Annotated Revision Specification Programs

Melvin Fitting
mlflc@cunyvm.cuny.edu

Dept. Mathematics and Computer Science
Lehman College (CUNY), Bronx, NY 10468

Abstract. Marek and Truszczyński have introduced an interesting mechanism for specifying revisions of knowledge bases by means of logic programs. Here we extend their idea to allow for confidence factors, multiple experts, and so on. The appropriate programming mechanism turns out to be *annotated logic programs* and the appropriate semantic tool, *bilattices*. This may be the first example of a setting in which both notions arise naturally, and complement each other. We also show that several of the results of Marek and Truszczyński turn out to be essentially algebraic, once the proper setting has been formulated.

1 Introduction

In [8] Marek and Truszczyński provided an interesting formalism for specifying revisions in databases, knowledge bases, and belief sets. The specifications could be quite complex, containing conditionals (e.g., if this and that is present, so-and-so must be absent). And they presented an ingenious test for determining whether a candidate for an update is one in fact. This test was suggested by the definition of stable model and, in a sense, properly extends the notion.

We extend their work in the following ways. First, we generalize their formalism to systems that allow confidence factors, evidence, multiple experts, and other similar notions. The (fairly obvious) tool for this is an *annotated* version ([7]) of a revision program. To give semantic meaning to such annotated programs, *bilattices* are a natural tool. (This seems to be the first joint exploitation of annotations and bilattices together.) We show that several of the results of Marek and Truszczyński extend naturally to this broader setting. Finally, we show that simple algebraic bilattice calculations yield interesting results about the update mechanism. The ultimate goal (the present paper representing an early stage of the work) is to extract away all the details of the update formalism to reveal the abstract structure underneath. Work on this continues.

2 The Original Version

The following is a brief sketch of revision specification programs as presented in [8]. First, we have an underlying set U whose elements are not analysed further — they are called *atoms*. These can be thought of as things that can appear in a knowledge base. Next, there are two special unary operators, **in** and **out**, which apply to atoms. The idea is, $\mathbf{in}(a)$ says the atom a is (or ought to be) in a knowledge base, while

out(a) says a is (or ought to be) out. We will call **in**(a) and **out**(a) *revision atoms*. A *revision rule* is an expression of one of the following forms:

$$\mathbf{in}(p) \leftarrow \mathbf{in}(q_1), \ldots, \mathbf{in}(q_m), \mathbf{out}(s_1), \mathbf{out}(s_n)$$
$$\mathbf{out}(p) \leftarrow \mathbf{in}(q_1), \ldots, \mathbf{in}(q_m), \mathbf{out}(s_1), \mathbf{out}(s_n)$$

The first kind are called *in-rules* and the second are *out-rules*. Bodies are allowed to be empty. Finally, a *revision program* is a collection of revision rules. (Note that, as defined here, a revision program is propositional. A formulation allowing variables is certainly possible, but let's keep things as simple as possible for the time being.) Now, the idea is to determine when a revised knowledge base B_R (a set of atoms) is a justified revision of an initial knowledge base B_I (another set of atoms) according to conditions specifiec by a revision program $\mathcal{P}$.

In order to do this, Marek and Truszczyński introduce a fairly elaborate process, inspired by the notion of stable model. First, some useful terminology. If B is a knowledge base, **in**(a) is *satisfied in* B if $a \in B$. Similarly **out**(a) is satisfied in B if $a \notin B$. Now let $\mathcal{P}$ be a revision program, and B_I and B_R be two knowledge bases. The process begins with the notion of the *reduct* of $\mathcal{P}$ with respect to (B_I, B_R), which is defined as follows.

1. Remove from $\mathcal{P}$ every rule whose body contains a revision atom that is not satisfied in B_R.
2. From the body of each remaining rule, delete any revision atom that is satisfied in B_I.

The notation Marek and Truszczyński use for this revised program is $\mathcal{P}_{B_R}|B_I$.

The revised program $\mathcal{P}_{B_R}|B_I$ is a logic program in the usual sense (without negations or function symbols) and so has a well-behaved minimal model. This minimal model is called the *necessary change basis*, and is denoted by $NCB(\mathcal{P}_{B_R}|B_I)$. It, in turn, determines the *necessary change* for $\mathcal{P}_{B_R}|B_I$ — this is the pair (I, O), where $I = \{a \mid \mathbf{in}(a) \in NCB(\mathcal{P}_{B_R}|B_I)\}$ and $O = \{a \mid \mathbf{out}(a) \in NCB(\mathcal{P}_{B_R}|B_I)\}$. If $I \cap O = \emptyset$ the revised program is called *coherent*.

Finally, suppose $\mathcal{P}_{B_R}|B_I$ is coherent, and (I, O) is the necessary change for $\mathcal{P}_{B_R}|B_I$. Then $(B_I \cup I)\backslash O$ is well-defined (add the members of I to B_I and remove the members of O). If it turns out that $B_R = (B_I \cup I)\backslash O$ then B_R is a *$\mathcal{P}$-justified revision of B_I*.

The intuitions about knowledge base revision that the notions above are intended to capture are these. First, a revised knowledge base should satisfy all the constraints imposed by the revision program. And second, each insertion or deletion performed in order to convert B_I into B_R must be justified by some rule in the revision program. See [8] for a discussion of these ideas, with several motivating examples. In this paper we simply take for granted that the intentions are correctly captured by the Marek, Truszczyński process, and note that their intuitions carry over to the setting of this paper. We concentrate on technical details.

