Bilattices and the Semantics of Logic Programming

Melvin Fitting
MLFLC@CUNYVM.BITNET
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 *

February 9, 1989

Abstract

Bilattices, due to M. Ginsberg, are a family of truth value spaces that allow elegantly for missing or conflicting information. The simplest example is Belnap's four-valued logic, based on classical two-valued logic. Among other examples are those based on finite many-valued logics, and on probabilistic valued logic. A fixed point semantics is developed for logic programming, allowing any bilattice as the space of truth values. The mathematics is little more complex than in the classical two-valued setting, but the result provides a natural semantics for distributed logic programs, including those involving confidence factors. The classical two-valued and the Kripke/Kleene three-valued semantics become special cases, since the logics involved are natural sublogics of Belnap's logic, the logic given by the simplest bilattice.

1 Introduction

Often useful information is spread over a number of sites ("Does anybody know, did Willie wear a hat when he left this morning?") that can be specifically addressed ("I'm not just talking to hear myself talk; I'm asking you, you and you."). Sometimes information is collective ("I know where he keeps his hat when he isn't wearing it." "I know it's not there now."). Sometimes things are more direct ("Yes he did."). All this is routine. But problems arise when answers conflict and must be reconciled. Reasonable possibilities are: if anyone says yes then it's yes; if anyone says no, then it's no. These correspond to the classical truth functional connectives $\vee$ and $\wedge$. But equally reasonable are: different answers tell me nothing; different answers tell me too much. These versions leave the realm of classical logic behind. In fact, a four-valued logic was introduced by Belnap [3] for precisely this purpose.

Logic programs can be distributed over many sites, and can exhibit behavior like that considered in the previous paragraph. It turns out that a simple fixed point semantics, similar to that of [21]

*Research partly supported by NSF Grant CCR-8702307 and PSC-CUNY Grant 6-67295.

and [1], can be developed, based on Belnap's logic (see [4] for a version). But we need not stop here. Van Emden has proposed using real numbers in $[0, 1]$ as quantitative truth values [20]. How should such a truth value space be modified if programs are distributed? Similar issues arise for any choice of truth value space, of course.

M. Ginsberg has invented the elegant notion of bilattice ([14], [15]), which deals with precisely this issue. We reserve the definition till later on, but for motivation we note: Belnap's four valued logic constitutes the simplest bilattice; a natural bilattice can be constructed based on any 'reasonable' truth value space; bilattices provide a truth value mechanism suitable for information that contains conflicts or gaps; and a logic programming fixed point semantics can be developed relative to any bilattice, rather easily at that. Demonstrating the truth of these assertions occupies the bulk of this paper.

Several varieties of fixed point semantics have been proposed for logic programming. The traditional one has been classical two-valued ([1], [21]). This is very satisfactory when negations are not allowed in clause bodies. A three-valued semantics has been urged ([7], [8], [17], [18]) as a way of coping with the problems of negation. Also the two valued semantics has been extended via the device of stratification [2]. See [12] and [13] for a discussion of the relationship between the stratified and the three valued approach. More recently a four valued logic has been proposed, to allow for programs containing inconsistencies [5]. Also the set of reals in $[0, 1]$ has been used as a kind of truth value space, to allow treating confidence factors as truth values [20]. And in [10], sets of possible worlds in a Kripke model were considered, as something like evidence factors. What do these approaches have in common? There must be enough machinery to ensure the existence of fixed points of the operator associated with a program. This means a partial ordering meeting special conditions must exist. [4] is a nice study of this point, and the wide range of circumstances under which the necessary conditions are obtained. But for the machinery to work smoothly, there must be an appropriate interaction between the logical operations allowed in the programming language and the underlying partial ordering. This is an issue addressed here. We claim that bilattices apply naturally to most of the kinds of fixed point semantics people have considered, and provide an account of the intended partial ordering, the truth functional connectives, and the interactions between them. Indeed, bilattices even suggest other operations we might want to incorporate into a logic programming language. In short, although a complete partial ordering (or a complete semi-lattice) is behind most proposed semantics, we believe the mathematical structure is actually richer than that in practice, and bilattices are the right abstract tools. Furthermore, we will see below that one of the nice 'extras' that comes from the bilattice approach is that both the two and the three valued semantical theories follow easily from work on Belnap's four-valued version (because two and three valued logics are natural sublogics of the four-valued logic). And this is not unique to the four-valued case; with no more work similar results can be established for bilattices in general, under certain simple additional conditions. Finally, there are nice relationships between the two and the three valued approaches (for programs where both can be applied). The bilattice setting makes possible an algebraic formulation of this connection, and a considerable extension of it. Without the use of bilattices the very possibility of such a formulation is doubtful. We have hopes that results like Theorem 7.7 will help to shed more light on the role of negation in logic programming.

In short, we believe bilattices are a very natural framework for the general consideration of logic programming semantics. We intend to justify this belief.

2 The Four Valued Case

We begin with the simplest example — the four valued logic $\mathcal{FOUR}$ due to Belnap ([3], [22]). And we begin with a motivation for considering it.

Suppose we have a fixed logic programming language L, whose details need not concern us just now. We assume clause heads are atomic, and clause bodies can contain negations, and also conjunctions and disjunctions, explicit or implicit. And say $\mathcal{P}$ is a program written in the language L. But, let us suppose $\mathcal{P}$ is distributed over a number of sites, with some provision for interaction. Some predicates might be local to a site, while others may be global. We issue a query to the network of sites. Each site attempts to answer the query by the usual mechanism of replacing one question by a list of others. Subsequent queries involving local predicates are handled locally, while those involving global predicates are broadcast to the system in the same way that my initial query was. Details of variable binding will not be gone into here — indeed, for our purposes we will identify a program with the set of ground instances of its clauses. But even with this simple model a fundamental problem arises: what do we do with conflicting responses? If we ask the system ?-q, a site having clauses covering q may respond with 'yes', while a site having no clauses for q will, via negation as failure, respond with 'no'. How should we reconcile these?

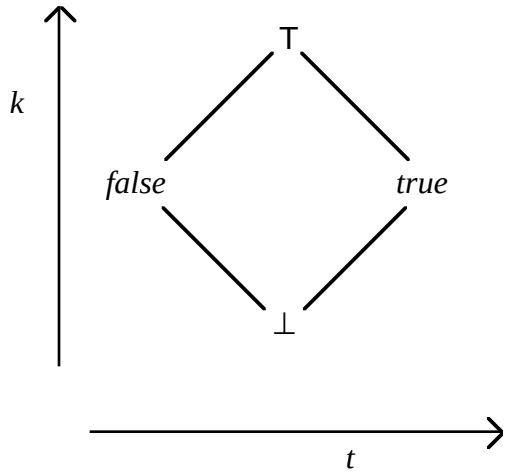
Several simple minded solutions are possible, some of which take us beyond classical truth values. We could, for instance, insist on a consensus. Then, faced with conflicting answers we could say we have no information. Or we could treat each site as an expert whose opinions we value, and so when faced with the problem of two answers, we accept them both, and simply record the existence of a conflict.

Belnap's four valued logic, which we call $\mathcal{FOUR}$ provides the right setting for this. One can think of his truth values as *sets* of ordinary truth values: $\{true\}$, which we will write as *true*; $\{false\}$, which we will write as $false$; $\emptyset$, which we will write as $\bot$ and read as *undefined*, and $\{true, false\}$, which we will write as $\top$ and read as *overdefined*.

Belnap noted that two natural partial orderings existed for these truth values. The subset relation is one; it can be thought of as an ordering reflecting information content. Thus $\{true, false\}$ contains more information than $\{true\}$ say. The other ordering expresses 'degree of truth'; for example, $\emptyset$ has a higher degree of truth than $\{false\}$, precisely because it does not contain $false$. Thus we might call a truth value t_1 *less-true-or-more-false* than t_2 if t_1 contains $false$ but t_2 doesn't, or t_2 contains *true* but t_1 doesn't. Both of these are natural orderings to consider. Ginsberg had the insight to see there are intimate interconnections and to generalize them. Figure 1 is a double Hasse diagram displaying the two partial orderings of $\mathcal{FOUR}$ at once. The *knowledge* or *information* direction is vertical (labeled k), while the *truth* direction is horizontal (labeled t). Thus $a \leq_k b$ if there is an upward path from a to b, and $a \leq_t b$ if there is a rightward path from a to b.

Both partial orderings give us a complete lattice. In particular meets and joins exist in each direction. We use the notation $\wedge$ and $\vee$ for finite meet and join, $\bigwedge$ and $\bigvee$ for arbitrary meet and join, in the $\leq_t$ ordering. And we use $\otimes$ and $\oplus$ for finite meet and join, $\prod$ and $\sum$ for arbitrary

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



meet and join in the $\leq_k$ ordering. Negation we define directly: $\neg true = false$; $\neg false = true$; $\neg\top = \top$; $\neg\bot = \bot$. Thinking of four valued truth values as sets of ordinary truth values, this negation amounts to the application of ordinary negation to each member of the set.

Kleene's strong three valued logic [16] and ordinary two valued logic are present as sublattices. The operations of the $\leq_t$ ordering, and negation, when restricted to $false$ and $true$ are the classical ones, and when restricted to $false$, $true$ and $\bot$, are Kleene's. Thus $\mathcal{FOUR}$ retains all the mathematical machinery of the two and three valued logics that have worked so well, and gives us complete lattices, thus simplifying the technical setting somewhat.

Note that the operations induced by the $\leq_k$ ordering also have a natural interpretation. Combining truth values using $\otimes$ amounts to the consensus approach to conflicts mentioned above, while the use of $\oplus$ corresponds to the accept-anything version. This suggests that counterparts of these operations should be part of a logic programming language designed for distributed implementation.

