Bilattices In Logic Programming

Melvin Fitting
mlflc@cunyvm.cuny.edu
Dept. Mathematics and Computer Science
Lehman College (CUNY), Bronx, NY 10468
Depts. Computer Science, Philosophy, Mathematics
Graduate Center (CUNY), 33 West 42nd Street, NYC, NY 10036 *

Abstract

Bilattices, introduced by M. Ginsberg, constitute an elegant family of multiple-valued logics. Those meeting certain natural conditions have provided the basis for the semantics of a family of logic programming languages. Now we consider further restrictions on bilattices, to narrow things down to logic programming languages that can, at least in principle, be implemented. Appropriate bilattice background information is presented, so the paper is relatively self-contained.

1 Introduction

Logic programming is more than just Prolog. It is a distinctive way of thinking about computers and programming that has led to the creation of a whole family of programming languages, mostly experimental. Some time ago I found that *bilattices* provided a uniform semantics for a rich and interesting group of logic programming languages [9]. Bilattices are a natural generalization of classical two-valued logic, and were introduced by Matt Ginsberg in [12], and more fully in [13]. Recently I have found that a special class of bilattices not only has good semantic properties, but also possesses a natural proof theory that lends itself well to implementation. One member of this family was considered in detail in [7]. At the time I prepared that paper I did not realize how general the ideas were. Here I will sketch an approach that applies to the family as a whole. Since this is a conference on multiple valued logics I will concentrate on the features of bilattices that are significant for my work. Details of logic programming will be merely summarized (or omitted altogether). I intend to write a fuller treatment of the logic programming aspects for presentation elsewhere.

2 Logic Programming Background

The logic of Prolog is classical, at least if one ignores negation as failure (see [17]). Operationally, queries either succeed or they don't. Denotationally, programs are given meanings in classical

*Research partly supported by NSF Grants CCR-8702307 and CCR 8901489, and PSC-CUNY Grant 668283.

models. There is a fixpoint semantics that picks out the 'intended model.' With a program one associates a 'single-step' operator, usually denoted T, mapping interpretations to interpretations. Such an operator always has a least fixed point, and this is taken to be the intended meaning of the program. There are results connecting the denotational and the (non-deterministic) operational semantics ([22], [1]).

More recently, Van Emden has proposed a natural generalization [21], in which 'confidence factors' are assigned to statements, rather than simple truth values. Better said, one can think of the unit interval as being a space of generalized truth values, and so the approach is rather like a fuzzy version of Prolog. Syntactically, programs are much like in (pure) Prolog, except that with each program clause is associated an 'attenuation factor.' The intention is, an application of a given program clause diminishes certainty by the attenuation factor associated with that clause. The result is a flexible language which naturaly generalizes Prolog, and which is suitable for writing some expert systems. Van Emden also showed that the standard fixpoint Prolog semantics carried over to his generalization in a direct, elegant way. I refer you to his paper for details.

Negation remains something of a problem, though. After all, negation as failure ([4]) is a problem for Prolog too. One well-known approach to the Prolog problem involves the notion of *stratification* [2]. Another approach that is especially pertinent to this conference involves a semantics based on a three-valued logic, [5], [15], [16]. There are even some connections between the two approaches, [10], [11]. Still, it is not entirely clear what negation as failure should even mean, once confidence factors have replaced simple truth values. It is this point I want to address in the rest of the paper. I want to propose a more 'positive' treatment of negation.

3 Bilattice Background

Bilattices are a family of multiple-valued logics with a particularly nice, useful algebraic structure. They were introduced by Matt Ginsberg; I refer you to his papers for his motivation [12], [13]. I have found bilattices to be natural tools both for the treatment of truth in languages allowing self-reference, [8], and for the semantics of logic programming generalizations, [6], [9]. It is their application to logic programming that will concern us here.

Essentially, a bilattice is a space of generalized truth values with *two* lattices orderings. One ordering, $\leq_t$, records degree of truth, whatever that means, while the other ordering, $\leq_k$, records degree of information or knowledge. As an informal example, no information, $\bot$, is certainly less information than falsehood, so one would want $\bot \leq_k \mathit{false}$. On the other hand, no information is of a greater degree of truth than falsehood, precisely because it doesn't contain the information of falsehood; hence $\mathit{false} \leq_t \bot$. This example will be made more formal below.

Of course if a bilattice were just a space with two lattice orderings, it would be nothing more than two lattices that happened to share a carrier. It is the postulated connections between the orderings that make it an interesting structure. And here, there are several degrees of connections that can be considered. Now it is time to introduce some formal definitions and terminology.

Definition 1 A *bilattice* is a structure $\langle \mathcal{B}, \leq_t, \leq_k, \neg \rangle$ consisting of a non-empty set $\mathcal{B}$, two partial orderings of it, $\leq_t$ and $\leq_k$, and a mapping from it to itself, $\neg$, such that:

1. each of $\leq_t$ and $\leq_k$ makes $\mathcal{B}$ a complete lattice;
2. $x \leq_t y \Longrightarrow \neg y \leq_t \neg x$;
3. $x \leq_k y \Longrightarrow \neg x \leq_k \neg y$;
4. $\neg\neg x = x$.

The definition above is due to Matt Ginsberg, [13], and is the basic definition. It is the negation operator that provides the connection between the two orderings. The connection is a very weak one, to be sure. Even so, there are interesting bilattice-like structures that do not have a notion of negation. Since the subject is still at an early stage, a proper choice of terminology is not always clear.

The intuition is elegantly straightforward. One expects negation to invert our notion of truth. On the other hand, we know no more and no less about $\neg P$ than we know about P. From an algebraic point of view, completeness of the lattices is a rather strong condition to build in at the ground level, and it is of interest to investigate bilattices that do not make such a requirement. We will say more about this below, although the bilattices we are primarily interested in here do satisfy the completeness requirement. Since the basic bilattice definition does contain a completeness requirement, there must be a top and a bottom element for each of the two orderings. From now on we assume neither ordering is trivial: bottom and top are different. We systematicaly use the following notation.