3 Annotated Logic Programs

Generalized Annotated Programs are fully treated in [7], to which we refer you for details. In the interests of simplicity, here we use only a much stripped-down version,

with no annotation variables, no function symbols, and more than minimal assumptions about the space of annotations. Given these simplifications, what we need of annotated logic programming can be easily sketched.

Let $\mathcal{T}$ be a complete distributive lattice, of truth values. These will be used as annotations in what follows. In addition we will assume $\mathcal{T}$ has a *de Morgan complement*, a one-one mapping that is order inverting, of period two. We denote the de Morgan complement of a by $\overline{a}$. It follows easily that de Morgan complementation satisfies the usual de Morgan laws. (In [7] annotations are only required to be an upper semilattice.)

Examples The following are typical and should be kept in mind for what follows.

1. $\mathcal{T} = \{\mathit{false}, \mathit{true}\}$, with $\mathit{false} < \mathit{true}$. The de Morgan complement is the usual negation operation. This is the grand-daddy of all examples, of course.
2. $\mathcal{T}$ is the collection of all subsets of some set S, with $\subseteq$ as the ordering relation. The de Morgan complement is set-theoretic complement. Think of the members of S as experts, and a member A of $\mathcal{T}$ as a set of experts who assert the truth of some proposed fact.
3. $\mathcal{T} = [0, 1]$, with the usual ordering. The de Morgan complement of a is $1 - a$. Think of a member of $\mathcal{T}$ as a degree of confidence in some proposition.
4. $\mathcal{T}$ is the set of all functions from some set S to the unit interval. The ordering is the pointwise one: $f \leq g$ provided $f(a) \leqq g(a)$ for all $a \in S$. The de Morgan complement of f is the function given by $\overline{f}(a) = 1 - f(a)$. This combines the two previous examples. Think of $f \in \mathcal{T}$ as assigning to each member of a set of experts a confidence factor.

An *annotated revision atom* is either $(\mathbf{in}(a) : \alpha)$ or $(\mathbf{out}(a) : \alpha)$ where a is an atom and α is an annotation — that is, a member of $\mathcal{T}$. Think of $(\mathbf{in}(a) : \alpha)$ as asserting that there is at least α reason to assume that a is (or should be) in the knowledge base. Similarly for $(\mathbf{out}(a) : \alpha)$. For instance, using the first example above, $(\mathbf{in}(a) : \mathit{true})$ simply says a is in, while $(\mathbf{in}(a) : \mathit{false})$ gives no information. Using the second example above, $(\mathbf{in}(a) : A)$, where A is a set of experts, can be thought of as saying the members of A assert that a should be in. Continuing this example, $(\mathbf{in}(a) : \emptyset)$ gives no information. There are similar readings that can be supplied for the other examples.

Now, an *annotated revision rule* is an expression of the form $p \leftarrow q_1, \ldots, q_n$ where p, q_1, …, q_n are annotated revision atoms, and an *annotated revision program* is a set of annotated revision rules.

Example A Suppose there are two independent experts, or sources of information, p and q. Let $\mathcal{T}$ be the collection of all possible sets of experts: $\{\emptyset, \{p\}, \{q\}, \{p, q\}\}$, ordered by inclusion. This is our space of annotations, with complementation as the de Morgan complement. Also, assume there are three atoms a, b and c that are candidates for inclusion in a knowledge base. Now, here is an annotated revision program, which we will continue to discuss throughout the paper.

$$(\mathbf{in}(a) : \{p\}) \leftarrow (\mathbf{out}(b) : \{p\})$$
$$(\mathbf{in}(b) : \{p\}) \leftarrow (\mathbf{out}(a) : \{p\})$$
$$(\mathbf{out}(a) : \{q\}) \leftarrow (\mathbf{in}(a) : \{p\})$$
$$(\mathbf{in}(c) : \{p, q\}) \leftarrow (\mathbf{in}(a) : \{p\})$$
$$(\mathbf{out}(c) : \{q\}) \leftarrow (\mathbf{out}(a) : \{q\})$$

Since there are no variables or function symbols, either in atoms or in annotations, a fixpoint semantics is easy to describe. First, a *$\mathcal{T}$-valuation* is a mapping v from revision atoms to $\mathcal{T}$. $\mathcal{T}$-valuations are given the pointwise ordering, which makes the space of $\mathcal{T}$-valuations into a complete lattice itself. The $\mathcal{T}$-valuation v *satisfies* the annotated revision atom $(\mathbf{in}(a) : \alpha)$ provided $v(\mathbf{in}(a)) \geq \alpha$ — v satisfies $(\mathbf{out}(a) : \alpha)$ if $v(\mathbf{out}(a)) \geq \alpha$. We say v satisfies a list of annotated revision atoms if it satisfies each member of the list. As "intermediate" notation, let $t_{\mathcal{P}}(v)$ be the set of all annotated revision atoms that occur as the head of a clause in $\mathcal{P}$ whose body is satisfied by v. Finally, let $T_{\mathcal{P}}$ be the mapping from $\mathcal{T}$-valuations to $\mathcal{T}$-valuations given by:

$$T_{\mathcal{P}}(v)(A) = \bigvee\{\alpha \mid (A : \alpha) \in t_{\mathcal{P}}(v)\}.$$

It is straightforward to check that $T_{\mathcal{P}}$ is monotonic on the space of $\mathcal{T}$-valuations, hence has a smallest fixed point. This is the intended meaning of the program. Since negations are not present, the intended meaning is quite unproblematic, and appropriate computational mechanisms can be specified (assuming the lattice $\mathcal{T}$ itself has computble operations).