Now, the meaning assigned to a program $\mathcal{P}$ will be a model, but a four valued one. As in classical logic, we need a domain $\mathbf{D}$, but in logic programming this domain is generally thought of as fixed. In practice it is the Herbrand universe, or maybe the space of regular trees, or some such thing. The details need not concern us now. We assume ground terms of the language L name members of $\mathbf{D}$, and every member of $\mathbf{D}$ has a unique such name, so we can just identify $\mathbf{D}$ with the set of ground terms. (A more general treatment allowing arbitrary domains is easily possible.) With this understood, we generally surpress mention of $\mathbf{D}$ for now. All that is left is to characterize *valuations.*

Definition 2.1 *A* valuation v *in* $\mathcal{FOUR}$ *is a mapping from the set of ground atomic formulas of* L *to* $\mathcal{FOUR}$. *Valuations are given two pointwise orderings as follows.* $v_1 \leq_k v_2$ *if* $v_1(A) \leq_k v_2(A)$ *for every ground atomic formula* A. $v_1 \leq_t v_2$ *if* $v_1(A) \leq_t v_2(A)$ *for every ground atomic formula* A.

The set of valuations constitutes a complete lattice under each of the induced orderings $\leq_k$ and $\leq_t$, and inherits much of the structure of $\mathcal{FOUR}$ in a natural way. Valuations can be extended to maps from the set of all ground formulas to $\mathcal{FOUR}$ in the following way. Here we assume formulas are built up from the atomic level using $\neg$, $\wedge$, $\vee$, maybe $\forall$, $\exists$, and maybe operation symbols $\otimes$, $\oplus$, $\prod$ and $\sum$.

Definition 2.2 *A valuation v determines a unique map, also denoted v, on the set of all ground formulas, according to the following conditions:*

1. $v(\neg X) = \neg v(X)$
2. $v(X \circ Y) = v(X) \circ v(Y)$ *for* $\circ$ *one of* $\wedge$, $\vee$, $\otimes$ *or* $\oplus$
3. $v((\forall x)P(x)) = \bigwedge_{d \in \mathbf{D}} v(P(d))$
4. $v((\exists x)P(x)) = \bigvee_{d \in \mathbf{D}} v(P(d))$
5. $v((\prod x)P(x)) = \prod_{d \in \mathbf{D}} v(P(d))$
6. $v((\sum x)P(x)) = \sum_{d \in \mathbf{D}} v(P(d))$.

Now the idea of a fixed point semantics can be given an intuitively appealing presentation. We want to associate with a program $\mathcal{P}$ an operator $\Phi_{\mathcal{P}}$ that maps valuations to valuations. It should calculate values for clause bodies in $\mathcal{P}$ according to the definition above. In particular, the classic truth functional operations have their meaning supplied by the $\leq_t$ ordering. And we want to find the least fixed point of $\Phi_{\mathcal{P}}$, but least in the $\leq_k$ direction: the fixed point containing no extraneous information. In order to do this we need to know that the various operations interact well with the $\leq_k$ ordering, so that a least fixed point is guaranteed to exist. For example, the behavior of the operation $\wedge$, defined using $\leq_t$, must mesh somehow with the properties of $\leq_k$ so that monotonicity of the operator $\Phi_{\mathcal{P}}$ is ensured.

In fact, this always happens. Moreover, what happens relates well to, and generalizes the two and three valued approaches already in the literature. The four valued setting is explored further in [11], and with certain restrictions placed on the language, it is shown to provide a natural and usable extension of conventional logic programming. An implementation on top of Prolog has been developed, and the result is a language that behaves well when negations and free variables are both allowed to occur in queries. But, rather than proceeding further with this four valued example here, we move to the broader setting of *bilattices*, establish our results in greater generality, and apply them to $\mathcal{FOUR}$ and other particular cases later on.

3 Bilattices

M. Ginsberg ([14], [15]) has proposed bilattices as an elegant generalization of $\mathcal{FOUR}$. We need a somewhat more restricted notion, which we call an *interlaced* bilattice. The Ginsberg references should be consulted for the underlying bilattice definition and examples.

Definition 3.1 *An* interlaced bilattice *is a set* $\mathbf{B}$ *together with two partial orderings $\leq_k$ and $\leq_t$ such that:*

1. *each of* $\leq_k$ *and* $\leq_t$ *gives* **B** *the structure of a complete lattice;*
2. *the meet and join operations for each partial ordering are monotone with respect to the other ordering.*

We call condition 2) the *interlacing* condition. We use notational conventions similar to those of the previous section, with an arbitrary interlaced bilattice. $\wedge$ and $\vee$ are finite meet and join under the $\leq_t$ ordering; $\bigwedge$ and $\bigvee$ are arbitrary meet and join under $\leq_t$. $\otimes$ and $\oplus$ are finite meet and join under $\leq_k$; $\prod$ and $\sum$ are arbitrary meet and join under $\leq_k$. $false$ and $true$ are least and greatest elements under $\leq_t$; $\bot$ and $\top$ are least and greatest elements under $\leq_k$.

If $a \leq_t b$ and $c \leq_t d$ then $a \wedge c \leq_t b \wedge d$, because this is a basic property of meets in a lattice. But condition 2) of Definition 3.1 requires also that $a \leq_k b$ and $c \leq_k d$ imply $a \wedge c \leq_k b \wedge d$. Similarly for other operation/ordering combinations. Condition 2) expresses the fundamental interconnection that makes an interlaced bilattice more than just two independent lattices stuck together. Condition 2) of Definition 3.1 requires some care in interpretation when infinitary operations are involved. We take its meaning in such cases as follows. Suppose A and B are subsets of **B**. We write $A \leq_t B$ if: for each $a \in A$ there is some $b \in B$ with $a \leq_t b$; and for each $b \in B$ there is some $a \in A$ with $a \leq_t b$. Then condition 2) is taken to require that if $A \leq_t B$ then $\prod A \leq_t \prod B$. And of course similarly for other operator/ordering combinations.

$\mathcal{FOUR}$ is an example of an interlaced bilattice. This can be checked directly; also it follows from more general results later on. We tacitly assume all interlaced bilattices are non-trivial. $\mathcal{FOUR}$, then, is the simplest example, and is isomorphically a sub-bilattice of all other bilattices. On the other hand, there are bilattices as defined in [14] and [15] that are not interlaced. Figure 2 displays a bilattice for a default logic. In it, *df* is intended to represent a value of default falsehood, while *dt* represents default truth. This does not satisfy the interlacing condition though. For instance, $false \leq_t df$ but $false \otimes * = *$ and $df \otimes * = df$, so $false \otimes * \leq_t df \otimes *$ fails.

Proposition 3.1 *In any interlaced bilattice,*

1. $true \oplus false = \top$; $true \otimes false = \bot$;
2. $\top \vee \bot = true$; $\top \wedge \bot = false$.

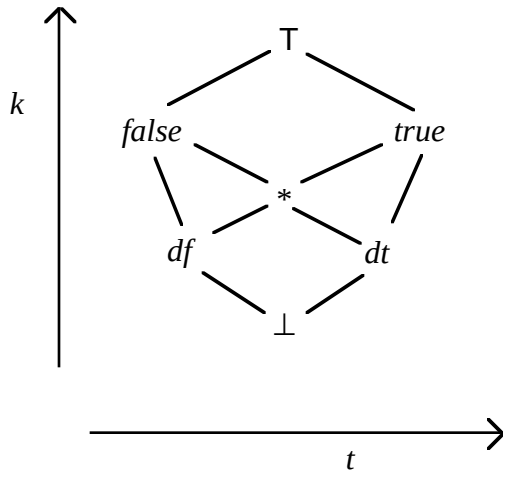
Proof It is easy to see that $a \oplus \top = \top$ for all a. Also since $false$ is the smallest element under $\leq_t$, $false \leq_t \top$, so since we have an interlaced bilattice, $a \oplus false \leq_t a \oplus \top$. Taking a to be $true$ we have $true \oplus false \leq_t true \oplus \top = \top$. In the other direction, $\top \leq_t true$, hence $a \oplus \top \leq_t a \oplus true$ for any a. Then $\top = false \oplus \top \leq_t false \oplus true$. These two inequalities establish that $false \oplus true = \top$. The rest of the cases are established in a similar way.

Notice that negation is not part of the basic interlaced bilattice structure. Still, a negation notion generally exists in cases we are interested in. We merely require that it mesh well with the orderings.

Definition 3.2 *An interlaced bilattice* **B** *has a* weak negation operation *if there is a mapping* $\neg : \mathbf{B} \rightarrow \mathbf{B}$ *such that:*

Figure 2: A Bilattice for a Default Logic

.



1. $a \leq_k b \Rightarrow \neg a \leq_k \neg b$;
2. $a \leq_t b \Rightarrow \neg b \leq_t \neg a$.

Note In bilattices as defined in [14] and [15], the existence of a negation operation is basic and, in addition to the conditions above, one must also have that $a = \neg\neg a$. Since we have postulated the rather powerful interlacing conditions here, we do not need the full strength of negation; generally the existence of a weak negation is sufficient. If the additional double negation condition is satisfied, we will say we have a *negation* operation rather than a *weak* negation operation.

The intuition here is straightforward. Negations should reverse truth; one expects that. But negations do not change knowledge; one knows as much about $\neg a$ as one knows about a. Hence a negation operation reverses the $\leq_t$ ordering but preserves the $\leq_k$ ordering.

There is one more basic operation on interlaced bilattices that we want to consider, though its familiarity is considerably less than that of negation. In $\mathcal{FOUR}$, some of the truth values were essentially classical, while others made up a natural three valued logic. What we need is some mechanism for distinguishing those parts of an interlaced bilattice that constitute natural sublogics, and that is what this extra operation is for (more will be said about intuitive motivation in the next section). We call it *conflation*. The basic ideas, though not the name, come from [22]. The intention is, conflation is like negation, but with the roles of $\leq_k$ and $\leq_t$ switched around. In particular, in $\mathcal{FOUR}$, negation 'flips' the diagram in Figure 1 from left to right. We take conflation in $\mathcal{FOUR}$ to be the operation that 'flips' the diagram from top to bottom, interchanging $\top$ and $\bot$,

and leaving $false$ and $true$ alone. It is easy to check that this is a conflation operation, according to the following definition.

Definition 3.3 *An interlaced bilattice* $\mathbf{B}$ *has a* conflation *operation if there is a mapping* $- : \mathbf{B} \rightarrow \mathbf{B}$ *such that:*

1. $a \leq_t b \Rightarrow -a \leq_t -b$;
2. $a \leq_k b \Rightarrow -b \leq_k -a$;
3. $--a = a$.