Definition 2 For the $\leq_t$ ordering: bottom and top are denoted *false* and *true*; meet and join are denoted $\wedge$ and $\vee$; infinitary meet and join are denoted $\bigwedge$ and $\bigvee$. For the $\leq_k$ ordering: bottom and top are denoted $\bot$ and $\top$; meet and join are denoted $\otimes$ and $\oplus$; infinitary meet and join are denoted $\prod$ and $\sum$.

It is straightforward to show that in a bilattice, *false* and *true* are switched by $\neg$, and the DeMorgan Laws hold with respect to $\vee$ and $\wedge$, while $\bot$ and $\top$ are left unchanged by $\neg$, and $\oplus$ and $\otimes$ are their own duals.

The essentials of the next definition are also due to Ginsberg. In [9] a completeness condition was included, which we omit here.

Definition 3 An *interlaced bilattice* is a structure $\langle \mathcal{B}, \leq_t, \leq_k \rangle$ where:

1. both $\leq_t$ and $\leq_k$ give $\mathcal{B}$ the structure of a lattice;
2. $x \leq_t y \Longrightarrow x \oplus z \leq_t y \oplus z$ and $x \otimes z \leq_t y \otimes z$;
3. $x \leq_k y \Longrightarrow x \vee z \leq_k y \vee z$ and $x \wedge z \leq_k y \wedge z$.

Thus in an interlaced bilattice an operation associated with one of the lattice orderings is required to be monotonic with respect to the other lattice ordering. This is a different kind of connection between the two orderings from that considered above, via negation.

Bilattices were required to be complete, and hence had tops and bottoms. Interlaced bilattices, as defined here, do not have a completeness requirement. Nonetheless, from now on we assume all

such structures have tops and bottoms with respect to both orderings. Consideration of weaker structures, while interesting, would take us too far afield here.

While the definitions may seem somewhat peculiar on first glance, there is a natural and large family of examples. We need one more definition, then the motivation will become easier to present.

Definition 4 Let $\langle L_1, \leq_1 \rangle$ and $\langle L_2, \leq_2 \rangle$ be lattices. By $L_1 \odot L_2$ we mean the structure $\langle L_1 \times L_2, \leq_t, \leq_k \rangle$ where:

1. $\langle x_1, x_2 \rangle \leq_t \langle y_1, y_2 \rangle$ provided $x_1 \leq_1 y_1$ and $y_2 \leq_2 x_2$;
2. $\langle x_1, x_2 \rangle \leq_k \langle y_1, y_2 \rangle$ provided $x_1 \leq_1 y_1$ and $x_2 \leq_2 y_2$.

Proposition 5 *If L_1 and L_2 are lattices (with tops and bottoms), $L_1 \odot L_2$ is an interlaced bilattice. Further, if $L_1 = L_2$ then the operation given by $\neg\langle x, y \rangle = \langle y, x \rangle$ satisfies the negation conditions of the bilattice definition. Finally, if both L_1 and L_2 are complete lattices, $L_1 \odot L_2$ satisfies the completeness condition for bilattices.*

Now for motivation. Think of the lattice L_1 as what we use when we measure the degree of belief, confidence, evidence, etc. that we have *in* a statement. Think of L_2 as what we use when we measure the degree of doubt, lack of confidence, counter-evidence, etc. that we have *against* a statement. Then a pair, $\langle x, y \rangle$, summarizes our evidence for and our evidence against. Notice that bilattices treat doubt not just as lack of belief; instead it is given a positive role to play. Notice also that no requirement is imposed that these pairs, or 'truth values,' be consistent. Anything is allowed; after the fact we can try to identify those truth values that are consistent. Consistent truth values are those in which belief and doubt complement each other in some reasonable sense. This is treated more fully in [9], and is ignored here.

Next we turn to specific examples. First, the simplest one. Take for both L_1 and L_2 the two-element lattice $\{0, 1\}$ where $0 \leq 1$. Think of 0 as no evidence, and 1 as complete certainty. Then $L_1 \odot L_2$ is a four-element bilattice in which $\bot$ is $\langle 0, 0 \rangle$, indicating we have no evidence either for or against, and $\top$ is $\langle 1, 1 \rangle$, indicating we are in the inconsistent situation of having full evidence both for and against. Likewise *false* is $\langle 0, 1 \rangle$ and *true* is $\langle 1, 0 \rangle$. This bilattice is presented graphically in Figure 1. It is, in fact, a well-known four-valued logic due to Belnap [3]. The operations associated with the $\leq_t$ ordering, together with negation, when confined to $\{false, true\}$ are simply those of classical logic, while when $\bot$ is added they are those of Kleene's strong three valued logic [14]. Moreover $\otimes$ can naturally be thought of as a *consensus* operator. $x \otimes y$ is the most information that x and y agree on. Likewise $\oplus$ is an *accept anything* or *gullability* operator.

Another natural example, and one that will be of special concern here, is based on the unit interval. Take both L_1 and L_2 to be $[0, 1]$. Now we can think of a member of the resulting bilattice as expressing a *degree* of belief and of doubt. The negation operation simply switches the roles of belief and doubt. Likewise $\wedge$ minimizes belief while maximizing doubt, and $\otimes$ minimizes both belief and doubt.

There is yet a stronger natural condition that can be imposed on bilattices. Since we have four basic operations, $\wedge$, $\vee$, $\otimes$ and $\oplus$, there are 12 possible distributive laws.

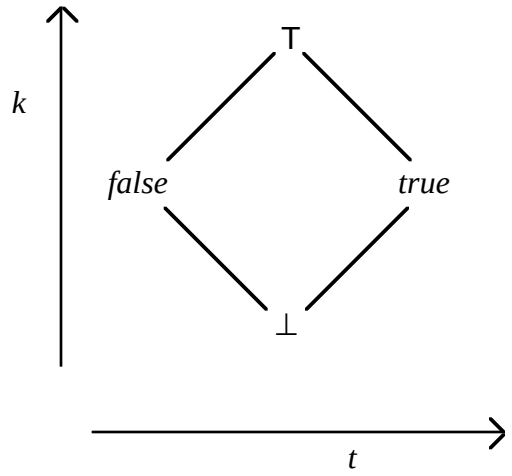

Figure 1: A four valued bilattice

Definition 6 A bilattice, or an interlaced bilattice, is *distributive* if all twelve distributive laws hold.