If we use the first example above, identify $(\mathbf{in}(a) : \mathit{true})$ with $\mathbf{in}(a)$, ignore $(\mathbf{in}(a) : \mathit{false})$, and treat $\mathbf{out}(a)$ similarly, revision programming is easily seen to be a special case of annotated revision programming.

4 Bilattices

Atoms can be in a knowledge base, or out of it, and each with a degree of confidence, an annotation. A natural semantic tool for representing this situation is a *bilattice*, as we will see shortly. We begin with a brief abstract presentation, then provide a concrete representation which should make their utility in this context clear. Bilattices were introduced in [6], and have turned out to be a useful tool for investigating logic programming semantics [1, 2], and stable model semantics in particular [3, 5]. We refer you to these papers for a fuller treatment.

Definition 1. A *pre-bilattice* is a structure $\langle \mathcal{B}, \leq_t, \leq_k \rangle$ where $\mathcal{B}$ is a non-empty set and $\leq_t$ and $\leq_k$ are each partial orderings giving $\mathcal{B}$ the structure of a lattice with a top and a bottom. We call $\mathcal{B}$ *complete* if each of the two lattices is complete in the usual sense that all meets and joins exist.

A pre-bilattice has a *negation* if there is a mapping $\neg$ from $\mathcal{B}$ to itself that is an involution, reverses the $\leq_t$ ordering, and preserves the $\leq_k$ ordering. Likewise it has a *conflation* if there is a mapping $-$ that is an involution, reverses the $\leq_k$ ordering, and preserves the $\leq_t$ ordering. If both a negation and a conflation exist, we generally will require that they commute with each other.

Definition 2. In a pre-bilattice $\langle \mathcal{B}, \leq_t, \leq_k \rangle$, meet and join under $\leq_t$ are denoted $\wedge$ and $\vee$, and meet and join under $\leq_k$ are denoted $\otimes$ and $\oplus$. Top and bottom under $\leq_t$ are denoted *true* and *false*, and top and bottom under $\leq_k$ are denoted $\top$ and $\bot$. If the pre-bilattice is complete, infinitary meet and join under $\leq_t$ are denoted $\bigwedge$ and $\bigvee$, and infinitary meet and join under $\leq_k$ are denoted $\prod$ and $\sum$.

Definition 3. A *distributive bilattice* is a pre-bilattice $\langle \mathcal{B}, \leq_t, \leq_k \rangle$ in which all 12 distributive laws connecting $\wedge$, $\vee$, $\otimes$ and $\oplus$ hold. An *infinitely distributive bilattice* is a complete pre-bilattice in which all infinitary, as well as all finitary, distributive laws hold.

It is an easy consequence that in a distributive bilattice, each of the lattice operations, $\wedge$, $\vee$, $\otimes$, $\oplus$, is monotone with respect to both orderings. These are generally called the *interlacing conditions*. If the pre-bilattice is infinitely distributive, each of the infinitary meet and join operations is monotone with respect to both orderings — the *infinitary interlacing conditions*.

Our interests here are confined to complete, infinitely distributive bilattices with negation and conflation that commute. There is a standard way of constructing such structures, which we now sketch. Let $\mathcal{T}$ be a complete distributive lattice with a de Morgen complement, such as could serve as a space of annotations above. We use it to construct a bilattice, which we denote $\mathcal{T} \odot \mathcal{T}$, as follows. The domain is $\mathcal{T} \times \mathcal{T}$. (Think of a member of the domain $\langle \alpha, \beta \rangle$ as saying, there is α evidence that some atom is in a knowledge base, and β evidence that it is out.) The two orderings have the following characterization. $\langle \alpha_1, \beta_1 \rangle \leq_k \langle \alpha_2, \beta_2 \rangle$ if $\alpha_1 \leq \alpha_2$ and $\beta_1 \leq \beta_2$. (Intuitively, knowledge goes up if all evidence, both for and against, increases.) $\langle \alpha_1, \beta_1 \rangle \leq_t \langle \alpha_2, \beta_2 \rangle$ if $\alpha_1 \leq \alpha_2$ and $\beta_1 \geq \beta_2$. (Intuitively, degree of truth goes up if evidence for increases and evidence against decreases.) It is not hard to check that this gives $\mathcal{T} \times \mathcal{T}$ the structure of a complete, infinitely distributive bilattice. Next, negation simply switches around the roles of for and against: $\neg\langle \alpha, \beta \rangle = \langle \beta, \alpha \rangle$. The intuition here is quite clear. Finally, conflation is a little more complicated, and involves the de Morgan complementation operation of $\mathcal{T}$. $-\langle \alpha, \beta \rangle = \langle \overline{\beta}, \overline{\alpha} \rangle$. (Intuitively, the conflation of a member b of $\mathcal{T} \times \mathcal{T}$ counts as evidence for inclusion in a knowledge base whatever b did not count as evidence against, and similarly the other way around.) Again it is easy to check that this does give a conflation that commutes with the negation operation.

We have sketched a method of constructing bilattices $\mathcal{T} \odot \mathcal{T}$ from lattices $\mathcal{T}$. The method is entirely general, in the sense that every complete, infinitely distributive bilattice with a negation and a conflation that commute is isomorphic to $\mathcal{T} \odot \mathcal{T}$ for some complete lattice $\mathcal{T}$ with a de Morgan complement. A proof of this, generalizing a representation theorem of Ginsberg, can be found in [1, 4].

