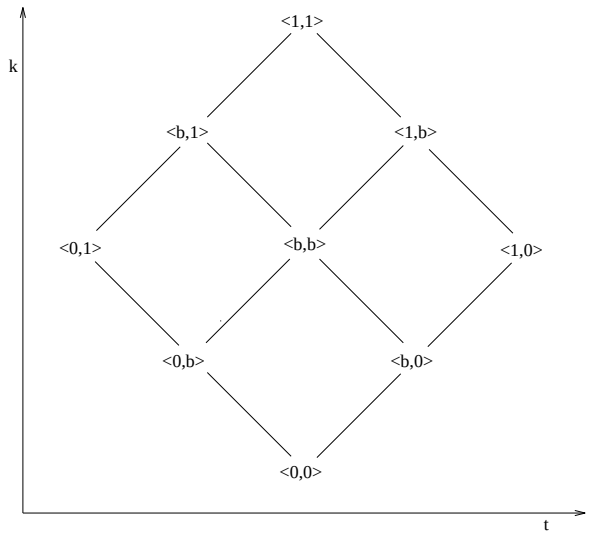

Figure 2: The bilattice $\mathcal{NINE}$

in the underlying lattice is join-irreducible. Furthermore, note that in $\mathcal{NINE}$ and in fact in any distributive bilattice $\mathcal{B}$, we have:

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

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

The bilattice $\mathcal{FOUR}$ contains four element, `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 and $2n-2$ k-join-irreducible elements. If the underlying lattice is a powerset lattice, then the difference between the set of grounded and the set of 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 $\mathcal{B}(L)$ will have $2^{n+1}-1$ grounded elements, but only $2n$ k-join-irreducible elements.

The following lemma provides a basis for the join-irreducible procedural semantics defined in the next section.

Lemma 2.8. *Let $\mathcal{B} = \mathcal{B}(L)$ be a distributive bilattice, where $\langle L, \leq \rangle$ 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 $c \leq_k b$;*
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 $c \leq_k b$.*

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. In addition, it includes the connectives $\leftarrow$, $\neg$, $\wedge$, $\vee$, $\otimes$, and $\oplus$. $\wedge$ 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 $\mathcal{L}$ [13]. An *atom* is either a constant b, where $b \in \mathcal{B} \setminus \{\top, \bot\}$, 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. 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 variables occurring in A, and $y_1, \cdots, y_m$ are 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. A *goal* is simply a formula. 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.

3.2 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, knowledge operators are self-dual under negation in the $\leq_k$ ordering, 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 [9].

An *interpretation* for a program P is a mapping $I : B_P \rightarrow \mathcal{B}$. I is extended in a natural way to ground formulas as follows: $I(\texttt{true}) = \texttt{true}$, $I(\texttt{false}) = \texttt{false}$, $I(\neg A) = \neg I(A)$ and $I(A_1 \square\ A_2) = I(A_1) \square\ I(A_2)$ for $\square\ \in \{\wedge, \vee, \otimes, \oplus\}$. 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 *semantic operator* Φ_P is the function from interpretations to interpretations defined as follows:

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

The knowledge ordering of $\mathcal{B}$ induces a pointwise partial ordering of interpretations. Φ_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, Φ_P has a least fixpoint, which provides the denotational meaning of the program P. In $\mathcal{FOUR}$, and in fact in any bilattice which satisfies the infinitary distributivity conditions, Φ_P is continuous. Hence the least fixpoint of Φ_P is obtained after ω iterations of the Φ_P starting with the smallest interpretation.

3.3 Generalized Procedural Semantics

Fitting's procedural model [8, 9] is based on a version of Smullyan style semantic tableaux. 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 constant b, where b is a truth value in the underlying bilattice, we say that A has a *b-proof*.

Substitutions, renamings, composition of substitutions, and basic unification concepts are defined in the standard manner as detailed in [13].

In our procedural semantics we employ a parallel computation model (see [19, 5]). 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 [14] (see [15]), to ensure the consistency of bindings obtained for shared variables during the computation. A theory of substitution unificaton based on the solution of equations is investigated in [6, 17]. The notion of unifiable substitutions has been used in concurrent logic programming systems which use parallelism [12].

Definition 3.1. Let S be a set of substitutions. Then a substitution γ is called a *substitution unifier* (*s-unifier*) of S, if $S\gamma = \{ \sigma\gamma : \sigma \in S \}$ is a singleton. If such a substitution γ exists, then we say that S is *unifiable*. γ is a *most general substitution unifier* of S, if for every s-unifier δ of S, there is a substitution η such that $\delta = \gamma\eta$. We denote the set of all most general s-unifiers of S by $mgsu(S)$.

Definition 3.2. Let S be a unifiable set of substitutions. A substitution δ 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 σ and τ, we often use a shorthand notation and denote the set of mgsu's of σ and τ by $mgsu(\sigma, \tau)$. Similarly, we denote the set of all substitution unifications of σ and τ by $\sigma \odot \tau$.

It is well-known that mgu's and mgsu's are unique up to renaming of variables [13, 6, 17, 14]. In the sequel we sometimes abuse the notation and interpret $mgu(E_1, E_2)$, $mgsu(\sigma_1, \sigma_2)$, and $\sigma_1 \odot \sigma_2$ as functions returning unique values.

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.

In the sequel we denote the ordering in the knowledge lattice by $\preceq$.