As a matter of fact, it is easy to see that the distributive laws imply the interlacing laws. (We use this without comment below.) Further, if L_1 and L_2 are distributive lattices, $L_1 \odot L_2$ will be distributive. Hence both examples above are distributive. Somewhat more remarkably, these are the only kinds of distributive bilattices, because of a representation theorem due to Ginsberg [13]. Before stating and proving it, we give a useful 'normal form' result.

Lemma 7 *In a distributive bilattice* $\mathcal{B}$, $x = (x \wedge \bot) \oplus (x \vee \bot)$.

Proof Using distributivity, and the absorption laws of lattices:

$$\begin{aligned}(x \wedge \bot) \oplus (x \vee \bot) &= [x \oplus (x \vee \bot)] \wedge [\bot \oplus (x \vee \bot)] \\ &= [(x \oplus x) \vee (x \oplus \bot)] \wedge [(\bot \oplus x) \vee (\bot \oplus \bot)] \\ &= [x \vee x] \wedge [x \vee \bot] \\ &= x \wedge (x \vee \bot) \\ &= x\end{aligned}$$

Now the representation theorem itself.

Proposition 8 *Suppose* $\mathcal{B}$ *is a distributive bilattice. Then there are distributive lattices* L_1 *and* L_2 *such that* $\mathcal{B}$ *is isomorphic to* $L_1 \odot L_2$.

Proof Let $L_1 = \{x \vee \bot \mid x \in \mathcal{B}\}$, and give it the ordering $\leq_1$ where $x \leq_1 y \Leftrightarrow x \leq_t y$. Let $L_2 = \{x \wedge \bot \mid x \in \mathcal{B}\}$ with ordering $x \leq_2 y \Leftrightarrow y \leq_t x$. Using the distributive laws it is easy to check that both L_1 and L_2 are closed under $\wedge$, $\vee$, $\otimes$ and $\oplus$. Now we verify several further items.

1) For $x, y \in L_1$, $x \leq_t y \Leftrightarrow x \leq_k y$. For $x, y \in L_2$, $x \leq_t y \Leftrightarrow y \leq_k x$.

We check the second of these assertions. Say $x = a \wedge \bot$ and $y = b \wedge \bot$. Each direction requires a separate argument.

Suppose $a \wedge \bot \leq_t b \wedge \bot$. Then $a \wedge \bot = a \wedge b \wedge \bot$. Now $\bot \leq_k a \wedge \bot$ so $b \wedge \bot \leq_k b \wedge a \wedge \bot = a \wedge \bot$.

Suppose $b \wedge \bot \leq_k a \wedge \bot$. Then $a \wedge b \wedge \bot \leq_k a \wedge \bot$. Also $\bot \leq_k b \wedge \bot$ so using the interlacing conditions, $a \wedge \bot \leq_k a \wedge b \wedge \bot$. Then $a \wedge \bot = a \wedge b \wedge \bot$ so $a \wedge \bot \leq_t b \wedge \bot$.

2) Define a map $\theta : \mathcal{B} \rightarrow L_1 \odot L_2$ by: $x\theta = \langle x \vee \bot, x \wedge \bot \rangle$. θ is one-one and onto.

That θ is one-one is an easy consequence of Lemma 7. To show it is onto, suppose $\langle a \vee \bot, b \wedge \bot \rangle \in L_1 \odot L_2$. Set $t = (b \wedge \bot) \oplus (a \vee \bot)$; we claim $t\theta = \langle a \vee \bot, b \wedge \bot \rangle$. By definition, $t\theta = \langle t \vee \bot, t \wedge \bot \rangle$. Now,

$$\begin{array}{rcl} t \vee \bot & = & [(b \wedge \bot) \oplus (a \vee \bot)] \vee \bot \\ & = & [(b \wedge \bot) \vee \bot] \oplus [a \vee \bot \vee \bot] \\ & = & \bot \oplus (a \vee \bot) \\ & = & a \vee \bot \end{array}$$

Equality of the second components is shown similarly.

3) Finally, $a \leq_t b \Leftrightarrow a\theta \leq_t b\theta$ and $a \leq_k b \Leftrightarrow a\theta \leq_k b\theta$.

We show the second of these equivalences. Suppose $a \leq_k b$. Then $a \wedge \bot \leq_k b \wedge \bot$ so by item 1), $b \wedge \bot \leq_t a \wedge \bot$, hence $a \wedge \bot \leq_2 b \wedge \bot$. Also $a \vee \bot \leq_k b \vee \bot$ so $a \vee \bot \leq_t b \vee \bot$, hence $a \vee \bot \leq_1 b \vee \bot$. Then $a\theta \leq_k b\theta$.

Conversely, suppose $a\theta \leq_k b\theta$. Then $a \vee \bot \leq_k b \vee \bot$ and $a \wedge \bot \leq_k b \wedge \bot$ so, using Lemma 7, $a = (a \wedge \bot) \oplus (a \vee \bot) \leq_k (b \wedge \bot) \oplus (b \vee \bot) = b$.

Incidentally, the Proposition can be generalized somewhat. Distributivity of the bilattice can be weakened to the assumption that all distributive laws hold between x, y and z provided one of the three is $\bot$. Of course the conclusion must also be weakened to drop distributivity for the resulting lattices. Similarly, with suitable modifications, $\top$, *false* or *true* can be used in place of $\bot$ in the proof.

Finally, if a completeness condition is imposed, then *infinitary* distributive laws can be considered, such as $a \otimes \bigvee_i b_i = \bigvee_i (a \otimes b_i)$ and $a \wedge \prod_i b_i = \prod_i (a \wedge b_i)$. If all such equations hold we say a bilattice meets the *infinite distributivity conditions*. As might be expected, if L_1 and L_2 are complete infinitely distributive lattices, then $L_1 \odot L_2$ will be a complete bilattice meeting the infinite distributivity conditions. We will need such bilattices later on.

4 Logic Programming Syntax

We want to set up a family of logic programming languages with a bilattice as the underlying space of truth values. We do this in a quite general way; then additional restrictions or extensions can be imposed depending on the choice of particular bilattice. The machinery presented here is essentially a condensation and specialization of that found in [9].