5 Annotated Revision Programs and Bilattices

In section 3, $\mathcal{T}$-valuations were defined. It will be more convenient to work with valuations in a bilattice, and connections are easy to make. First, by a $(\mathcal{T} \odot \mathcal{T})$-*valuation* we mean a mapping from *atoms* (not revision atoms) to members of the bilattice $\mathcal{T} \odot \mathcal{T}$. If the atom a maps to $\langle \alpha, \beta \rangle$ under some $(\mathcal{T} \odot \mathcal{T})$-valuation, think

of this as saying there is α reason to have a in a knowledge base and β reason to have a out. $(\mathcal{T} \odot \mathcal{T})$-valuations are given two orderings, denoted $\leq_k$ and $\leq_t$, in the obvious pointwise way. Likewise the operations of negation, $\neg$, and conflation, $-$, lift pointwise to the space of $(\mathcal{T} \odot \mathcal{T})$-valuations. It is easy to check that the space of $(\mathcal{T} \odot \mathcal{T})$-valuations is, again, a complete infinitely distributive bilattice with negation and conflation that commute. As it happens, in this paper the $\leq_k$ ordering will play the primary role, with almost no role given to $\leq_t$.

There is an obvious correspondence between $\mathcal{T}$-valuations and $(\mathcal{T} \odot \mathcal{T})$-valuations — we denote this correspondence by θ. Suppose v is a $\mathcal{T}$-valuation, mapping revision atoms to members of $\mathcal{T}$. Associate with this the $(\mathcal{T} \odot \mathcal{T})$-valuation $\theta(v)$ defined by: $\theta(v)(a) = \langle \alpha, \beta \rangle$ where $v(\mathbf{in}(a)) = \alpha$ and $v(\mathbf{out}(a)) = \beta$. θ is easily seen to be $1-1$ and onto, and so has an inverse, characterized by: if w is a $(\mathcal{T} \odot \mathcal{T})$-valuation, $\theta^{-1}(w)$ is the $\mathcal{T}$-valuation given by: $\theta^{-1}(w)(\mathbf{in}(a)) = \alpha$ and $\theta^{-1}(w)(\mathbf{out}(a)) = \beta$ provided $w(a) = \langle \alpha, \beta \rangle$.

The mapping θ is order-preserving, in the following sense. Suppose v and w are $\mathcal{T}$-valuations, and $v \leq w$ in the ordering of $\mathcal{T}$. Then $\theta(v) \leq_k \theta(w)$. This is easy to see, as is the fact that θ^{-1} is also order-preserving in the other direction.

Let $\mathcal{P}$ be an annotated revision program. In section 3 we associated with it a mapping $T_{\mathcal{P}}$ on the space of $\mathcal{T}$-valuations. This induces a bilattice mapping which we denote $T^b_{\mathcal{P}}$ on the space of $(\mathcal{T} \odot \mathcal{T})$-valuations, in a direct way:

$$T^b_{\mathcal{P}}(v) = (\theta T_{\mathcal{P}} \theta^{-1})(v).$$

We noted earlier that $T_{\mathcal{P}}$ is monotonic in the space of $\mathcal{T}$-valuations. It follows from this, and the order-preserving properties of θ and θ^{-1}, that $T^b_{\mathcal{P}}$ is monotonic in the space of $(\mathcal{T} \odot \mathcal{T})$-valuations, with respect to the $\leq_k$ ordering. From now on we generally confine our work to the bilattice $\mathcal{T} \odot \mathcal{T}$, and use the $T^b_{\mathcal{P}}$ mapping, pushing $\mathcal{T}$ and $T_{\mathcal{P}}$ into the background.

6 Program Transformation

Now we carry over directly to the present setting the program transformation of Marek and Truszczyński. If v is a $(\mathcal{T} \odot \mathcal{T})$-valuation, we say v *satisfies* an annotated revision atom if the $\mathcal{T}$-valuation $\theta^{-1}(v)$ satisfies it, as defined in section 3.

Definition 4. Let $\mathcal{P}$ be an annotated revision program and let B_I and B_R be $(\mathcal{T} \odot \mathcal{T})$-valuations. We define the reduct of $\mathcal{P}$ with respect to (B_I, B_R) as follows.

1. First, remove from $\mathcal{P}$ every rule whose body contains an annotated revision atom that is not satisfied in B_R.
2. Second, from the body of each remaining rule delete any annotated revision atom that is satisfied in B_I.

The resulting program is denoted $\mathcal{P}_{B_R}|B_I$.

Now we use the notion of program reduct to define an operator on bilattices, as follows.

Definition 5. Let B_I, B_R, and v be three $(\mathcal{T} \odot \mathcal{T})$-valuations, and let $\mathcal{P}$ be an annotated revision program. The mapping $\mathfrak{R}_{\mathcal{P}}$ is given by the following.

$$\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v) = T^b_{\mathcal{P}_{B_R}|B_I}(v).$$

The T notation is somewhat hair-raising, which is partly why we have introduced the $\mathfrak{R}$ (for "revision") notation. The idea, nonetheless, is straightforward. To compute $\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$, begin with the annotated revision program $\mathcal{P}$, carry out step 1 of the reduction process using B_R; next carry out step 2 using B_I. This yields another annotated revision program $\mathcal{P}_{B_R}|B_I$. Apply the "single-step" T operator for this program, using v as input, after translating v from the bilattice $\mathcal{T} \odot \mathcal{T}$ to the underlying truth-value space $\mathcal{T}$, then translating the result back to the bilattice setting. The outcome is the value of $\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$.

As an operator on $\mathcal{T} \odot \mathcal{T}$, $\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$ has several nice properties. First, it is obviously monotonic in v, in the $\leq_k$ ordering (since T^b operators are). Next, it is also monotonic in B_I in the $\leq_k$ ordering (because if B_I is increased, more annotated revision atoms will be satisfied, so more parts of clause bodies will be deleted in step 2 of the reduction process, yielding a program whose clause bodies are more easily satisfied). Finally, it is even monotonic in B_R in the $\leq_k$ ordering (because if B_R is increased, fewer annotated revision atoms will be unsatisfied, so fewer clauses will be deleted in step 1 of the reduction process, again yielding a program whose clause bodies are more easily satisfied, since there are more of them).