Definition 3.3. Let $\mathcal{B}$ be a distributive bilattice, P a program over $\mathcal{B}$, and G a goal. Let $b \in \mathcal{B}$. Then G has a *b-derivation* with answer θ if:

1. $G = c$ for some $c \in \mathcal{B}$ such that $b \preceq c$, and θ is the identity substitution ε; or
2. G is an atom that unifies with the head of a clause $A \leftarrow G' \in P$ with $\sigma = mgu(G, A)$, $G'\sigma$ has a c-derivation with answer θ' for some $c \succeq b$, and θ is the restriction of $\sigma\theta'$ to the variables of G; or
3. G is $\neg G'$, and G' has a c-derivation with answer θ for some $c \succeq \neg b$; or
4. G is $G_1 \sqsquare G_2$, where $\sqsquare \in \{ \oplus, \otimes, \vee, \wedge \}$, and G_1 has a c-derivation with answer θ_1 and G_2 has a d-derivation with answer θ_2 for some c, d such that $b \preceq c \sqsquare d$, and θ is the restriction of $\theta_1 \odot \theta_2$ to the variables of G.

Definition 3.4. 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-proof with answer* θ, if either

1. G has a b-derivation with answer θ; or
2. G has a b_i-derivation with answer θ_i, for $i = 1, 2, \cdots, n$, where $b = b_1 \oplus b_2 \oplus \cdots \oplus b_n$, and θ is the restriction of $\odot\{\theta_i \mid 1 \leq i \leq n\}$ to the variables of G.

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

Theorem 3.6 (Completeness). *Let $\mathcal{B}$ be a distributive bilattice, and let $b \in \mathcal{B}$. Let P be a program, G a goal, and θ a substitution for the variables of G. If $(\Phi_P \uparrow \omega)\ (G\theta) \succeq b$, then G has a b-proof with answer σ such that $G\theta = G\sigma\gamma$, for some substitution γ.*

The Soundness theorem is proved by induction on the length of b-derivations and the Completeness Theorem by induction on the number of iterations of Φ_P required to obtain the fixpoint. The proofs can be found in the full paper [16] in preparation. The key to the proof of the Completeness Theorem is the following lifting lemma. It generalizes the lifting lemma which is used in establishing the completeness of SLD-resolution (see [13]).

Lemma 3.7 (Lifting Lemma). *Let $\mathcal{B}$ be a distributive bilattice and $b \in \mathcal{B}$. Suppose $G\theta$ has a b-derivation with answer η, with respect to a program P. Then G has a b-derivation with answer σ such that $G\theta\eta = G\sigma\gamma$, for some substitution γ.*

4 Join-irreducible Procedural Semantics

Although the definition of a b-proof provides a sound and complete procedural semantics for logic programming over arbitrary distributive bilattices, it has a drawback. For a given truth value b, the search for a b-proof of a complex goal G may entail searches for c-proofs 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 essentially restrict our attention to b-proofs where b ranges over the relatively small subset of k-join-irreducible truth-values. Moreover, in the search for a b-proof for G, we need only look for b-proofs of the subformulas of G.

We now present a *join-irreducible* operational semantics as an alternative to the standard one presented above.

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

1. $G = c$ for some $c \in \mathcal{B}$ such that $b \preceq c$, and $\theta = \varepsilon$; or
2. G is an atom that unifies with the head of a clause $A \leftarrow G' \in P$ with $\sigma = mgu(G, A)$, $G'\sigma$ has a b-*JIR*-derivation with answer θ', and θ is the restriction of $\sigma\theta'$ to the variables of G; or
3. G is $\neg G'$, and G' has a $\neg b$-*JIR*-derivation with answer θ; 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 θ; 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 θ_i $(i = 1, 2)$, such that $\theta = \theta_1 \odot \theta_2$; 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 θ_i $(i = 1, 2)$, such that $\theta = \theta_1 \odot \theta_2$; 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 θ.

Definition 4.2. Let $\mathcal{B}$ be a distributive bilattice that has the DCP in the knowledge ordering. 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 the irredundant decomposition of a as join of join-irreducible elements of $\mathcal{B}$ in the knowledge ordering. Let G be a goal and P a program over $\mathcal{B}$. Then G has an *a-JIR-proof* with answer θ if G has a *b_i-JIR*-derivation with answer θ_i $(1 \leq i \leq n)$, and θ is the restriction of $\odot\{\theta_1, \cdots, \theta_n\}$ to the variables of G.

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 an irredundant decomposition of b as a join of join-irreducible elements of the bilattice (using most standard representation techniques for bilattices, this task can be accomplished in polynomial time). 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 $i = 1, 2, \cdots, n$.

We can now state the following theorem showing the relationship between the two bilattice-based procedural semantics.

Theorem 4.3. *Let $\mathcal{B}$ be a distributive bilattice having the DCP in the knowledge ordering. Let P be a program over $\mathcal{B}$ and G a goal. Suppose that $b \in \mathcal{B}$.*

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

This theorem is proved by induction on the length of derivations and *JIR*-derivations. Lemma 2.8 plays the key role in the proof, which is given in [16]. Theorem 4.3 implies a completeness theorem for the join-irreducible operational semantics as a corollary of Theorem 3.6.

References

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

[2] 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.

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

[4] 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.

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

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

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

[8] 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.

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

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

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

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

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

[14] B. Mobasher, *Generalized Knowledge-based Semantics for Multivalued Logic Programs*, Ph.D. Dissertation, Iowa State University, Ames, Iowa (1993).

[15] 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.

[16] B. Mobasher, D. Pigozzi, and G. Slutzki, Knowledge-based logic programs: an algebraic and uniform approach to deductive reasoning, Manuscript in preparation (1994).

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

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

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

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

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