Let $\mathcal{B}$ be a bilattice that is also interlaced. We set up a logic programming language $L(\mathcal{B})$ over $\mathcal{B}$. To this end we assume we have the usual notions of term and atomic formula from first-order

logic. Conventional Prolog has the equivalent of a truth constant; when one writes $A \leftarrow$ it is as if we had $A \leftarrow \mathit{true}$. We will need something more general; we assume that we have *constants* in our language $L(\mathcal{B})$, particular atomic formulas, that designate members of the bilattice $\mathcal{B}$. To keep things simple we assume members of $\mathcal{B}$ themselves occur in the language $L(\mathcal{B})$, to avoid details on how such members can be named.

The domain of the intended model for $L(\mathcal{B})$ will be the Herbrand universe (of closed terms), familiar from Prolog semantics (see [17]). We greatly broaden the notion of formula, however. To this end we assume we have formal symbols, $\wedge$, $\vee$, $\otimes$, $\oplus$ and $\neg$, corresponding to the various operations on the bilattice. We also assume we have quantifiers $\forall$ and $\exists$ to correspond to the infinitary truth operations $\bigwedge$ and $\bigvee$ of bilattices, and also $\prod$ and $\sum$ corresponding to the infinitary information operations of the $\leq_k$ ordering.

Definition 9 *Formulas* of $L(\mathcal{B})$ are expressions built up from atomic formulas using $\neg$, the binary connectives $\wedge$, $\vee$, $\otimes$ and $\oplus$, and the quantifiers $\forall$, $\exists$, $\prod$ and $\sum$.

Definition 10 A *program clause* of $L(\mathcal{B})$ is an expression of the form $A \leftarrow \Phi$ where A is atomic (not a constant) and Φ is an arbitrary formula, all of whose free variables occur in A. A is the *head* and Φ is the *body* of this program clause.

Definition 11 A *program* of $L(\mathcal{B})$ is a finite set of program clauses in which no two program clauses have heads involving the same relation symbol.

There is a standard way in Prolog to combine two program clauses for the same relation symbol, using equality. Similar ideas carry over to languages based on a wide variety of bilattices, so we have chosen to simply build uniqueness into the definition of program. Also we have required that all free variables in clause bodies occur in clause heads. This is no real restriction since we allow explicit quantification in bodies.

Note that the syntax so far does not commit us to a particular bilattice (except for the choice of constants allowed as atomic formulas). Further, if we restrict ourselves to programs not involving the $\leq_k$ connectives and quantifiers, not involving negation, and not involving $\forall$, we have essentially the syntax of Prolog, give or take some obvious transformations.

5 Fixpoint Semantics

For pure Prolog, [22] and [1] showed how a single-step operator could be associated with a program. If negations are not present, such an operator has a least fixed point in a natural sense, and that least fixed point supplies a denotational meaning for the program. We showed in [9] that these ideas carry over in a straightforward way to logic programming languages with an interlaced bilattice as the space of truth values. The most natural 'direction' in which to evaluate a *least* fixed point is the information or knowledge direction; it turns out that negation poses none of the familiar problems then. However if negation is not present, least, and greatest, fixed points will exist under both bilattice orderings, and there are deep connections between them in many cases. This is more than

we can go into here. We merely sketch the basic ideas of the fixpoint semantics, using bilattices. The ideas involve less of logic programming than of logic, and should be of some interest here.

For the rest of this section, $L(\mathcal{B})$ is a logic programming language allowing all the syntactic machinery sketched earlier. Thus program clause bodies can contain any of $\wedge$, $\vee$, $\neg$, $\otimes$, $\oplus$, $\forall$, $\exists$, $\prod$ and $\sum$. Also we assume $\langle \mathcal{B}, \leq_t, \leq_k \rangle$ is a a bilattice that is also interlaced. That is, it meets the conditions for being a bilattice, including having a negation operation and being complete with respect to both orderings, and it meets the interlacing conditions. $\mathcal{B}$ is the intended space of truth values for our logic programming language.

Let P be a program in the language $L(\mathcal{B})$. Let H be the Herbrand universe, consisting of ground terms of the language $L(\mathcal{B})$. We assume H is non-trivial. H will serve as the intended domain of quantification. By P^H we mean the (generally infinite) family of ground instances of members of P. Note that because of the way we defined programs, if a ground atomic formula A occurs as the head of some member of P^H, it occurs only once.

Definition 12 An *interpretation* is a mapping v from ground atomic formulas to the bilattice $\mathcal{B}$, that maps members of $\mathcal{B}$ to themselves.

Pointwise partial orderings are defined on the family of interpretations. That is, $v_1 \leq_t v_2$ provided $v_1(A) \leq_t v_2(A)$ for every ground atomic formula A, and similarly for $\leq_k$. It is straightforward to check that the space of interpretations itself becomes an interlaced bilattice. In it the smallest interpretation, in the $\leq_k$ ordering, is the one that maps members of $\mathcal{B}$ to themselves, and is otherwise $\bot$.

Interpretations are defined only on ground atoms. But they have a direct extension to arbitrary closed formulas. Set $v(\neg X) = \neg v(X)$, $v(X \otimes Y) = v(X) \otimes v(Y)$, and so on for the other binary connectives. Further, set $v((\forall x)Z(x)) = \bigwedge_{h \in H} v(Z(h))$, $v((\prod x)Z(x)) = \prod_{h \in H} v(Z(h))$, etc. With these definitions, it turns out we have the following nice monotonicity result:

$$v_1 \leq_k v_2 \Longrightarrow v_1(X) \leq_k v_2(X) \text{ for any ground sentence } X.$$

A similar result does not hold for the $\leq_t$ ordering unless negation is disallowed.

Now we associate with the program P an operator, which we denote Φ_P, mapping interpretations to interpretations. For an interpretation v, $\Phi_P(v)$ is the interpretation whose value on the ground atomic formula A is given by:

$$\Phi_P(v)(A) = \begin{cases} A & \text{if } A \in \mathcal{B} \\ v(B) & \text{if } A \leftarrow B \in P^H \\ \bot & \text{otherwise.} \end{cases}$$

It follows from the monotonicity of interpretations on ground formulas that Φ_P itself is monotonic in the information direction. That is:

$$v_1 \leq_k v_2 \Longrightarrow \Phi_P(v_1) \leq_k \Phi_P(v_2)$$

Now, by the Knaster-Tarski Theorem [20], Φ_P has a least fixed point. It is this that we take as supplying the denotational meaning for the program P.