Example A continued (Example A began in section 3.) Let B_I be given by:

$$\begin{aligned} B_I(a) &= \langle \{q\}, \{p, q\} \rangle \\ B_I(b) &= \langle \emptyset, \{p, q\} \rangle \\ B_I(c) &= \langle \emptyset, \{q\} \rangle \end{aligned}$$

And let B_R be given by:

$$\begin{aligned} B_R(a) &= \langle \{p\}, \{q\} \rangle \\ B_R(b) &= \langle \emptyset, \{p, q\} \rangle \\ B_R(c) &= \langle \{p, q\}, \{q\} \rangle \end{aligned}$$

Now, $(\mathbf{out}(a) : \{p\})$ is not satisfied by B_R, but all other annotated revision atoms in clause bodies of $\mathcal{P}$ are, so only the second clause of $\mathcal{P}$ is deleted in step 1 above. Next, both $(\mathbf{out}(b) : \{p\})$ and $(\mathbf{out}(a) : \{q\})$ are satisfied by B_I so these are deleted from clause bodies in step 2. The resulting program, $\mathcal{P}_{B_R}|B_I$, is:

$$\begin{aligned} (\mathbf{in}(a) : \{p\}) &\leftarrow \\ (\mathbf{out}(a) : \{q\}) &\leftarrow (\mathbf{in}(a) : \{p\}) \\ (\mathbf{in}(c) : \{p, q\}) &\leftarrow (\mathbf{in}(a) : \{p\}) \\ (\mathbf{out}(c) : \{q\}) &\leftarrow \end{aligned}$$

Notice, in this example, that according to B_I, expert q is acting inconsistently, asserting that a should both be present and absent. In the original Marek and Truszczyński setting such inconsistencies were not allowed — coherency conditions were explicitly imposed. We find it more natural to allow inconsistencies, and simply record their presence. This is especially useful in a setting like that above, where q is being inconsistent with regard to a, but p is not.

7 How To Make Change

Now that the notion of reduced program has been introduced in our setting, we can use it to compute the necessary change basis, and consequently the necessary change. Then we have the problem of how to carry out this change. But first things first.

Thought of as an annotated logic program, a reduced program $\mathcal{P}_{B_R}|B_I$ has a minimal model — a $\mathcal{T}$-valuation. Equivalently, because of the correspondence between $\mathcal{T}$-valuations and $(\mathcal{T} \odot \mathcal{T})$-valuations, we can take the least fixed point, in the $\leq_k$ ordering, of the mapping:

$$(\lambda v)\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$$

We call the least fixed point of this mapping the *necessary change* for $\mathcal{P}_{B_R}|B_I$.

Example A continued In the previous section the revised program $\mathcal{P}_{B_R}|B_I$ was given. It is easy to see that the minimal model for this is the set consisting of: $(\mathbf{in}(a) : \{p\})$, $(\mathbf{out}(a) : \{q\})$, $(\mathbf{in}(c) : \{p, q\})$, and $(\mathbf{out}(c) : \{q\})$. Corresponding to this is the $(\mathcal{T} \odot \mathcal{T})$-valuation C, the *necessary change* for $\mathcal{P}_{B_R}|B_I$, as follows:

$$\begin{aligned} C(a) &= \langle \{p\}, \{q\} \rangle \\ C(b) &= \langle \emptyset, \emptyset \rangle \\ C(c) &= \langle \{p, q\}, \{q\} \rangle \end{aligned}$$

Now that we have said what necessary change amounts to in the present context, we must say how to effectuate it. For this, we propose the following.

Definition 6. The result of applying change C to an initial knowledge base B_I is:

$$(B_I \otimes -C) \oplus C.$$

In this, $\otimes$ and $\oplus$ are the bilattice meet and join with respect to the $\leq_k$ ordering, and $-$ is the conflation operation. To get a feeling for how this works, suppose $\mathcal{T}$ is the lattice of subsets of the three-expert set $\{p, q, r\}$, $B_I = \langle \{p, q\}, \{r\} \rangle$, and $C(a) = \langle \emptyset, \{p\} \rangle$. Essentially $C(a)$ amounts to: put p in the "against" column, as far as a is concerned, and don't change anybody else. For this choice of $\mathcal{T} \odot \mathcal{T}$, conflation is given by $-\langle X, Y \rangle = \langle \overline{Y}, \overline{X} \rangle$, where $\overline{X}$ denotes complement. Then $-C(a) = \langle \{q, r\}, \{p, q, r\} \rangle$. Also, for $\mathcal{T} \odot \mathcal{T}$, $\langle X_1, X_2 \rangle \otimes \langle Y_1, Y_2 \rangle$ is $\langle X_1 \cap Y_1, X_2 \cap Y_2 \rangle$ and $\langle X_1, X_2 \rangle \oplus \langle Y_1, Y_2 \rangle$ is $\langle X_1 \cup Y_1, X_2 \cup Y_2 \rangle$. Then $(B_I \otimes -C)(a) = \langle \{q\}, \{r\} \rangle$. Notice that this amounts to the removal of p from the "for" side. Next, $((B_I \otimes -C) \oplus C)(a) = \langle \{q\}, \{p, r\} \rangle$, and indeed p has been withdrawn as "for" and added as "against." We leave it to you to try out examples involving inconsistent or incomplete information. We also note that if we restrict the setting to the original one of Marek and Truszczyński, the present definition is equivalent to theirs. (Recall, their setting is the simplest non-trivial bilattice, the four-element one, with valuations never taking on either $\bot$ or $\top$ as values.)