We said the space of valuations in $\mathcal{FOUR}$ 'inherited' much of the structure of $\mathcal{FOUR}$. Now we can be more precise about what this means.

Definition 3.4 *Suppose* $\mathbf{B}$ *is an interlaced bilattice and* S *is a set. Let* $\mathbf{B}^S$ *be the set of all mappings from* S *to* $\mathbf{B}$*, with induced orderings* $\leq_k$ *and* $\leq_t$ *defined pointwise on* $\mathbf{B}^S$*. That is, for* $f, g \in \mathbf{B}^S$*,* $f \leq_k g$ *provided* $f(s) \leq_k g(s)$ *for all* $s \in S$*, and similarly for* $\leq_t$*. If* $\mathbf{B}$ *has a weak negation operation, this induces a corresponding operation on* $\mathbf{B}^S$ *according to* $(\neg f)(s) = \neg f(s)$*. Similarly for a conflation operation.*

Proposition 3.2 *Suppose* $\mathbf{B}$ *is an interlaced bilattice and* S *is a set. Then* $\mathbf{B}^S$ *is also an interlaced bilattice. The operations in* $\mathbf{B}^S$ *are the pointwise ones:* $(f \otimes g)(s) = f(s) \otimes g(s)$*, and so on. In addition, if* $\mathbf{B}$ *is an interlaced bilattice with weak negation, so is* $\mathbf{B}^S$*, and similarly for conflation.*

Proof A straightforward matter of checking, which we omit.

We use the obvious extension of notation from elements of an interlaced bilattice to subsets: $\neg A = \{\neg a \mid a \in A\}$ and $-A = \{-a \mid a \in A\}$.

Proposition 3.3 *In an interlaced bilattice with conflation,*

1. $-\bigwedge A = \bigwedge -A$; $-(a \wedge b) = (-a \wedge -b)$;
2. $-\bigvee A = \bigvee -A$; $-(a \vee b) = (-a \vee -b)$;
3. $-\prod A = \sum -A$; $-(a \otimes b) = (-a \oplus -b)$;
4. $-\sum A = \prod -A$; $-(a \oplus b) = (-a \otimes -b)$.

Proof We demonstrate part 1); the other parts are similar. If a is an arbitrary member of A then $\bigwedge A \leq_t a$, so $-\bigwedge A \leq_t -a$. It follows that $-\bigwedge A \leq_t \bigwedge -A$. Conversely, for an arbitrary $a \in A$, $-a \in -A$, and so $\bigwedge -A \leq_t -a$. It follows that $-\bigwedge -A \leq_t a$, and so $-\bigwedge -A \leq_t \bigwedge A$, and hence $\bigwedge -A \leq_t -\bigwedge A$.

Definition 3.5 *If* $\mathbf{B}$ *is an interlaced bilattice with weak negation and conflation, we say conflation* commutes with *negation provided* $-\neg a = \neg - a$ *for all* a.

In $\mathcal{FOUR}$ conflation commutes with negation. The notion of commuting extends in an obvious way to binary (and infinitary) operations. Then Proposition 3.3 says conflation commutes with $\wedge$, $\bigwedge$, $\vee$ and $\bigvee$.

Proposition 3.4 *In any interlaced bilattice with conflation,*

1. $-\bot = \top$; $-\top = \bot$;
2. $-false = false$; $-true = true$;

Proof Since $\bot \leq_k a$ for any a, $\bot \leq_k -\top$, and so $\top \leq_k -\bot$. It follows from the definition of $\top$ that $\top = -\bot$. Then easily, $-\top = \bot$.

Since $false \leq_t a$ for any a, it follows from the definition of $\vee$ that $false \vee a = a$ for any a. Using this we have $-false = -false \vee false = -false \vee --false = -(false \vee -false) = -(-false) = false$. Similarly for $-true = true$.

The interlaced bilattices of most interest to us in this paper all have a conflation operation and usually a negation operation. These operations can not be taken for granted though. In [10] we considered a family of interlaced bilattices with conflation and a weak negation operation that satisfied the condition $a \leq_t \neg\neg a$ though not generally $a = \neg\neg a$. Thus the weak negation operation there had an Intuitionistic flavor. The interlaced bilattice displayed in Figure 3 does not have a conflation operation. It has a weak negation operation, but one that does not even meet the intuitionistic condition stated above. (In the diagram of Figure 3, $false$ and b are incomparable under $\leq_k$, while b and $\bot$ are incomparable under $\leq_t$. Similarly for other horizontal or vertical edges.)

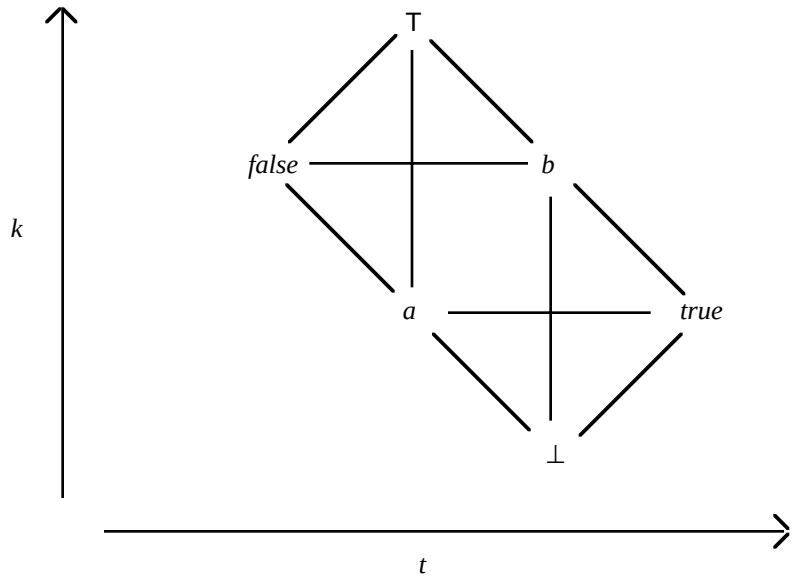
The diagram in Figure 3 does represent an interlaced bilattice, $\mathcal{SIX}$ — we will verify this in Section 4. Checking that there is no conflation operation is simple. If there were one, its behavior on $\top$, $\bot$, $false$ and $true$ would be determined by Proposition 3.4, so only $-a$ and $-b$ need to be specified. If $-a = \top$ then $a = --a = -\top = \bot$, which is not the case. Similarly all of $\top$, $\bot$, $true$ and $false$ are ruled out as possible values for $-a$ and $-b$, so we are forced to have either $-a = a$ or $-a = b$. Both of these are impossible. For instance, since $a \leq_k false$ then $-false \leq_k -a$. If we had $-a = a$ then this would say $false \leq_k a$, which is not the case. $-a = b$ can be ruled out in a similar way. Likewise one can check that there is no negation. There is a weak negation operation though. Take $\neg b = \top$, $\neg a = \bot$, $\neg false = true$, $\neg true = false$, $\neg\top = \top$ and $\neg\bot = \bot$. It is easy to verify that this is a weak negation operation, but since $\neg\neg b = \top$ we do not have $b \leq_t \neg\neg b$.

Now we take up the issue of possible sublogics of an interlaced bilattice that may be of use and interest.

Definition 3.6 *In an interlaced bilattice* $\mathbf{B}$ *with conflation, for* $a \in \mathbf{B}$

1. *a is* exact *if* $a = -a$;
2. *a is* consistent *if* $a \leq_k -a$.

Figure 3: The Interlaced Bilattice $\mathcal{SIX}$



We remark that [22] used other terminology for these notions, but the definitions are essentially from that paper. In $\mathcal{FOUR}$ the exact truth values are the classical ones, $false$ and $true$, and the consistent ones are (of course) these, and also $\bot$. In other words, we get the classical values, and Kleene's values as exact and consistent. Now, the classical values are closed under the $\leq_t$ based operations in $\mathcal{FOUR}$, and similarly for the truth values of Kleene's logic. Likewise Kleene's values yield a complete semilattice under $\leq_k$, a point which played a significant role in [7]. Features like these are, in fact, quite general.

Proposition 3.5 *Let* $\mathbf{B}$ *be an interlaced bilattice with conflation. The exact members include* $false$ *and* $true$ *and are closed under* $\wedge$, $\bigwedge$, $\vee$ *and* $\bigvee$. *If* $\mathbf{B}$ *has a weak negation which commutes with the conflation operation then the exact members are closed under negation as well. The exact members do not include* $\bot$ *or* $\top$ *and are not closed under* $\otimes$ *or* $\oplus$.

Proof $false$ and $true$ are exact by Proposition 3.4. If a and b are exact, by Proposition 3.3 $-(a \wedge b) = -a \wedge -b = a \wedge b$, hence there is closure under $\wedge$. Similarly for the other cases. Neither $\bot$ nor $\top$ is exact, by Proposition 3.4 (and the fact that $\bot \neq \top$). The exact members are not closed under $\otimes$ because then $true \otimes false = \bot$ would be exact. Similarly for $\oplus$.

Definition 3.7 *A* complete semilattice *is a partially ordered set that is closed under arbitrary meets, and under joins of* directed *subsets. A subset* D *is* directed *if for all* $x, y \in D$, *there is a* $z \in D$ *such that* $x \leq z$ *and* $y \leq z$.

Proposition 3.6 *Let* $\mathbf{B}$ *be an interlaced bilattice with conflation. The consistent members include the exact members,* $\bot$*, and are closed under* $\wedge$*,* $\bigwedge$*,* $\vee$*,* $\bigvee$*. If* $\mathbf{B}$ *has a weak negation that commutes with conflation, the consistent members are closed under negation as well. Finally the consistent members constitute a complete semilattice under* $\leq_k$*, being closed under* $\prod$ *and under directed* $\sum$*.*

Proof Exactness trivially implies consistency. If S is a set of consistent members it follows that $S \leq_k -S$ so $\bigwedge S \leq_k \bigwedge -S = -\bigwedge S$, hence the consistent members are closed under $\bigwedge$. The other truth functional closures are similar.

Again, let S be a set of consistent members, so $S \leq_k -S$. Then $\prod S \leq_k \prod -S = -\sum S$. Also $\prod S \leq_k \sum S$, so $-\sum S \leq_k -\prod S$. It follows that $\prod S \leq_k -\prod S$, so $\prod S$ is consistent. Thus we have closure under $\prod$.