There is a standard way of approximating to least fixed points. Define a transfinite sequence v_P^α as follows:

$$\begin{array}{lcl} v_P^0 & = & \text{the smallest interpretation under } \leq_k \\ v_P^{\alpha+1} & = & \Phi_P(v_P^\alpha) \\ v_P^\lambda & = & \sum_{\alpha<\lambda} v_P^\alpha \text{ for limit } \lambda \end{array}$$

This sequence increases in the $\leq_k$ ordering with increasing ordinals, eventually becoming constant, settling on the least fixed point of Φ_P. The smallest ordinal α such that v_P^α is the least fixed point of Φ_P is called the *closure ordinal* of Φ_P. One would like the closure ordinal to be ω (or less) because otherwise an implementation is doubtful. Unfortunately this is not the case generally. It does happen when certain restrictions are imposed though. We take these up next, after some preliminaries.

Lemma 13 *In any bilattice,* $\neg \sum_i a_i = \sum_i \neg a_i$ *and* $\sum_i \sum_j a_{i,j} = \sum_j \sum_i a_{i,j}$.

Proof The second equality is standard for complete lattices. We prove the first. By definition, $\neg a_j \leq_k \sum_i \neg a_i$. Then, since negation preserves the information ordering, $a_j \leq_k \neg \sum_i \neg a_i$. Since j is arbitrary, $\sum_i a_i \leq_k \neg \sum_i \neg a_i$, and so $\neg \sum_i a_i \leq_k \sum_i \neg a_i$. Since this is the case for an arbitrary family a_i we have, on replacing a_i by $\neg a_i$, $\neg \sum_i \neg a_i \leq_k \sum_i a_i$ and hence $\sum_i \neg a_i \leq_k \neg \sum_i a_i$.

Lemma 14 *Suppose* $\mathcal{B}$ *is a bilattice that meets the infinite distributivity conditions. Let* $a_1 \leq_k a_2 \leq_k a_3 \ldots$ *and* $b_1 \leq_k b_2 \leq_k b_3 \leq_k \ldots$ *be two sequences in* $\mathcal{B}$*. Finally let* $\circ$ *be one of* $\oplus$, $\otimes$, $\vee$ *or* $\wedge$*. Then*

$$\sum_i (a_i \circ b_i) = \sum_i a_i \circ \sum_i b_i.$$

Proof Infinite distributivity implies distributivity. This in turn implies interlacing, all of which we use below.

Let m be the maximum of i and j. Then, since the sequences a_i and b_i are increasing,

$$a_i \circ b_j \leq_k a_m \circ b_m \leq_k \sum_i (a_i \circ b_i).$$

Since j is arbitrary,

$$\sum_j (a_i \circ b_j) \leq_k \sum_i (a_i \circ b_i)$$

so using an infinitary distributive law,

$$a_i \circ \sum_j b_j \leq_k \sum_i (a_i \circ b_i).$$

Finally, since i is arbitrary,

$$\sum_i (a_i \circ \sum_j b_j) \leq_k \sum_i (a_i \circ b_i),$$

so by distributivity again,

$$\sum_i a_i \circ \sum_j b_j \leq_k \sum_i (a_i \circ b_i).$$

Conversely $a_i \leq_k \sum_i a_i$ and $b_i \leq_k \sum_i b_i$, so by the interlacing conditions,

$$a_i \circ b_i \leq_k \sum_i a_i \circ \sum_i b_i$$

and so

$$\sum_i (a_i \circ b_i) \leq_k \sum_i a_i \circ \sum_i b_i.$$

With these preliminaries out of the way we return to logic programming issues.

Definition 15 We call a logic program *k-existential* provided $\sum$ is the only infinitary operation symbol allowed to appear in program clause bodies. That is, none of $\prod$, $\forall$ or $\exists$ are allowed.

Theorem 16 *Suppose $\mathcal{B}$ is a bilattice that meets the infinite distributivity conditions, and P is a k-existential program. Then the operator Φ_P is continuous, and its closure ordinal is ω.*

Proof To say an operator Φ_P is continuous means, if $v_1 \leq_k v_2 \leq_k v_3 \leq_k \ldots$ then $\sum_i \Phi_P(v_i) = \Phi_P(\sum_i v_i)$. It is a standard result that a continuous operator has a closure ordinal of ω, so we concentrate on showing Φ_P is continuous. So, suppose we have an increasing sequence v_i of interpretations. It is enough to show that for each ground atomic formula A,

$$\left(\sum_i \Phi_P(v_i)\right)(A) = \left(\Phi_P(\sum_i v_i)\right)(A).$$

Since the operations on the space of interpretations have pointwise definitions, this comes down to showing

$$\sum_i \left(\Phi_P(v_i)\right)(A) = \left(\Phi_P(\sum_i v_i)\right)(A).$$

Say the program clause $A \leftarrow B \in P^H$. Then showing the formula above is equivalent to showing

$$\sum_i v_i(B) = (\sum_i v_i)(B).$$

This can be showed by induction on the complexity of B. If B is atomic, the result is immediate because of the pointwise nature of operations in the space of interpretations. For more complicated cases the lemmas above supply what is needed.

The two element lattice $\{0, 1\}$ trivially satisfies the infinitary distributive laws, and so the result above applies to the bilattice displayed in Figure 1. A logic programming language for this bilattice was investigated in some detail in [7], and the result contained in the Theorem above was stated, but not proved. The complete lattice $[0, 1]$, with the standard ordering, also satisfies the infinitary distributive laws, and hence the Theorem applies to the bilattice $[0, 1] \odot [0, 1]$. This is an example that will concern us below. Clearly many other examples exist. Incidentally, the Theorem above fits nicely into the framework of [18]. The preliminary Lemmas can be looked at as establishing that the framework of that paper is appropriate.