Example A continued We have computed the necessary change for Example A. Now, $-C$ is as follows:

$$-C(a) = \langle \{p\}, \{q\} \rangle$$
$$-C(b) = \langle \{p, q\}, \{p, q\} \rangle$$
$$-C(c) = \langle \{p\}, \emptyset \rangle$$

Continuing,

$$((B_I \otimes -C) \oplus C)(a) = \langle \{p\}, \{q\} \rangle$$
$$((B_I \otimes -C) \oplus C)(b) = \langle \emptyset, \{p, q\} \rangle$$
$$((B_I \otimes -C) \oplus C)(c) = \langle \{p, q\}, \{q\} \rangle$$

Definition 7. Extending the terminology of Marek and Truszczyński, we say B_R is a *$\mathcal{P}$-justified revision of* B_I if $B_R = (B_I \otimes -C) \oplus C$.

Notice that in Example A, B_R is in fact a $\mathcal{P}$-justified revision of B_I. We leave it to you to verify that the following is also a $\mathcal{P}$-justified revision of B_I:

$$B_R(a) = \langle \{q\}, \{p, q\} \rangle$$
$$B_R(b) = \langle \{p\}, \{q\} \rangle$$
$$B_R(c) = \langle \emptyset, \{q\} \rangle$$

8 Elementary Results

The notion of applying a necessary change to a knowledge base, introduced above in a general context, has several nice features, and some of these can be established by essentially algebraic methods.

Why only two values? We begin by showing why the full structure of a bilattice never arose in the original treatment of Marek and Truszczyński. As we noted earlier, their setting can be thought of as the four-element bilattice, but restricted to the portion consisting of only *false* and *true*. Ruling out $\top$ is done explicitly: they say a change is not well-defined if it is not coherent. Ruling out $\bot$ is done implicitly: all their models are considered totally defined. Now, how can such restrictions be extended to more general bilattices?

We showed in [2] that the conflation operator is the key. Suppose we call a member A of $\mathcal{T} \odot \mathcal{T}$ *exact* if $A = -A$. In the four-element bilattice, the exact members are simply *false* and *true*. In the setting of Example A, the exact values are those $\langle A, B \rangle$ for which B is exactly the complement of A (with respect to $\{p, q\}$). In general the exact members of $\mathcal{T} \odot \mathcal{T}$ always possess many important properties of the classical truth values (such as closure under $\wedge$, $\vee$, $\neg$, $\bigwedge$, and $\bigvee$) and can be considered a reasonable generalization. Further, suppose we call A *consistent* if $A \leq_k -A$. In the four-element setting, the consistent members are *false*, *true*, and $\bot$, and if the $\leq_t$ operations are restricted to these values, Kleene's strong three-valued logic results. In the setting of Example A, the consistent values are those $\langle A, B \rangle$ for which A and B do not overlap. In general, the consistent members of $\mathcal{T} \odot \mathcal{T}$ always constitute a complete semi-lattice with respect to $\leq_k$ and are closed under the operations of $\leq_t$, and thus naturally generalize Kleene's logic. With all this in mind, we have the following simple result.

Theorem 8. *Working in $\mathcal{T} \odot \mathcal{T}$, suppose an initial knowledge base B_I is exact and the necessary change C is consistent. Then the result of applying the change, $(B_I \otimes -C) \oplus C$, is exact.*

Proof. Let $D = (B_I \otimes -C) \oplus C$. We show $-D = D$, assuming that $-B_I = B_I$ and $C \leq_k -C$. Note, from the latter it follows that $C \otimes -C = C$. Now, using distributive laws and the de Morgan properties of conflation, we have the following calculation.

$$\begin{aligned} -D &= -((B_I \otimes -C) \oplus C) \\ &= -((-B_I \otimes -C) \oplus C) \\ &= (B_I \oplus C) \otimes -C \\ &= (B_I \otimes -C) \oplus (C \otimes -C) \\ &= (B_I \otimes -C) \oplus C \\ &= D \end{aligned}$$

Undoing changes. We wish to investigate under what circumstances changes can be undone. More specifically we will show that, at least sometimes, if C changes A to B, $\neg C$ will change B back to A. Now we should not expect this under all circumstances. For instance, if C says to add p to a knowledge base, and p is already in A, applying C to A yields A again. But $\neg C$ will tell us to remove p, and this certainly does not leave A unchanged. The problem here is that C is telling us to do something unnecessary.

Consider an example using the annotation space $\mathcal{T}$ of Example A. Suppose $C(a) = \langle \{p\}, \{q\} \rangle$ and $A(a) = \langle \alpha, \beta \rangle$. In order for C to be telling us to do something that actually needs doing, we must have $p \notin \alpha$ and $q \notin \beta$, or equivalently, $p \in \overline{\alpha}$ and $q \in \overline{\beta}$. But $-\neg\langle \alpha, \beta \rangle = \langle \overline{\alpha}, \overline{\beta} \rangle$, so we can state our requirements quite simply: we want $\langle \{p\}, \{q\} \rangle \leq_k -\neg\langle \alpha, \beta \rangle$. This leads us to the following notion.

Definition 9. We say C is an *essential change* with respect to A if $C \leq_k -\neg A$.