Finally, suppose S is a set of consistent members that also is directed by $\leq_k$. To show $\sum S$ is consistent we must show $\sum S \leq_k -\sum S = \prod -S$. For this, it is enough to show that for any $a, b \in S$, $a \leq_k -b$. But, since S is directed and $a, b \in S$, there is some $c \in S$ with $a \leq_k c$ and $b \leq_k c$. Then: $-c \leq_k -b$; c is consistent so $c \leq_k -c$; and hence $a \leq_k -b$.

Note Many of these arguments, including that for being a semilattice, come from [22].

4 Examples

The basic bilattice construction is due to Ginsberg, and is easily described. Suppose $C = \langle C, \leq \rangle$ and $D = \langle D, \leq \rangle$ are complete lattices. (We use the same notation, $\leq$, for both orderings, since context can determine which is meant.) Form the set of points $C \times D$, and give it two orderings, $\leq_k$ and $\leq_t$, as follows.

$\langle c_1, d_1 \rangle \leq_k \langle c_2, d_2 \rangle$ if $c_1 \leq c_2$ and $d_1 \leq d_2$

$\langle c_1, d_1 \rangle \leq_t \langle c_2, d_2 \rangle$ if $c_1 \leq c_2$ and $d_2 \leq d_1$

We denote the resulting structure, $\langle C \times D, \leq_k, \leq_t \rangle$ by $\mathcal{B}(C, D)$

Proposition 4.1 *For complete lattices* C *and* D*,* $\mathcal{B}(C, D)$ *is an interlaced bilattice.*

Proof Straightforward. It is easy to check that, if we use $\sqcap$ and $\sqcup$ for meet and join in both of C and D, then

$\langle c_1, d_1 \rangle \otimes \langle c_2, d_2 \rangle = \langle c_1 \sqcap c_2, d_1 \sqcap d_2 \rangle$

$\langle c_1, d_1 \rangle \oplus \langle c_2, d_2 \rangle = \langle c_1 \sqcup c_2, d_1 \sqcup d_2 \rangle$

$\langle c_1, d_1 \rangle \wedge \langle c_2, d_2 \rangle = \langle c_1 \sqcap c_2, d_1 \sqcup d_2 \rangle$

$\langle c_1, d_1 \rangle \vee \langle c_2, d_2 \rangle = \langle c_1 \sqcup c_2, d_1 \sqcap d_2 \rangle$

and similarly for the infinitary cases.

The intuition here is illuminating. Suppose we think of a pair $\langle c, d\rangle$ in $\mathcal{B}(C, D)$ as representing two independent judgements concerning the truth of some statement: c represents our degree of belief *for* the statement; d represents our degree of belief *against* it. These may, for instance, have been arrived at by asking independent experts. And since C and D can be different lattices, expressions of belief for and against need not be measured in the same way. Now, if $\langle c_1, d_1\rangle \leq_k \langle c_2, d_2\rangle$ then $\langle c_2, d_2\rangle$ embodies more 'knowledge' than $\langle c_1, d_1\rangle$, which is reflected by an increased degree of belief both for and against; perhaps more information has been received which has increased the certainty of our 'experts'. On the other hand, if $\langle c_1, d_1\rangle \leq_t \langle c_2, d_2\rangle$ then $\langle c_2, d_2\rangle$ embodies more 'truth' than $\langle c_1, d_1\rangle$, which is reflected by an increased degree of belief for, and a decreased degree of belief against.

The simplest particular example begins with the simplest non-trivial lattice $\mathcal{TR}$: $\{f\!alse, t\!rue\}$ with $f\!alse \leq t\!rue$. $\mathcal{B}(\mathcal{TR}, \mathcal{TR})$ is simply an isomorphic copy of $\mathcal{FOUR}$ (Figure 1). In this representation, $\bot$ is $\langle f\!alse, f\!alse\rangle$, no belief for and no belief against; similarly $\top$ is $\langle t\!rue, t\!rue\rangle$. Likewise $f\!alse$ is $\langle f\!alse, t\!rue\rangle$, no belief for, total belief against; and $t\!rue$ is $\langle t\!rue, f\!alse\rangle$.

Another example of importance is a quantitative one, based on the complete lattice $[0, 1]$ with the usual ordering $\leq$ of reals. This lattice was used as a space of truth values by van Emden [20]. $\mathcal{B}([0, 1], [0, 1])$ is an interlaced bilattice that bears the same relation to van Emden's system that $\mathcal{FOUR}$ bears to classical logic: it accomodates conflicting quantitative information smoothly.

Kripke models for modal logics provide another family of examples. As Ginsberg suggests, we can think of the set of possible worlds in which a formula is true as the *evidence for* a formula; similarly for false and evidence against. Then, given a particular Kripke model with $\mathcal{W}$ as the set of possible worlds, use the power set lattice $P = \langle \mathbf{P}(\mathcal{W}), \subseteq\rangle$ to create the interlaced bilattice $\mathcal{B}(P, P)$. This is a natural sort of evidence space.

A different family of interlaced bilattices was considered in [10], based on topological spaces arising out of Kripke Intuitionistic Logic models. Let $\mathcal{T}$ be a topological space. The family $\mathcal{O}$ of open sets is a complete lattice under $\subseteq$. Join is union, while meet is interior of intersection. Likewise the family $\mathcal{C}$ of closed sets is a complete lattice under $\subseteq$. In [10] we investigated interlaced bilattices of the form $\mathcal{B}(\mathcal{O}, \mathcal{C})$.

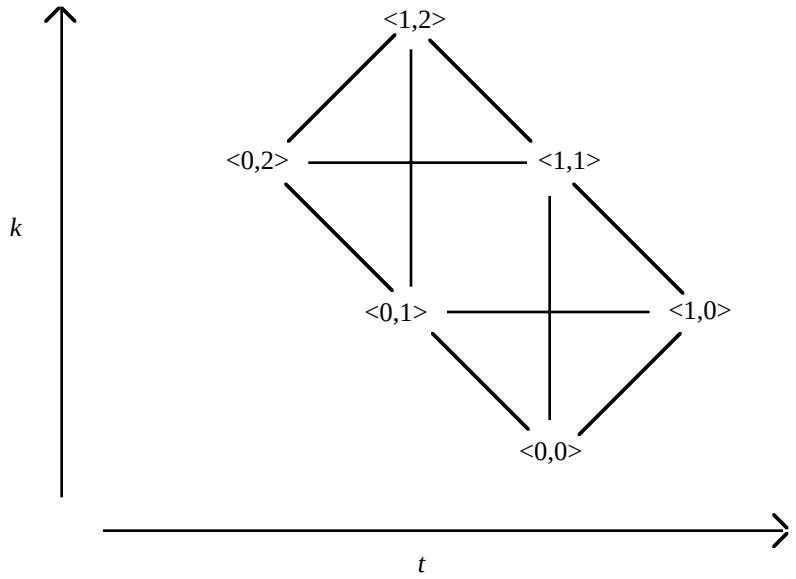
Finally, suppose C is the lattice $\{0, 1\}$ with $0 \leq 1$ and D is the lattice $\{0, 1, 2\}$ with $0 \leq 1 \leq 2$. Then $\mathcal{B}(C, D)$, shown in Figure 4, is isomorphic to the interlaced bilattice $\mathcal{SIX}$, given in Figure 3, which incidentally verifies that $\mathcal{SIX}$ is an interlaced bilattice.

If the two lattices being combined to create an interlaced bilattice are the same, then a negation is easy to define.

Definition 4.1 *In $\mathcal{B}(C, C)$, let $\neg\langle a, b\rangle = \langle b, a\rangle$.*

The intuition, once again, is appealing. In passing from $\langle a, b\rangle$ to $\neg\langle a, b\rangle$ we are reversing evidence roles, counting for what was counted against, and conversely. Of course doing this presupposes that evidence for and against is measured the same way, hence the need for a single underlying lattice. This also suggests why the interlaced bilattice $\mathcal{SIX}$, in Figure 3, was plausible as a candidate for

Figure 4: The Interlaced Bilattice $\mathcal{B}(C, D)$



one without a negation. The verification that the operation defined above is a negation is simple, and is omitted.

The definition of negation does not apply to the family of interlaced bilattices based on open and closed sets of a topological space, since it presupposes that only one lattice is involved, rather than two different ones. But for such an interlaced bilattice, we can take $\neg\langle O, C\rangle$, where O is an open set and C is a closed set, to be $\langle interior(C), closure(O)\rangle$. In general this gives a weak negation satisfying the condition $a \leq_t \neg\neg a$, though not the full negation condition.

For a conflation, additional lattice machinery is necessary.

Definition 4.2 *We say a lattice has an* involution *if there is a one-to-one mapping, denoted* $-$*, such that* $a \leq b$ *implies* $-b \leq -a$*.*

For example, in $\mathcal{TR} = \{false, true\}$, classical negation is an involution. In $[0, 1]$ the map taking x to $1 - x$ is an involution. In a power set lattice $P = \langle \mathbf{P}(\mathcal{W}), \subseteq\rangle$, based on a Kripke model say, complementation is an involution.

Proposition 4.2 *If* C *is a complete lattice with an involution, then* $-\langle a, b\rangle = \langle -b, -a\rangle$ *is a conflation in* $\mathcal{B}(C, C)$*, that commutes with negation defined as above.*

Proof Straightforward.

Once again, there is a loose intuition at work here. In passing from $\langle a, b\rangle$ to $-\langle a, b\rangle = \langle -b, -a\rangle$ we are moving to 'default' evidence. We are now counting for whatever was not counted against

before, and against what was not counted for. Now our definitions of consistent and exact can be given further motivation. In $\mathcal{B}([0,1],[0,1])$ for instance, a truth value $\langle x, y\rangle$ is consistent if $x+y \leq 1$ and exact if $x + y = 1$. Likewise in $\mathcal{B}(P, P)$ where P is $\langle \mathbf{P}(\mathcal{W}), \subseteq\rangle$, $\langle x, y\rangle$ is consistent if $x \cap y = \emptyset$, and exact if x and y are complementary. Generally the idea is that consistency means there is no conflict between our beliefs for and against; exactness means these beliefs cover all cases.