6 Smullyan style propositional rules

In [19] Smullyan presented a proof procedure for classical logic using *signed formulas* in *semantic tableaux*. A signed formula is $T\,X$ or $F\,X$, where X is a formula. The intuition is straightforward: $T\,X$ means that X is true (in some model); $F\,X$ means X is false (again, in some model). If one can produce a contradiction starting from $F\,X$, informally X can not be false, and so must be valid. This intuition is the direct motivation behind the Smullyan tableau rules.

In addition, Smullyan introduced *uniform notation*. It is easy to check that $T\,X \wedge Y \Leftrightarrow T\,X$ and $T\,Y$, and also $F\,X \vee Y \Leftrightarrow F\,X$ and $F\,Y$. Since both $T\,X \wedge Y$ and $F\,X \vee Y$ act conjunctively, they are grouped into the same category. Then one can give rules for, and investigate, whole groups of signed formulas together. Propositionally Smullyan had two categories: α or conjunctive; and β or disjunctive. For each α signed formula two *components*, α_1 and α_2 were defined, so that α was equivalent to the conjunction of α_1 and α_2. For example, $F\,X \vee Y$ is an α, and for it α_1 is $F\,X$ and α_2 is $F\,Y$. Likewise β formulas have their components, β_1 and β_2, with each β being equivalent to the disjunction of them.

Now, a *proof* of X in Smullyan's system is a closed tableau for $F\,X$. A tableau for $F\,X$ is a tree with $F\,X$ at the root, constructed using the branch extension rules given in Table 1:

$$\begin{array}{cccc} \dfrac{\alpha}{\begin{array}{l}\alpha_1\\ \alpha_2\end{array}} & \dfrac{\beta}{\beta_1 \mid \beta_2} & \dfrac{T\,\neg Z}{F\,Z} & \dfrac{F\,\neg Z}{T\,Z} \end{array}$$

Table 1: Branch Extension Rules

The intention is, if an α occurs on a branch, α_1 and α_2 can be added to the branch end. If a β occurs on a branch, the branch is split and β_1 is added to one fork and β_2 to the other. Similarly for the negation rules. A tableau is *closed* if each branch contains a contradiction, $T\,Z$ and $F\,Z$ for some Z.

In a similar way we can define a *disproof* of X in Smullyan's system: it is a closed tableau for $T\,X$.

We want to extend Smullyan's ideas, as far as possible, to bilattices. Now, if a bilattice $\mathcal{B}$ is distributive it will be of the form $L_1 \odot L_2$ for lattices L_1 and L_2. It is convenient to make use of this representation. Recall, in Smullyan's system one works backward. To establish that X is valid one begins by assuming it is not, $F\,X$, and one tries for a contradiction. Likewise now, if X is given values in the bilattice $L_1 \odot L_2$, and we want to establish the 'evidence for' X is at least a, we begin by assuming it isn't, and try for a contradiction. Thus we introduce a *family* of signs F_a for each $a \in L_1$, and we think of $F_a\,X$ as asserting that the valuation of X in $L_1 \odot L_2$ has a first component that is not $\geq a$. Then a contradiction based on $F_a\,X$ will establish that the evidence for X is at least a. Similarly $T_b\,X$ will mean the valuation of X has a second component that is not $\geq b$, where $b \in L_2$. Now we proceed more formally.

- For the rest of this section, $\mathcal{B} = L_1 \odot L_2$ is a fixed bilattice.

Definition 17 A *$\mathcal{B}$-signed formula* is one of $F_a X$ or $T_b X$, where X is a formula, $a \in L_1$ and $b \in L_2$.

Definition 18 For a valuation v in $\mathcal{B}$, we extend its action to map signed formulas to *classical* truth values, true or false, as follows:

1. $v(F_a X)$ is true if $v(X) = \langle x_1, x_2 \rangle$ and $x_1 \not\geq a$;
2. $v(T_b X)$ is true if $v(X) = \langle x_1, x_2 \rangle$ and $x_2 \not\geq b$.

Now, it is straightforward to check that at least some of Smullyan's conditions extend directly. For example, for a given valuation v, $v(T_b X \vee Y)$ is true if and only if $v(T_b X)$ is true or $v(T_b Y)$ is true. To state things as simply as possible we extend Smullyan's uniform notation system. We also classify $\mathcal{B}$-signed formulas into *conjunctive* or α formulas, with *components* α_1 and α_2, and into *disjunctive* or β formulas, with *components* β_1 and β_2. These are given in Table 2.

α	α_1	α_2
$T_b X \wedge Y$	$T_b X$	$T_b Y$
$F_a X \vee Y$	$F_a X$	$F_a Y$
$T_b X \oplus Y$	$T_b X$	$T_b Y$
$F_a X \oplus Y$	$F_a X$	$F_a Y$

β	β_1	β_2
$T_b X \vee Y$	$T_b X$	$T_b Y$
$F_a X \wedge Y$	$F_a X$	$F_a Y$
$T_b X \otimes Y$	$T_b X$	$T_b Y$
$F_a X \otimes Y$	$F_a X$	$F_a Y$

Table 2: Signed Formulas

Now the following is straightforward to verify.

Proposition 19 *For any valuation v in $\mathcal{B}$:*

1. *$v(\beta)$ is true $\Longleftrightarrow$ $v(\beta_1)$ is true or $v(\beta_2)$ is true;*
2. *$v(\alpha)$ is true $\Longrightarrow$ $v(\alpha_1)$ is true and $v(\alpha_2)$ is true.*

Unfortunately, we do not generally have the converse of item 2) above. This means that a tableau-like proof procedure based on these ideas will generally be sound, but not complete. There is, however, a class of interesting bilattices for which we do have the converse.