Theorem 10. *Suppose that $\mathcal{T}$ is not just a complete lattice with a de Morgan complement, but that it is also a Boolean algebra. Working in the bilattice $\mathcal{T} \odot \mathcal{T}$, suppose C is an essential change with respect to A, and A is classical. If C changes A into B, then $\neg C$ changes B into A. More precisely, if $B = (A \otimes -C) \oplus C$, then $A = (B \otimes -\neg C) \oplus \neg C$.*

Proof. If we assume the complement operation of $\mathcal{T}$ satisfies the Boolean algebra law, $x \wedge \overline{x} = \bot$, it follows easily that in $\mathcal{T} \odot \mathcal{T}$ we have (1) $X \otimes -\neg X = \bot$. Applying conflation to both sides of this, we also have (2) $-X \oplus \neg X = \top$. Since C is an essential change with respect to A, $C \leq_k -\neg A$, or equivalently, (3) $A \leq_k -\neg C$, or

$\neg C \leq_k -A$. Finally A is classical, so (4) $-A = A$. Now, the argument is as follows.

$$\begin{aligned}
(A \otimes -C) \oplus C &= B && \text{Assumption}\\
[(A \otimes -C) \oplus C] \otimes -\neg C &= B \otimes -\neg C &&\\
(A \otimes -C \otimes -\neg C) \oplus (C \otimes -\neg C) &= B \otimes -\neg C && \text{Distributive Law}\\
A \otimes -C \otimes -\neg C &= B \otimes -\neg C && \text{By (1)}\\
A \otimes -C &= B \otimes -\neg C && \text{By (3)}\\
(A \otimes -C) \oplus \neg C &= (B \otimes -\neg C) \oplus \neg C &&\\
(A \oplus \neg C) \otimes (-C \oplus \neg C) &= (B \otimes -\neg C) \oplus \neg C && \text{Distributive Law}\\
A \oplus \neg C &= (B \otimes -\neg C) \oplus \neg C && \text{By (2)}\\
-A \oplus \neg C &= (B \otimes -\neg C) \oplus \neg C && \text{By (4)}\\
-A &= (B \otimes -\neg C) \oplus \neg C && \text{By (3)}\\
A &= (B \otimes -\neg C) \oplus \neg C && \text{By (4)}
\end{aligned}$$

Notice, incidentally, that the multiple expert examples do satisfy the Boolean algebra requirement of the Theorem above, but the unit interval example does not.

Dual revision programs. In [8] the notion of a *dual* revision program was introduced. $\mathcal{P}^D$ is the dual of $\mathcal{P}$ if every occurrence of **out** is replaced with an occurrence of **in**, and conversely. In fact the behavior of a dual program is easily characterized in bilattice terms — the program operator is the dual operator in the most straightforward sense. Suppose, for notational simplicity, we let $\mathcal{Q}$ be the dual of $\mathcal{P}$. Then it is quite easy to see that:

$$T^b_{\mathcal{Q}}(v) = \neg T^b_{\mathcal{P}}(\neg v).$$

It follows immediately that if C is a fixed point of $T^b_{\mathcal{P}}$, then $\neg C$ is a fixed point of $T^b_{\mathcal{Q}}$, because $T^b_{\mathcal{Q}}(\neg C) = \neg T^b_{\mathcal{P}}(\neg\neg C) = \neg T^b_{\mathcal{P}}(C) = \neg C$. It now follows from the previous discussion that dual programs can be used, under appropriate circumstances, to compute inverse changes.

Preprocessing. Both the original Marek and Truszczyński version and the present generalization only serve to verify that a candidate B_R for a revision of B_I in accordence with $\mathcal{P}$ really is one. No method is provided for computing a revision — indeed it is not clear what such a method would compute, since revisions are not generally unique, as Example A shows. What we consider now is whether there is some technique for computing at least that portion of the necessary changes that are common to all changes leading to correct revisions. We propose a simple method that is, in a sense, correct but not complete. Exactly what this means will become clear after the method has been presented.

Recall that for a given annotated revision program $\mathcal{P}$d an operator $\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$ was defined to be $T^b_{\mathcal{P}_{B_R}|B_I}(v)$, and we observed that this was monotonic in each of B_I, B_R, and v, with respect to the $\leq_k$ ordering. Then we defined the necessary change for $\mathcal{P}_{B_R}|B_I$ to be the least fixed point of $(\lambda v)\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v)$. It is convenient now to turn this notion itself into an operator.

$$\mathfrak{R}'_{\mathcal{P}}(B_I, B_R) = \text{the least fixed point of } (\lambda v)\mathfrak{R}_{\mathcal{P}}(B_I, B_R, v).$$

Then $\mathfrak{R}'_{\mathcal{P}}(B_I, B_R)$ is the necessary change (candidate) for revising B_I into B_R. It is not hard to verify, from general lattice properties, that $\mathfrak{R}'_{\mathcal{P}}(B_I, B_R)$ itself is monotonic in both B_I and B_R, in the $\leq_k$ ordering.

Definition 11. Let C_{B_I} be the least fixed point of $(\lambda w)\mathfrak{R}'_{\mathcal{P}}(B_I, w)$.

The utility of this notion is simple: it yields a $\mathcal{T} \odot \mathcal{T}$-valuation that is compatible with the necessary change for revising B_I, *no matter what the candidate for revision might be*. More precisely, we have the following.