Conflations based on involutions only make sense when a single underlying lattice is involved, and so this does not apply to the topological examples. But a conflation can be defined there as well. Take $-\langle O, C\rangle$, where O is open and C is closed, to be $\langle \bar{C}, \bar{O}\rangle$, where the overbar denotes complement. This also gives a conflation that commutes with negation, as defined above. Clearly this example can be generalized, to cases where order reversing mappings exist between two lattices — we do not do this here.

There are also examples of interesting and useful operations that make sense only for very special interlaced bilattices. A good instance involves the interlaced bilattice $\mathcal{B}([0,1],[0,1])$. For each real $r \in [0,1]$, let $\bigtriangledown_r$ be the scalar multiplication operation: $\bigtriangledown_r(\langle a, b\rangle) = \langle ra, rb\rangle$. In $\mathcal{B}([0,1],[0,1])$, $\bigtriangledown_r$ is monotone with respect to both the $\leq_k$ and the $\leq_t$ orderings and commutes with negation but not with conflation. $\bigtriangledown_r$ finds a natural use in the logic programming setting later on.

5 Logic Programming Syntax

Logic programming is generally thought of as being relative to the Herbrand universe, with the space $\{false, true\}$ of truth values. Both restrictions are unnecessary, and rather narrow. In this section we set up the syntax for a logic programming language relative to an arbitrary data structure, not just the Herbrand universe. We also introduce syntactical machinery to deal with distributed programs. In the next section we turn to semantical issues.

Definition 5.1 *A* data structure *is a tuple* $\langle \mathbf{D}; \mathbf{R}_1, \ldots, \mathbf{R}_n\rangle$ *where* $\mathbf{D}$ *is a non-empty set, called the* domain*, and* $\mathbf{R}_1, \ldots, \mathbf{R}_n$ *are relations on* $\mathbf{D}$*, called* given *relations. A* work space $\mathcal{W}$ *consists of a data structure* $\langle \mathbf{D}; \mathbf{R}_1, \ldots, \mathbf{R}_n\rangle$ *and a permanent association of relation symbols* $R_1, \ldots, R_n$ *with each of the given relations* $\mathbf{R}_1, \ldots, \mathbf{R}_n$*. The relation symbols* $R_1, \ldots, R_n$ *are* reserved *relation symbols.*

One example of a work space in common use in many Prologs is that of arithmetic: the domain is the integers, the given relations are the (three-place) addition and multiplication relations, and these are associated with the relation symbols *plus* and *times*. Another example is strings: the domain is all words over a finite alphabet; the concatenation relation is given. The usual Herbrand universe can be made into a work space: take the Herbrand universe as the domain, and for each n-place function symbol f, take a given $n+1$-place relation $\mathbf{R}_f$ where $\mathbf{R}_f(x_1, \ldots, x_n, y) \Leftrightarrow y$ is the term $f(x_1, \ldots, x_n)$. In a similar way one can make the rational trees into a work space. Examples like these are considered more fully in [9].

Data structures are really semantic objects, but we need to know a little about them now, in order to specify syntax. Let $\mathcal{W}$ be a work space with data structure $\langle \mathbf{D}; \mathbf{R}_1, \ldots, \mathbf{R}_n\rangle$ and reserved relation symbols $R_1, \ldots, R_n$. $\mathcal{W}$ is fixed for the rest of this section. We want to specify a logic programming language $L(\mathcal{W})$ that 'uses' $\mathcal{W}$.

To begin, $L(\mathcal{W})$ needs constants, to refer to members of $\mathbf{D}$. If $\mathbf{D}$ were, say, the integers, $L(\mathcal{W})$ would have some notion of numeral so that particular integers could be referenced. Since this sort of issue is not central to our present concerns, we simply assume members of $\mathbf{D}$ themselves are allowed as constants in $L(\mathcal{W})$.

Definition 5.2 *The* terms *of* $L(\mathcal{W})$ *are* constants *(members of* $\mathbf{D}$*) and* variables *(*$x_1, x_2, \ldots$*). We generally use* x*,* y*, ... to stand for variables.*

We also assume we have an unlimited supply of relation symbols of all arities.

Definition 5.3 *An* atomic formula *of* $L(\mathcal{W})$ *is an expression* $P(t_1, \ldots, t_k)$ *where* P *is a* k*-place relation symbol and* $t_1, \ldots, t_k$ *are terms of* $L(\mathcal{W})$*. If the terms are members of* $\mathbf{D}$ *the atomic formula is* ground*. If* P *is reserved in* $\mathcal{W}$ *we refer to* $P(t_1, \ldots, t_k)$ *as* reserved *too.*

Logic programs are generally written with implicit quantifiers and connectives. But by applying techniques like those of Clark's completion [6], many clauses for a given relation symbol can be collapsed into a single 'clause'. For example, consider the (conventional) logic program:

$$P(f(x)) \leftarrow A(x), B(x, y).$$
$$P(g(x)) \leftarrow C(x).$$

This converts to

$$P(z) \leftarrow (\exists x)(\exists y)(R_f(x, z) \wedge A(x) \wedge B(x, y)) \vee (\exists x)(R_g(x, z) \wedge C(x)).$$

It will be simplest for us to assume all program clauses are already in such a form, and that only a single clause for each relation symbol is present in any program. Further, by making use of explicit quantifiers, we can assume without loss of generality that all variables present in a clause body must occur in the head. Finally we do not allow reserved relation symbols to appear in clause heads. These represent given relations, not relations we are computing.

Definition 5.4

1. *A* simple formula *is any formula built up from atomic formulas in the usual way using* $\wedge$*,* $\vee$*,* $\neg$*,* $\forall$ *and* $\exists$*.*
2. *A* simple clause *is an expression* $P(x_1, \ldots, x_k) \leftarrow \phi(x_1, \ldots, x_k)$ *where* $x_1, \ldots, x_k$ *are variables,* $P(x_1, \ldots, x_k)$ *is unreserved atomic, and* $\phi(x_1, \ldots, x_k)$ *is a simple formula whose free variables are among* $x_1, \ldots, x_k$*. The atomic formula to the left of the arrow is the* clause head*, and the formula to the right is the* clause body*.*
3. *A* simple program *is a finite set of simple clauses with no relation symbol appearing in more than one clause head.*

Note We call a formula *positive* if it does not contain $\neg$. We call a simple program *positive* if its clause bodies are positive. We use the same terminology with other kinds of programs, defined below.

Next we turn to programs distributed over a number of sites. Say we have a finite, fixed collection of sites S, labeled 1, 2, ..., s. Each site will have its own program clauses. We assume that each site can ask other sites for information. To keep syntactic machinery uncluttered, we merely subscript relation symbols to indicate what site should be asked. Thus $P_3(a, g(b))$ will be taken to mean that site 3 should be asked to establish that $P(a, g(b))$. Since the number of sites is finite and fixed, things like "does anybody know..." can be turned into "does 1 know; does 2 know; ...; does s know." Thus what we need is versatile machinery for combining possibly conflicting answers from various sites. Syntactically we add operations $\otimes$ and $\oplus$. When we come to semantics in the next section, these will correspond to 'knowledge direction' bilattice operations, as we have seen in previous sections.

Definition 5.5

1. *An S* atomic formula *is an atomic formula of the form $P_i(t_1, \ldots, t_k)$ where P is an unreserved relation symbol and $i \in S$; also $R(t_1, \ldots, t_k)$ is an S atomic formula if R is an (unsubscripted) reserved relation symbol. An S* formula *is a formula built up from S atomic formulas using $\wedge$, $\vee$, $\neg$, $\forall$, $\exists$, $\otimes$ and $\oplus$.*
2. *A* local clause *is an expression of the form $P(x_1, \ldots, x_k) \leftarrow \phi(x_1, \ldots, x_k)$ where $P(x_1, \ldots, x_k)$ is unreserved atomic, and $\phi(x_1, \ldots, x_k)$ is an S formula whose free variables are among $x_1, \ldots, x_k$.*
3. *A* local program *is a finite set of local clauses with no relation symbol appearing in the head of more than one local clause.*
4. *A* distributed program *is a set $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_s\}$ of local programs.*

Thus we think of $\mathcal{P}_1$ as the program clauses 'at' site 1, $\mathcal{P}_2$ 'at' site 2, and so on. Thinking of a program as being distributed over multiple sites, with communication by query and response, is suggestive. But we do not intend, here, to consider computational or implementation issues, only semantic ones. And for semantic purposes we only need to record which sites have which clauses — communication mechanisms can be ignored. Thus it is convenient to convert a distributed program into a single entity, which we do as follows. Suppose $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_s\}$ is a distributed program. Rewrite each clause in $\mathcal{P}_i$ by attaching the subscript i to the relation symbol occurring in the clause head (recall, unreserved relation symbols in clause bodies already have subscripts attached, since clause bodies are S formulas). Then let $\mathcal{P}$ be the set of all rewritten clauses, coming from $\mathcal{P}_1$, $\mathcal{P}_2$,..., $\mathcal{P}_s$. We say $\mathcal{P}$ is the *compound program* corresponding to the distributed program $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_s\}$.

A compound program, then, is a single set of clauses, and differs from a simple program by the presence of subscripts (inessential) and by the use of $\otimes$ and $\oplus$ (essential). Semantically we can think of subscripted relation symbols as just different relation symbols, and ignore explicit subscripts from now on. Then simple programs become special cases of compound programs; compound programs

differ from simple ones by allowing $\otimes$ and $\oplus$. (We could also allow $\prod$ and $\sum$; it does not affect the semantics, but we do not know of a use for them.) It is the semantics of compound programs that we take up in the next section.

6 Logic Programming Semantics

For this section, *program* means compound program, as characterized in the previous section. Let $\langle \mathbf{D}; \mathbf{R}_1, \ldots, \mathbf{R}_n \rangle$ be a data structure, fixed for this section, with a work space $\mathcal{W}$ based on it that assigns the reserved relation symbol R_i to the given relation $\mathbf{R}_i$. We use a logic programming language $L(\mathcal{W})$, relative to this work space, as defined in the previous section. Also let $\mathbf{B}$ be an interlaced bilattice, fixed for the section.