Proposition 20 *The following are equivalent:*

1. *for any valuation v in $L_1 \odot L_2$, $v(\alpha_1)$ is true and $v(\alpha_2)$ is true $\Longrightarrow$ $v(\alpha)$ is true;*
2. *the lattice orderings of L_1 and L_2 are linear.*

Proof We show item 1) implies item 2); the converse direction is a special case of the Proposition below. Consider the formula $X \vee Y$, and suppose v is the valuation such that $v(X) = \langle x_1, x_2 \rangle$ and $v(Y) = \langle y_1, y_2 \rangle$. Then $v(X \vee Y) = \langle x_1 \vee y_1, x_2 \wedge y_2 \rangle$. Now consider the α signed formula $F_{x_1 \vee y_1} X \vee Y$. Item 1) asserts that $x_1 \not\geq x_1 \vee y_1$ and $y_1 \not\geq x_1 \vee y_1$ implies $x_1 \vee y_1 \not\geq x_1 \vee y_1$. It follows from this that $x_1 \geq x_1 \vee y_1$ or $y_1 \geq x_1 \vee y_1$. Since $x_1 \vee y_1 \geq x_1$ and $x_1 \vee y_1 \geq y_1$, we have $x_1 \geq y_1$ or $y_1 \geq x_1$. Thus L_1 is linear; the linearity of L_2 is shown similarly.

Proposition 21 *Let $\mathcal{B} = L_1 \odot L_2$ where both L_1 and L_2 have lattice orderings that are linear. Then for valuations v_1 and v_2:*

$$v_1(\alpha_1) \text{ and } v_2(\alpha_2) \text{ both true } \Longrightarrow (v_1 \otimes v_2)(\alpha) \text{ true.}$$

Proof Suppose $v_1(X) = \langle x_1, x_2\rangle$, $v_2(Y) = \langle y_1, y_2\rangle$, and $\alpha = F_a\, X \vee Y$. Suppose also that $v_1(\alpha_1)$ and $v_2(\alpha_2)$ are both true. Then $x_1 \not\geq a$ and $y_1 \not\geq a$ so, since the ordering in L_1 is linear, $x_1 < a$ and $y_1 < a$. Now

$$\begin{array}{rcl} (v_1 \otimes v_2)(X \vee Y) & = & (v_1 \otimes v_2)(X) \vee (v_1 \otimes v_2)(Y) \\ & \leq_k & v_1(X) \vee v_2(Y) \\ & = & \langle x_1, x_2\rangle \vee \langle y_1, y_2\rangle \\ & = & \langle x_1 \vee y_1, x_2 \wedge y_2\rangle. \end{array}$$

Since $x_1 < a$ and $y_1 < a$, $x_1 \vee y_1 < a$, so $x_1 \vee y_1 \not\geq a$, and hence $(v_1 \otimes v_2)(F_a\, X \vee Y)$ is true.

Finally, if $L_1 = L_2$ then a natural notion of negation is available: $\neg\langle x, y\rangle = \langle y, x\rangle$. Under these circumstances the following is easy to verify.

Proposition 22 *For any valuation v in $L \odot L$:*

1. *$v(T_a \neg X)$ is true $\Longleftrightarrow$ $v(F_a\, X)$ is true.*
2. *$v(F_a \neg X)$ is true $\Longleftrightarrow$ $v(T_a\, X)$ is true.*

So far we have established that if we assume the linearity of L_1 and L_2 then Smullyan-style α and β rules will be appropriate, and if $L_1 = L_2$ so that we have negation, the Smullyan rules for negation extend directly. Thus we have the rules in Table 3, for $\mathcal{B}$-signed formulas, where $\mathcal{B} = L \odot L$ with L linear.

$$\begin{array}{cccc} \dfrac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} & \dfrac{\beta}{\beta_1 \mid \beta_2} & \dfrac{T_a \neg Z}{F_a\, Z} & \dfrac{F_a \neg Z}{T_a\, Z} \end{array}$$

Table 3: Bilattice Branch Extension Rules

Smullyan's rules for branch closure are a problem however. In his system $T\, Z$ and $F\, Z$ constitute a contradiction. But now, $T_a\, Z$ and $F_a\, Z$ are not jointly impossible. Together they merely say we have contradictory evidence, and so an inconsistent truth value for Z. Inconsistent truth values are allowed, and can be used to record important facts about our attempts at reasoning. In place of Smullyan's contradictions we make use of the presence of members of the bilattice $\mathcal{B}$ in the language $L(\mathcal{B})$.

Definition 23 $F_a\, \langle x, y\rangle$ is *contradictory* if $x \geq a$. $T_a\, \langle x, y\rangle$ is *contradictory* if $y \geq a$.

Now a closed tableau is one in which each branch contains a contradictory signed formula. A closed tableau for $F_a\,\langle x, y\rangle$ establishes that $x \geq a$ and a closed tableau for $T_b\,\langle x, y\rangle$ establishes that $y \geq b$. Then to determine lower bounds on the information content of X we must run *two* tableaux, one for $T_b\,X$ and one for $F_a\,X$. Based on the propositions above, the resulting tableau system is sound and complete.

As it happens, Smullyan's rules can be considerably simplified here. In his classical system, contradictions involved more than one signed formula, but that is not the case now. Consequently branches need never contain more than a single formula, and the α rule can be replaced by the version in Table 4

$$\frac{\alpha}{\alpha_1} \qquad \frac{\alpha}{\alpha_2}$$

Table 4: Simplified α Rules

The tableau system with the revised α rules is now essentially non-deterministic. It is possible to miss a proof by applying the wrong rule, and so an implementation must be prepared to backtrack.

7 Putting Things Together

We now have all the pieces needed to implement, at least in principle, an interesting family of logic programming languages. Since logic programming is not the central concern here, this section is rather sketchy. A more detailed presentation is in preparation.

For the rest of this section we take $\mathcal{B}$ to be the bilattice $L \odot L$ where L is a complete lattice whose ordering is linear. Then $\mathcal{B}$ has every nice property we might want. It is a bilattice that is interlaced, distributive, and even satisfies the infinitary distributive laws. Further, the logic programming language $L(\mathcal{B})$ has a good fixpoint semantics [9], and k-existential programs give rise to continuous operators. (Then if P is k-existential, Φ_P attains its least fixed point in ω steps. We use $\mu\Phi_P$ to denote this least fixed point.) Finally, a tableau style proof procedure is available.

Since we have infinitary distributive laws, any k-existential formula is equivalent to one in which all occurrences of $\sum$ are in front. Then, in the usual Prolog fashion, $\sum$ quantifiers can be left implicit, i.e. omitted. From now on we assume program clauses are of the form $A \leftarrow \phi$ where ϕ has no quantifiers at all.