Theorem 12. *Suppose B_R is a* $\mathcal{P}$-justified revision of B_I, *and C is the necessary change for $\mathcal{P}_{B_R}|B_I$. Then $C_{B_I} \leq_k C$.*

Proof. Since C is the necessary change for $\mathcal{P}_{B_R}|B_I$, $\mathfrak{R}'_{\mathcal{P}}(B_I, B_R) = C$. Also since B_R is a $\mathcal{P}$-justified revision of B_I, $B_R = (B_I \otimes -C) \oplus C$, so $C \leq_k B_R$. Since $\mathfrak{R}'_{\mathcal{P}}(x, y)$ is monotonic in both inputs, $\mathfrak{R}'_{\mathcal{P}}(B_I, C) \leq \mathfrak{R}'_{\mathcal{P}}(B_I, B_R) = C$. It follows that the least fixed point of $(\lambda w)\mathfrak{R}'_{\mathcal{P}}(B_I, w)$ is $\leq_k C$, or $C_{B_I} \leq_k C$.

What this means is, if we are given $\mathcal{P}$ and an initial knowledge base B_I, we can compute at least some of the necessary change, C_{B_I}, before we have a candidate B_R for a revised knowledge base. In a sense, C_{B_I} constitutes the *uniform* part of the change, that is, the part whose justification will be the same for every B_R. An example should help clarify this.

Example B Let $\mathcal{P}$ be the following annotated revision program, where the space of annotations is the collection of subsets of $\{p, q\}$.

$$\begin{aligned}
(\mathbf{out}(a) : \{p\}) &\leftarrow (\mathbf{in}(b) : \{p\}) \\
(\mathbf{out}(b) : \{p\}) &\leftarrow (\mathbf{in}(a) : \{p\}) \\
(\mathbf{in}(c) : \{p\}) &\leftarrow (\mathbf{in}(a) : \{p\}) \\
(\mathbf{in}(c) : \{p\}) &\leftarrow (\mathbf{in}(b) : \{p\}) \\
(\mathbf{in}(a) : \{q\}) &\leftarrow \\
(\mathbf{out}(b) : \{q\}) &\leftarrow (\mathbf{in}(a) : \{q\})
\end{aligned}$$

Also let B_I be the following.

$$\begin{aligned}
B_I(a) &= \langle \{p\}, \emptyset \rangle \\
B_I(b) &= \langle \{p\}, \emptyset \rangle \\
B_I(c) &= \langle \emptyset, \emptyset \rangle
\end{aligned}$$

We can then calculate C_{B_I} by the usual technique of starting with $\bot$ and iterating operator application.

$$\begin{aligned}
C_{B_I}(a) &= \langle \{q\}, \emptyset \rangle \\
C_{B_I}(b) &= \langle \emptyset, \{q\} \rangle \\
C_{B_I}(c) &= \langle \emptyset, \emptyset \rangle.
\end{aligned}$$

Next, consider the following, both of which are, in fact, $\mathcal{P}$-justified revisions of B_I.

$$\begin{aligned}
B^1_R(a) &= \langle \{p, q\}, \emptyset \rangle \\
B^1_R(b) &= \langle \emptyset, \{p, q\} \rangle \\
B^1_R(c) &= \langle \{p\}, \emptyset \rangle
\end{aligned}$$

$$B^2_R(a) = \langle \{q\}, \{p\} \rangle$$
$$B^2_R(b) = \langle \{p\}, \{q\} \rangle$$
$$B^2_R(c) = \langle \{p\}, \emptyset \rangle$$

For B^1_R the necessary change turns out to be the following.

$$C^1(a) = \langle \{q\}, \emptyset \rangle$$
$$C^1(b) = \langle \emptyset, \{p, q\} \rangle$$
$$C^1(c) = \langle \{p\}, \emptyset \rangle$$

And for B^2_R the necessary change is the following.

$$C^2(a) = \langle \{q\}, \{p\} \rangle$$
$$C^2(b) = \langle \emptyset, \{q\} \rangle$$
$$C^2(c) = \langle \{p\}, \emptyset \rangle$$

As predicted, $C_{B_I} \leq_k C^1$ and $C_{B_I} \leq_k C^2$. Note, however, that if C is any change that effects a $\mathcal{P}$-justified revision of B_I, it must be the case that $\langle \{p\}, \emptyset \rangle \leq_k C(c)$, though $\langle \{p\}, \emptyset \rangle \not\leq_k C_{B_I}(c)$. Essentially this is because the reasons p has for asserting c are not uniform in all models. In B^1_R, p asserts c because p asserts a, while in B^2_R it is because p asserts b. Nonetheless, use of C_{B_I} should allow a certain amount of preprocessing, and thus improve efficiency somewhat.

9 Conclusion

We believe the understanding of revision specification programs is still in early stages. Algorithms for computing revisions are missing. Non-trivial examples to which the theory applies would be useful. Still we believe generalizing as we have done does not obfuscate, but clarify — serving to bring out the essential algebraic structure underneath. We intend to continue with further investigations. We urge others to join in.

References

1. Fitting, M. C. Bilattices in logic programming. In *The Twentieth International Symposium on Multiple-Valued Logic* (1990), G. Epstein, Ed., IEEE, pp. 238–246.
2. Fitting, M. C. Bilattices and the semantics of logic programming. *Journal of Logic Programming 11* (1991), 91–116.
3. Fitting, M. C. The family of stable models. *Journal of Logic Programming 17* (1993), 197–225.
4. Fitting, M. C. Kleene's three-valued logics and their children. *Fundamenta Informaticae 20* (1994), 113–131.
5. Fitting, M. C. On prudent bravery and other abstractions. Submitted, 1994.
6. Ginsberg, M. L. Multivalued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence 4* (1988), 265–316.
7. Kifer, M., and Subrahmanian, V. S. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming 12* (1992), 335–367.
8. Marek, V. W., and Truszczyński, M. Revision specifications by means of programs. Presented at LPNMR workshop, Lexington, KY, 1994., 1994.