Definition 6.1 *An* interpretation *is a mapping v from ground atomic formulas to $\mathbf{B}$. The interpretation is* in *the work space $\mathcal{W}$ provided, for each given relation $\mathbf{R}_i$, if $\mathbf{R}_i(d_1, \ldots, d_k)$ is true then $v(R_i(d_1, \ldots, d_k)) = true$ and if $\mathbf{R}_i(d_1, \ldots, d_k)$ is not true then $v(R_i(d_1, \ldots, d_k)) = false$.*

Note *true* and *false*, in the definition above, refer to the least and greatest elements of $\mathbf{B}$ in the $\leq_t$ ordering. As defined, given relations are classical, hence interpretations *in* a work space assign to atomic formulas involving reserved relation symbols only 'classical' truth values. This restriction can be relaxed for many of the results below, and so a more general notion of work space can be used, for instance allowing 'partial' relations as in [12] and [13]. We keep to classical values now for given relations for simplicity of presentation.

Definition 6.2 *The orderings $\leq_t$ and $\leq_k$ are extended to interpretations pointwise. That is, $v_1 \leq_t v_2$ provided, for each closed atomic formula A, $v_1(A) \leq_t v_2(A)$, and similarly for $\leq_k$.*

Given a closed formula ϕ and an interpretation v, a 'truth value' $v(\phi)$ in $\mathbf{B}$ is determined in the obvious way, by associating the operation symbol $\wedge$ with the bilattice operation $\wedge$, $\forall$ with $\bigwedge$, and so on. We assume that negations are allowed in ϕ only in cases where $\mathbf{B}$ has a weak negation operation. The pointwise ordering, though defined using ground *atomic* formulas, extends to arbitrary closed formulas.

Proposition 6.1 *Let ϕ be a closed formula and let v_1 and v_2 be interpretations.*

1. *$v_1 \leq_k v_2$ implies $v_1(\phi) \leq_k v_2(\phi)$;*
2. *$v_1 \leq_t v_2$ implies $v_1(\phi) \leq_t v_2(\phi)$ provided ϕ is positive.*

Proof Immediate from the definitions of interlaced bilattice and negation.

We are about to associate an operator with each program. Before we do, we must decide what action to take for atomic formulas that do not match the head of any clause of a program. The two obvious possibilities are: assign such formulas the value $\bot$; assign the value $false$. It is technically convenient, as well as being consistent with negation as failure, to use the value $false$ in such cases.

Definition 6.3 *Let $\mathcal{P}$ be a compound program in the work space $\mathcal{W}$. $\Phi_{\mathcal{P}}$ is the mapping from interpretations to interpretations such that, for an interpretation v:*

1. *for a reserved relation symbol R_i, $\Phi_{\mathcal{P}}(v)(R_i(d_1, \ldots, d_k)) = true$ if $\mathbf{R}_i(d_1, \ldots, d_l)$ is true and $\Phi_{\mathcal{P}}(v)(R_i(d_1, \ldots, d_k)) = false$ if $\mathbf{R}_i(d_1, \ldots, d_l)$ is false;*
2. *for an unreserved relation symbol Q, if there is a clause in $\mathcal{P}$ of the form $Q(x_1, \ldots, x_k) \leftarrow \phi(x_1, \ldots, x_k)$ then $\Phi_{\mathcal{P}}(v)(Q(d_1, \ldots, d_k)) = v(\phi(d_1, \ldots, d_k))$;*
3. *for an unreserved relation symbol Q, if there is no clause in $\mathcal{P}$ of the form $Q(x_1, \ldots, x_k) \leftarrow \phi(x_1, \ldots, x_k)$ then $\Phi_{\mathcal{P}}(v)(Q(d_1, \ldots, d_k)) = false$.*

It is in part 2) that we use our assumption that all variables occurring in a clause body also occur in the head. Part 1) says that $\Phi_{\mathcal{P}}(v)$ will be an interpretation *in* the given work space. What is more, it is immediate from Proposition 6.1 that monotonicity extends to operators generally.

Proposition 6.2 *Let $\mathcal{P}$ be a compound program.*

1. *if $v_1 \leq_k v_2$ then $\Phi_{\mathcal{P}}(v_1) \leq_k \Phi_{\mathcal{P}}(v_2)$;*
2. *if $v_1 \leq_t v_2$ then $\Phi_{\mathcal{P}}(v_1) \leq_t \Phi_{\mathcal{P}}(v_2)$ provided $\mathcal{P}$ is positive.*

The family of interpretations is a complete lattice under both the $\leq_t$ and the $\leq_k$ orderings. By the Knaster-Tarski Theorem [19], least and greatest fixed points exist in a complete lattice for any monotone mapping. We suggest that a natural semantical meaning for a compound program $\mathcal{P}$ is the least fixed point of $\Phi_{\mathcal{P}}$ under the $\leq_k$ ordering. It is a program model that interprets the various operations reasonably, and contains the least information consistent with the program $\mathcal{P}$.

Proposition 6.2 continues to hold, of course, if other operation symbols are allowed in clause bodies, provided they are interpreted by monotone operations. For instance, in Section 4 we introduced a family of scalar multiplication operators $\bigtriangledown_r$ in the interlaced bilattice $\mathcal{B}([0,1],[0,1])$, and we observed that they were monotone with respect to each ordering. Thus operator symbols for them could be introduced into clause bodies when $\mathcal{B}([0,1],[0,1])$ is being used as the space of truth values, and all appropriate fixed points would still exist. Application of $\bigtriangledown_r$ amounts to a proportional diminuation of the degree of certainty. Van Emden built counterparts of these operators into his logic programming system [20]. They cause no complication in the present interlaced bilattice setting.

7 Connections

When programs are restricted in various ways, more can be said about the extremal fixed points. The next few propositions consider this. For the rest of this section, semantics is with respect to an interlaced bilattice **B**. Some of the results use the notions of exact or consistent, and hence require a conflation operation. If a weak negation operation is present, we always assume it commutes with conflation if there is a conflation operation.

Definition 7.1 *Assuming* **B** *has a conflation operation, an interpretation v is* consistent *(or* exact*) if, for each ground atomic formula A, $v(A)$ is consistent (or exact).*

Note This is equivalent to saying v is consistent (or exact) in the interlaced bilattice of interpretations, using the induced conflation operation of Proposition 3.2.

Proposition 7.1 *Suppose $\mathcal{P}$ is a simple program, and* **B** *has a conflation operation. Then the least fixed point of $\Phi_{\mathcal{P}}$ under the $\leq_k$ ordering is consistent.*

Proof One 'approximates' to the least fixed point of a monotone operator in a complete lattice through a transfinite sequence of steps. We show by transfinite induction that every member of such an approximation sequence is consistent; it follows that the limit (which must be attained at some ordinal) is also consistent. The initial term in the approximation sequence is the smallest member of the complete lattice. In our case, the smallest interpretation under the $\leq_k$ ordering is the one that gives ground atomic formulas the value $\bot$. Such an interpretation is consistent.

Next, having defined the α^{th} term of the approximation sequence, the $\alpha + 1^{st}$ results from it by applying the operator $\Phi_{\mathcal{P}}$. But Proposition 3.6 says the family of consistent truth values is closed under the operations allowed in the body of a simple program. Also the given relations have consistent (indeed exact) truth values. It follows that $\Phi_{\mathcal{P}}$, applied to a consistent interpretation, yields another consistent interpretation. Thus if the α^{th} term of the approximation sequence is consistent, so is the $\alpha + 1^{st}$.

Finally, at a limit ordinal λ one takes the *sup* of interpretations associated with smaller ordinals (a collection which constitutes a chain). But again, Proposition 3.6 says we have closure of consistent truth values under directed $\sum$. Then, if every member of the approximation sequence up to λ is consistent, it follows that the λ^{th} member also is consistent. This completes the transfinite induction proof.

In the simplest case, where the interlaced bilattice is $\mathcal{FOUR}$, this proposition says the least fixed point in the $\leq_k$ ordering, for a simple program, must actually be an interpretation in Kleene's strong three valued logic. It is not hard to check that it is the same as the least fixed point assigned by the three-valued semantics discussed in [7]. Indeed, the three-valued operator denoted Φ in [7] coincides with the four-valued bilattice operator, when restricted to consistent truth values. In other words, the three-valued semantical approach is subsumed by the present, more general one. Indeed, the bilattice approach provides nicely suggestive terminology and motivation for the three-valued semantics. The operations used are truth-functional, being based on the $\leq_t$ ordering. But the least fixed point is evaluated in the $\leq_k$ ordering, minimizing knowledge rather than truth. It is this shift from truth to knowledge that makes a smooth treatment of negation possible.

Proposition 7.2 *Suppose $\mathcal{P}$ is a simple, positive program, and* **B** *has a conflation operation. Then the least and the greatest fixed points of $\Phi_{\mathcal{P}}$ under the $\leq_t$ ordering are exact.*

Proof The argument for the least fixed point is essentially the same as in the preceeding proof, except that Proposition 3.5 must be used instead of Proposition 3.6. Also the initial term of the approximation sequence must be smallest in the $\leq_t$ ordering now, so it should assign to ground atomic formulas the value $false$ instead of $\bot$. The argument for the greatest fixed point is similar, except that one constructs the approximation sequence down from the top instead of up from the

bottom, beginning with the biggest interpretation, which assigns to ground atomic formulas the value *true*, and at limit ordinals one takes infs, $\bigwedge$, instead of sups.

Again in the simplest case, using the interlaced bilattice $\mathcal{FOUR}$, the proposition above says the least fixed point and the greatest fixed point, under the $\leq_t$ ordering, must be classical two-valued interpretations. In fact, for a simple, positive program $\mathcal{P}$, these are the least and greatest fixed points of the well-known $T_{\mathcal{P}}$ operator, from [1]. So the classical semantics developed for positive programs is also a special case under the bilattice approach. The bilattice $\mathcal{FOUR}$ is also considered in [5], though in a somewhat more restricted way, with emphasis entirely on the $\leq_t$ operations. That paper considers the issues of model, continuity, and operational semantics, none of which we take up here.