Next, an analog of SLD resolution is straightforward, incorporating the Smullyan style rules.

Definition 24 A *goal list* is a finite list $[S_1, \ldots, S_n]$ of signed (quantifier free) formulas.

There are *propositional transformations* of goal list, using the Smullyan rules. For example, a goal list of the form $[\ldots, \alpha, \ldots]$ can be replaced (non-deterministically) by either $[\ldots, \alpha_1, \ldots]$ or $[\ldots, \alpha_2, \ldots]$. Likewise a goal list of the form $[\ldots, \beta, \ldots]$ can be replaced by $[\ldots, \beta_1, \beta_2, \ldots]$. And so on. Finally, $F_a\,\langle x, y\rangle$ can be dropped from a goal list if $x \geq a$, and similarly for $T_a\,\langle x, y\rangle$. We omit a detailed statement of these rules.

Finally there are *program calls*. Suppose P is a program and $[\ldots, T_b\, A, \ldots]$ is a goal list, where A is atomic. Say $A' \leftarrow \phi$ is a program clause of P, and σ is a most general unifier of A and A'. Then $[\ldots, T_b\, \phi, \ldots]\sigma$ results from $[\ldots, T_b\, A, \ldots]$ by a *P-program call*. There is a similar notion when $F_a\, A$ rather than $T_b\, A$ is in the goal list.

Definition 25 A *P-derivation* from a goal list $[S_1, \ldots, S_n]$ is a sequence of goal lists beginning with $[S_1, \ldots, S_n]$, with each goal list after the first coming from its predecessor by a propositional transformation or by a P-program call. The derivation *succeeds* if it ends with $[\,]$.

The analogy between this and SLD resolution should be clear. Now, soundness can be established in the following form.

Theorem 26 *Suppose A is ground atomic, and $\mu\Phi_P(A) = \langle x, y\rangle$.*

1. *If there is a successful P-derivation from $[F_a\, A]$ then $x \geq a$.*
2. *If there is a successful P-derivation from $[T_a\, A]$ then $y \geq a$.*

Completeness is a little more of a problem. The following generally holds.

Theorem 27 *Again suppose A is ground atomic, and $\mu\Phi_P(A) = \langle x, y\rangle$.*

1. *If $x > a$ then there is a successful P-derivation from $[F_a\, A]$.*
2. *If $y > a$ then there is a successful P-derivation from $[T_a\, A]$.*

Whether or not $>$ can be replaced by $\geq$ in the Theorem above depends on the lattice L. If L is $[0, 1]$, it is not the case. On the other hand, if L is discrete, it is.

Finally, the programming system of [21] included attenuation factors associated with program clauses. I have not considered these here, for reasons of space and general interest. In fact, it is possible to introduce a version of them into the $[0, 1] \odot [0, 1]$ bilattice setting. The nicest way is to consider attenuation factors to be operators, something like negation. I will have to leave a proper discussion of them for elsewhere, just as I must leave proofs of the soundness and completeness results for elsewhere.

I hope I have been able to accomplish what I intended. I wanted to show that bilattices are both natural and useful tools for logic programming, and are worth pursuing for this reason as well as for their own sakes.

References

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

[2] K. R. Apt, H. A. Blair and A. Walker, Towards a theory of declarative knowledge, in *Foundations of Deductive Databases and Logic Programming*, Jack Minker editor, pp 89–148, Morgan-Kauffmann (1987).

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

[4] K. L. Clark, Negation as failure, *Logic and Databases*, H. Gallaire and J. Minker editors, pp 293–322, Plenum Press, New York (1978), reprinted in *Readings in Nonmonotonic Reasoning*, M. L. Ginsberg editor, pp 311–325, Morgan Kaufmann, Los Altos, CA (1987).

[5] M. C. Fitting, A Kripke/Kleene semantics for logic programs, *Journal of Logic Programming*, pp 295–312 (1985).

[6] M. C. Fitting, Logic programming on a topological bilattice, *Fundamenta Informaticae* pp 209–218, vol 11 (1988).

[7] M. C. Fitting, Negation as refutation, Proceedings Fourth Annual Symposium on Logic in Computer Science, pp 63–70, IEEE Computer Society Press (1989).

[8] M. C. Fitting, Bilattices and the theory of truth, *Journal of Philosophical Logic*, vol 18, pp 225–256 (1989).

[9] M. C. Fitting, Bilattices and the semantics of logic programming, *Journal of Logic Programming* (to appear).

[10] M. C. Fitting and M. Ben-Jacob, Stratified and three-valued logic programming semantics, *Logic Programming, Proc. of the Fifth International Conference and Symposium*, R. A. Kowalski and K. A. Bowen editors, pp 1054–1069, MIT Press (1988).

[11] M. C. Fitting and M. Ben-Jacob, Stratified, weak stratified and three-valued semantics, *Fundamenta Informaticae* (to appear).

[12] M. L. Ginsberg, Multi-valued logics, *Proc. AAAI-86, fifth national conference on artificial intelligence*, pp 243–247, Morgan Kaufmann Publishers (1986).

[13] M. L. Ginsberg, Multivalued Logics: A Uniform Approach to Inference in Artificial Intelligence, *Computational Intelligence*, vol 4, no. 3.

[14] S. C. Kleene, *Introduction to Methmathematics*, Van Nostrand (1950).

[15] K. Kunen, Negation in logic programming, *J. Logic Programming*, pp 289–308 (1987).

[16] K. Kunen, Signed data dependencies in logic programs, *Journal of Logic Programming*, vol 7, pp 231–245 (1989).

[17] J. Lloyd, *Foundations of Logic Programming*, Springer-Verlag (1984).

[18] S. Morishita, Lattice-valued Programs, TRL Research Report, TR87-0041, IBM Tokyo Research Laboratory, 5-19 Sanbancho, Chiyoda-ku, Tokyo 102 (1988).

[19] R. Smullyan, *First-Order Logic*, Springer-Verlag (1968).

[20] A. Tarski, A lattice-theoretical fixpoint theorem and its applications, *Pacific Journal of Mathematics*, vol 5, pp 285–309 (1955).

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

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