For positive programs, both $\leq_k$ and $\leq_t$ least and greatest fixed points exist. Some easy results are obtained concerning their interrelationships. We use the following handy notation: v_t and V_t are the least and greatest fixed points of $\Phi_{\mathcal{P}}$ under the $\leq_t$ ordering, and v_k is the least fixed point under the $\leq_k$ ordering. The following inequalities are immediate, by definition of least and greatest:

1. $v_t \leq_t v_k \leq_t V_t$;
2. $v_k \leq_k v_t$;
3. $v_k \leq_k V_t$.

Proposition 7.3 *Let $\mathcal{P}$ be a simple, positive program, and let A be ground atomic. Also assume* **B** *has a conflation operation. $v_k(A)$ is exact iff $v_t(A) = V_t(A)$ iff $v_t(A) = V_t(A) = v_k(A)$.*

Proof By 1), if $v_t(A) = V_t(A)$, then $v_t(A) = V_t(A) = v_k(A)$. It then follows that $v_k(A)$ is exact, by Proposition 7.2.

By 2), $v_k(A) \leq_k v_t(A)$. Then $-v_t(A) \leq_k -v_k(A)$. $v_t(A)$ is exact, by Proposition 7.2, so if $v_k(A)$ is exact too, then $v_t(A) \leq_k v_k(A)$, and hence $v_t(A) = v_k(A)$. Similarly if $v_k(A)$ is exact, $V_t(A) = v_k(A)$.

Corollary 7.4 *For a simple, positive program $\mathcal{P}$, assuming there is a conflation, v_k is exact iff $v_t = V_t$ iff $v_t = V_t = v_k$.*

Finally, for the classical truth values, we can be even more precise.

Corollary 7.5 *For a simple, positive program $\mathcal{P}$, assuming there is a conflation,*

1. *$v_k(A) = true$ iff $v_t(A) = true$;*
2. *$v_k(A) = false$ iff $V_t(A) = false$.*

Proof If $v_k(A) = true$ then $v_k(A)$ is exact, hence $v_k(A) = v_t(A)$. Conversely, if $v_t(A) = true$, by inequality 1) earlier, $true \leq_t v_k(A)$, hence $v_k(A) = true$ since $true$ is the largest member of **B** under the $\leq_t$ ordering. Part 2) is similar.

For the simplest interlaced bilattice, $\mathcal{FOUR}$, the only exact truth values are the classical ones, and the only consistent ones are these and $\bot$, so this Corollary completely characterizes v_k in such a case. In the previous section we observed that the operator associated with a simple, positive program, using the interlaced bilattice $\mathcal{FOUR}$, embodies the behavior of both the classical $T_{\mathcal{P}}$ operator and the three-valued operator. Then the Corollary above specializes to the following result: using $\mathcal{FOUR}$, for a simple, positive program, and for a ground atomic formula A, the least fixed point of the operator in the three-valued semantics assigns A the value *true* iff the least fixed point of $T_{\mathcal{P}}$ assigns A true in the classical two-valued semantics; and the least fixed point of the three-valued operator assigns A $false$ iff the greatest fixed point of $T_{\mathcal{P}}$ assigns A the value $false$. This result first appeared in [7], with a different proof, and with essentially the present proof in [10].

The result from [7] about the interlaced bilattice $\mathcal{FOUR}$ just discussed has a rather far-reaching generalization, with which we conclude. Before stating it, we re-formulate the result connecting the two and the three-valued semantics in more algebraic terms. One reason this re-formulation was not discovered earlier is that the appropriate algebraic operations are not evident until the full interlaced bilattice $\mathcal{FOUR}$ is considered. Specifically, we need the operation $\otimes$.

> In $\mathcal{FOUR}$, for a simple, positive program $\mathcal{P}$, using the notation for least and greatest fixed points given earlier: $v_k = v_t \otimes V_t$.

The justification of this is easy. If $v_k(A) = true$, by Corollary 7.5, $v_t(A) = true$, hence $V_t(A) = true$ since $v_t \leq_t V_t$, and so $(v_t \otimes V_t)(A) = true$. If $v_k(A) = false$, then $(v_t \otimes V_t)(A) = false$ by a similar argument. $v_k(A)$ can not be $\top$ by Proposition 7.1. Finally, if $v_k(A) = \bot$, by Proposition 7.3, $v_t(A) \neq V_t(A)$, though both are exact, and it follows that $(v_t \otimes V_t)(A) = \bot$.

Now, we generalize this in three directions. First, above we only considered the fixed points v_t, V_t and v_k. There are *four* extremal fixed points for $\Phi_{\mathcal{P}}$ when $\mathcal{P}$ is positive. We intend to find relationships between them all. Second, the result above was only for positive, *simple* programs. The proof made use of simplicity in citing results about exactness and consistency. This restriction can be relaxed. Also the existence of a conflation operation was necessary, in order for exactness and consistency to make sense. Finally, the result above was only for the interlaced bilattice $\mathcal{FOUR}$. We extend it to a broad class of bilattices including some, like $\mathcal{SIX}$, with no conflation operation.

An interlaced bilattice has four basic binary operations, $\oplus$, $\otimes$, $\wedge$ and $\vee$, and so there are twelve possible distributive laws:

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$

$$a \wedge (b \oplus c) = (a \wedge b) \oplus (a \wedge c)$$

etc.

Definition 7.2 *We call an interlaced bilattice* distributive *if all twelve distributive laws hold.*

Note We will continue to refer to an interlaced distributive bilattice, but in fact this is somewhat redundant since the distributivity laws imply the interlacing condition. For example, suppose

$a \leq_k b$, where $\leq_k$ is a lattice ordering. Then $a \oplus b = b$ and so $(a \oplus b) \wedge c = b \wedge c$. If we are allowed to distribute, this yields $(a \wedge c) \oplus (b \wedge c) = b \wedge c$, and hence $a \wedge c \leq_k b \wedge c$. The other interlacing conditions are established in a similar way.

Distributive bilattices are fairly common. They include $\mathcal{FOUR}$ and $\mathcal{SIX}$, for instance. This follows from a general result.

Proposition 7.6 *If C and D are distributive lattices then $\mathcal{B}(C, D)$ is a distributive bilattice.*

Proof Straightforward checking.

Next, we remarked above that four, not three extremal fixed points were available for positive programs. We extend the notation used earlier, in the obvious way: if $\mathcal{P}$ is a positive (not necessarily simple) program, and $\Phi_{\mathcal{P}}$ is the associated operator on an interlaced bilattice of interpretations, then v_t and V_t are the least and greatest fixed points of $\Phi_{\mathcal{P}}$ under the $\leq_t$ ordering, and v_k and V_k are the least and greatest fixed points of $\Phi_{\mathcal{P}}$ under the $\leq_k$ ordering. Now, our main result is as follows.

Theorem 7.7 *Let $\mathcal{P}$ be a positive compound program, and suppose the interlaced bilattice* **B** *of truth values is distributive and finite. Then:*

1. $v_k = v_t \otimes V_t$
2. $V_k = v_t \oplus V_t$
3. $v_t = v_k \wedge V_k$
4. $V_t = v_k \vee V_k$

Thus there is a nice, symmetric relationship between the extremal fixed points. The rest of this section is devoted to a proof of the proposition above, and can be omitted if you wish to avoid the technical details.

Lemma 7.8 *Suppose that in a distributive interlaced bilattice, $a_1 \leq_t A_1$ and $a_2 \leq_t A_2$ (or $a_1 \leq_k A_1$ and $a_2 \leq_k A_2$). Then:*

$(a_1 \otimes A_1) \wedge (a_2 \otimes A_2) = (a_1 \wedge a_2) \otimes (A_1 \wedge A_2)$,
$(a_1 \oplus A_1) \wedge (a_2 \oplus A_2) = (a_1 \wedge a_2) \oplus (A_1 \wedge A_2)$,
$(a_1 \wedge A_1) \otimes (a_2 \wedge A_2) = (a_1 \otimes a_2) \wedge (A_1 \otimes A_2)$,
etc.

Proof We show the first equality; the others are similar. Using a distributive law:

$$(a_1 \otimes A_1) \wedge (a_2 \otimes A_2) = (a_1 \wedge a_2) \otimes (a_1 \wedge A_2) \otimes (A_1 \wedge a_2) \otimes (A_1 \wedge A_2) = *$$

If $a_1 \leq_t A_1$ and $a_2 \leq_t A_2$ then using the basic interlacing conditions:

$$* \leq_t (a_1 \wedge a_2) \otimes (A_1 \wedge A_2) \otimes (A_1 \wedge A_2) \otimes (A_1 \wedge A_2) = (a_1 \wedge a_2) \otimes (A_1 \wedge A_2)$$

Similarly

$$* \geq_t (a_1 \wedge a_2) \otimes (a_1 \wedge a_2) \otimes (a_1 \wedge a_2) \otimes (A_1 \wedge A_2) = (a_1 \wedge a_2) \otimes (A_1 \wedge A_2).$$

Lemma 7.9 *Suppose that in a* finite, *distributive interlaced bilattice, $a_i \leq_t A_i$ for each $i \in \mathcal{I}$ (or $a_i \leq_k A_i$ for each $i \in \mathcal{I}$). Then*

$$\bigwedge_{i\in\mathcal{I}} (a_i \otimes A_i) = \bigwedge_{i\in\mathcal{I}} a_i \otimes \bigwedge_{i\in\mathcal{I}} A_i$$

$$\bigwedge_{i\in\mathcal{I}} (a_i \oplus A_i) = \bigwedge_{i\in\mathcal{I}} a_i \oplus \bigwedge_{i\in\mathcal{I}} A_i$$

$$\prod_{i\in\mathcal{I}} (a_i \wedge A_i) = \prod_{i\in\mathcal{I}} a_i \wedge \prod_{i\in\mathcal{I}} A_i$$

etc.

Proof As in the previous lemma; since the interlaced bilattice is assumed to be finite, 'infinitary' operations are really 'finitary' ones.

Proposition 7.10 *Suppose* **B** *is a finite, distributive interlaced bilattice, v and V are interpretations of the language $L(\mathcal{W})$ in* **B**, *and ϕ is a closed, positive formula. If $v \leq_t V$ or $v \leq_k V$ then:*

1. $(v \oplus V)(\phi) = v(\phi) \oplus V(\phi)$,
2. $(v \otimes V)(\phi) = v(\phi) \otimes V(\phi)$,
3. $(v \wedge V)(\phi) = v(\phi) \wedge V(\phi)$,
4. $(v \vee V)(\phi) = v(\phi) \vee V(\phi)$,

Note The operations on the left are the induced ones in the interlaced bilattice of interpretations, which need not be finite. The operations on the right are in the finite interlaced bilattice **B**.

Proof By induction on the complexity of ϕ. If ϕ is ground atomic, the result is immediate from the definitions of the pointwise orderings in the space of interpretations, and does not depend on the inequalities between v and V. We consider one of the induction cases, as a representative example. Suppose ϕ is $(\exists x)\psi(x)$, and 1) is known for formulas simpler than ϕ, in particular for $\psi(d)$ for each d in the work space domain **D**. Then

$$\begin{aligned} (v \oplus V)(\phi) &= (v \oplus V)((\exists x)\psi(x)) \\ &= \bigvee_{d\in\mathbf{D}} (v \oplus V)(\psi(d)) \end{aligned}$$

$$
\begin{aligned}
&= \bigvee_{d\in\mathbf{D}} [v(\psi(d)) \oplus V(\psi(d))] \quad \text{(by induction hypothesis)}\\
&= \bigvee_{d\in\mathbf{D}} v(\psi(d)) \oplus \bigvee_{d\in\mathbf{D}} V(\psi(d)) \quad \text{(by Lemma 7.9)}\\
&= v((\exists x)\psi(x)) \oplus V((\exists x)\psi(x))\\
&= v(\phi) \oplus V(\phi)
\end{aligned}
$$

Finally we come to the proof of Theorem 7.7 itself.

Proof Smallest and biggest fixed points are approached via a transfinite sequence of approximations. We use the following notation to represent this. $(v_t)_0$ is the smallest interpretation in the $\leq_t$ direction. For an arbitrary ordinal α, $(v_t)_{\alpha+1} = \Phi_{\mathcal{P}}((v_t)_\alpha)$. Finally, for a limit ordinal λ, $(v_t)_\lambda = \bigvee_{\alpha<\lambda}(v_t)_\alpha$. As usual, the sequence $(v_t)_\alpha$ is monotone increasing in α ($\alpha < \beta$ implies $(v_t)_\alpha \leq_t (v_t)_\beta$). And for some ordinal ∞, $(v_t)_\infty = v_t$, the least fixed point of $\Phi_{\mathcal{P}}$ in the $\leq_t$ ordering. In fact, from that stage on, things remain fixed; that is, if $\alpha \geq \infty$ then $(v_t)_\alpha = v_t$.

More notation. $(V_t)_0$ is the largest interpretation in the $\leq_t$ ordering. $(V_t)_{\alpha+1} = \Phi_{\mathcal{P}}((V_t)_\alpha)$. And for limit λ, $(V_t)_\lambda = \bigwedge_{\alpha<\lambda}(V_t)_\alpha$. Then $(V_t)_\alpha$ decreases in the $\leq_t$ ordering, and for some ordinal ∞, $(V_t)_\infty = V_t$. Finally we use $(v_k)_\alpha$ and $(V_k)_\alpha$ analogously, but with $\leq_k$, $\sum$ and $\prod$ playing the rules that $\leq_t$, $\bigvee$ and $\bigwedge$ played above.

Now to show item 1) of Theorem 7.7, say, $v_k = v_t \otimes V_t$, it is enough to show by transfinite induction that, for each ordinal α, $(v_k)_\alpha = (v_t)_\alpha \otimes (V_t)_\alpha$. Items 2), 3) and 4) are proved in exactly the same way.

Initial Case. Let A be ground atomic. Because the initial interpretations are smallest and greatest in their respective orderings, $(v_k)_0(A) = \bot$, $(v_t)_0(A) = \mathit{false}$ and $(V_t)_0(A) = \mathit{true}$, so $(v_k)_0(A) = (v_t)_0(A) \otimes (V_t)_0(A)$ by Proposition 3.1. Thus $(v_k)_0 = (v_t)_0 \otimes (V_t)_0$.

Induction Case. Suppose $(v_k)_\alpha = (v_t)_\alpha \otimes (V_t)_\alpha$. Let $A = P(d_1, \ldots, d_n)$ be ground atomic, and suppose P is unreserved. Say $P(x_1, \ldots, x_n) \leftarrow \phi(x_1, \ldots, x_n)$ is the clause of program $\mathcal{P}$ for P. Then:

$$
\begin{aligned}
(v_k)_{\alpha+1}(A) &= (v_k)_{\alpha+1}(P(d_1, \ldots, d_n))\\
&= \Phi_{\mathcal{P}}((v_k)_\alpha)(P(d_1, \ldots, d_n))\\
&= (v_k)_\alpha(\phi(d_1, \ldots, d_n))\\
&= [(v_t)_\alpha \otimes (V_t)_\alpha](\phi(d_1, \ldots, d_n)) \quad \text{(by induction hypothesis)}\\
&= (v_t)_\alpha(\phi(d_1, \ldots, d_n)) \otimes (V_t)_\alpha(\phi(d_1, \ldots, d_n)) \quad \text{(by Proposition 7.10)}\\
&= \Phi_{\mathcal{P}}((v_t)_\alpha(P(d_1, \ldots, d_n)) \otimes \Phi_{\mathcal{P}}((V_t)_\alpha(P(d_1, \ldots, d_n))\\
&= (v_t)_{\alpha+1}(P(d_1, \ldots, d_n)) \otimes (V_t)_{\alpha+1}(P(d_1, \ldots, d_n))\\
&= (v_t)_{\alpha+1}(A) \otimes (V_t)_{\alpha+1}(A).
\end{aligned}
$$

If P is unreserved but there is no program clause for P, or if P is reserved, then $(v_k)_{\alpha+1}(A) = (v_t)_{\alpha+1}(A) \otimes (V_t)_{\alpha+1}(A)$ trivially. Hence $(v_k)_{\alpha+1} = (v_t)_{\alpha+1} \otimes (V_t)_{\alpha+1}$.

Limit Case. Suppose $(v_k)_\alpha = (v_t)_\alpha \otimes (V_t)_\alpha$ for every $\alpha < \lambda$, where λ is a limit ordinal. Let A be ground atomic. Then $(v_k)_\lambda(A) = (\sum_{\alpha<\lambda}(v_k)_\alpha)(A) = \sum_{\alpha<\lambda}(v_k)_\alpha(A)$. Since the interlaced bilattice $\mathbf{B}$ is finite, and $(v_k)_\alpha$ is increasing with α in the $\leq_k$ ordering, there must be an $\alpha_o < \lambda$ so that $\sum_{\alpha<\lambda}(v_k)_\alpha(A) = (v_k)_{\alpha_0}(A)$. Further, for any ordinal β with $\alpha_0 \leq \beta \leq \lambda$ we must have $(v_k)_{\alpha_0}(A) = (v_k)_\beta(A) = (v_k)_\lambda(A)$. Similarly, using the facts that $(v_t)_\alpha$ is increasing and $(V_t)_\alpha$ is decreasing in the $\leq_t$ ordering, there must be ordinals $\alpha_1, \alpha_2 < \lambda$ such that $\alpha_1 \leq \beta \leq \lambda \Rightarrow (v_t)_{\alpha_1}(A) = (v_t)_\beta(A) = (v_t)_\lambda(A) = \bigvee_{\alpha<\lambda}(v_t)_\alpha(A)$ and $\alpha_2 \leq \beta \leq \lambda \Rightarrow (V_t)_{\alpha_2}(A) = (V_t)_\beta(A) = (V_t)_\lambda(A) = \bigwedge_{\alpha<\lambda}(V_t)_\alpha(A)$. Now, let $\gamma = \max\{\alpha_0, \alpha_1, \alpha_2\}$. Then, since $\gamma < \lambda$ we can use the induction hypothesis, and so:

$$\begin{aligned}(v_k)_\lambda(A) &= (v_k)_\gamma(A)\\ &= [(v_t)_\gamma \otimes (V_t)_\gamma](A)\\ &= (v_t)_\gamma(A) \otimes (V_t)_\gamma(A)\\ &= (v_t)_\lambda(A) \otimes (V_t)_\lambda(A).\end{aligned}$$

Hence $(v_k)_\lambda = (v_t)_\lambda \otimes (V_t)_\lambda$.

This concludes the proof.

8 Conclusion

Interlaced bilattices provide a simple, elegant setting for the consideration of logic programming extensions allowing for incomplete or contradictory answers. On a theoretical level a considerable unification of several ideas is achieved. Though in the abstract all interlaced bilattices are quite natural, in practice not all are appropriate for computer implementation. A version based on $\mathcal{FOUR}$ is a practical goal, taking the Prolog implementation of the classical logic programming paradigm as a model. An implementation along these lines may be found in [11]. Also logic programming using $\mathcal{B}([0,1],[0,1])$ should be possible as an extension of the version proposed in [20]. In other cases desirability and practicality are issues that remain to be decided. We hope interlaced bilattices will provide the same kind of motivation, for future logic programming language development, that the classical semantics has supplied heretofore.

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] H. A. Blair, A. L. Brown and V. S. Subrahmanian, A Logic programming semantics scheme, Technical Report LPRG-TR-88-8, Syracuse University (1988).

[5] H. A. Blair, V. S. Subrahmanian, Paraconsistent logic programming, *Proc. of the 7th Conference on Foundations of Software Technology and Theoretical Computer Science, Springer Lecture Notes in Computer Science*, vol 287.

[6] 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).

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

[8] M. C. Fitting, Partial models and logic programming, *Theoretical Computer Science*, pp 229–255, vol 48 (1986).

[9] M. C. Fitting, *Computability Theory, Semantics, and Logic Programming*, Oxford University Press (1987).

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

[11] M. C. Fitting, Negation as refutation, to appear in *LICS 1989*.

[12] 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).

[13] M. C. Fitting and M. Ben-Jacob, Stratified, weak stratified and three-valued semantics, to appear *Fundamenta Informatica*.

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

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

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

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

[18] K. Kunen, Signed data dependencies in logic programs, forthcoming in *Journal of Logic Programming*.

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

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

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

[22] A. Visser, Four valued semantics and the liar, *Journal of Philosophical Logic*, pp 181–212, vol 13 (1984).