On the Use of Epistemic Ordering Functions as Decision Criteria for Automated and Assisted Belief Revision in SNePS

by

Ari Ilan Fogel

A thesis submitted to the
Faculty of the Graduate School of
the University at Buffalo, State University of New York
in partial fulfillment of the requirements for the
degree of

Master of Science

Department of Computer Science and Engineering

June 1, 2011

Acknowledgments

I would like to express my deepest thanks to my major professor, Dr. Stuart Shapiro, who has been a wonderful mentor these past two years. It was he who introduced me to knowledge representation, and he has been extremely influential in shaping the direction of my academic career. Throughout our interactions I have always taken his comments to heart. His wisdom and scholarship (and patience) have been an example to me, qualities I can only hope someday to attain to a similar degree.

I would like to thank Prof. William Rapaport for serving on my thesis committee. His comments have been invaluable in the course of my research. He also opened up the world of teaching to me as a green TA. He has been a great boss, a veritable instructor in instruction. I have since had the opportunity to teach my own courses, for which I owe him a debt of gratitude.

I offer sincerest thanks to Prof. Russ Miller, who provided excellent critique on my algorithmic analysis days before a deadline. His comments have been integrated into this thesis, and are much appreciated.

I also want to thank Prof. Alan Selman. He taught the first course I attended in graduate school, which was incidentally the most difficult course I have ever taken in my life. And thus he prepared me for everything to follow.

Mom and Dad, you have been a constant source of encouragement in my life. You have always taught me to hold education in the highest regard, to give it the highest priority. My accomplishments are a testament to your values. Ayelet, you were always there for me in hard times growing up. Yoni, you are quite literally the reason I am a computer scientist. Daniel, I've always appreciated your advice over the years, even (especially) when I haven't taken it. You've all made me who I am today. I love you all.

Contents

Abstract

In this thesis I implement belief revision in SNePS based on a user-supplied epistemic ordering of propositions. I provide a decision procedure that performs revision completely automatically when given a well preorder. I also provide a decision procedure that, when given a total preorder, performs revision with a minimal number of queries to the user when multiple propositions within a minimally-inconsistent set are minimally epistemically entrenched. These procedures are implemented in SNePS as options alongside the old belief revision subsystem, wherein revision must be done entirely by hand. I implement both prioritized and nonprioritized belief revision by adjusting the epistemic ordering function passed to the new procedures. The first procedure uses $O(|\Sigma|)$ units of space, and completes within $O(|\Sigma|^2 \cdot s_{max})$ units of time, where Σ is the set of distinct minimally-inconsistent sets, and s_{max} is the number of propositions in the largest minimally-inconsistent set. The second procedure uses $O(|\Sigma|^2 \cdot s_{max}^2)$ space and $O(|\Sigma|^2 \cdot s_{max}^2)$ time. The examples provided herein demonstrate that the new procedures generalize previous work on belief revision in SNePS.

1 Introduction

1.1 Belief Revision

At its most basic level, *belief revision* refers to the general problem of changing belief states (Gärdenfors and Rott, 1995). Several varieties of belief revision have appeared in the literature over the years. AGM revision typically refers to the addition of a belief to a belief set, at the expense of its negation and any other beliefs supporting its negation (Alchourron et al., 1985). Alternatively, revision can refer to the process of resolving inconsistencies in a contradictory knowledge base, or one *known to be inconsistent* (Martins and Shapiro, 1988). This is accomplished by removing one or more beliefs responsible for the inconsistency, or *culprits*. This is the task with which I am concerned. In particular, I have devised a means of automatically resolving inconsistencies by discarding the least-preferred beliefs in a belief base, according to some *epistemic ordering* (Gärdenfors, 1988; Williams, 1994; Gärdenfors and Rott, 1995).

1.1.1 Motivation

Let us say that you are told to believe the following information (the logical formalization appears on the right):

Blackbeard was a pirate	$Pirate(Blackbeard)$
All pirates are uneducfated	$\forall x[Pirate(x) \rightarrow \neg Educated(x)]$

Using the rules of classical first-order logic, the following fact is derivable:

Blackbeard was uneducated $\neg Educated(Blackbeard)$

But you know in your heart that Blackbeard was indeed an educated man:

Blackbeard was educated $Educated(Blackbeard)$

If you choose to accept all of the above information as fact, then you hold a contradictory set of beliefs. That is, you believe that Blackbeard both is and is not educated:$Educated(Blackbeard) \wedge \neg Educated(Blackbeard)$

We would say that your belief set is *inconsistent*. In order to restore consistency, you would need to *contract* some belief, or remove a belief from your belief set. The result would be a *revision* of your beliefs (Gärdenfors and Rott, 1995).

> The problem of belief revision is that logical considerations alone do not tell you which beliefs to give up, but this has to be decided by some other means. What makes things more complicated is that beliefs in a database have logical consequences. So when giving up a belief you have to decide as well which of its *consequences* to retain and which to retract... (Gärdenfors and Rott, 1995)

In later sections I will discuss in detail how to make a choice of belief(s) to retract when presented with an inconsistent belief set.

1.1.2 AGM Paradigm

In (Gärdenfors, 1982; Alchourron et al., 1985), operators and rationality postulates for *theory change* are discussed. Let $Cn(A)$ refer to the closure under logical consequence of a set of propositions A. A *theory* is defined to be a set of propositions closed under logical consequence. Thus

for any set of propositions A, $Cn(A)$ is a theory. We will use the terms *belief* and *proposition* interchangeably (and likewise *belief set* and *set of propositions*). It is worth noting that theories are infinite sets. (Alchourron et al., 1985) discusses operations that may be performed on theories. The following operations are part of what is referred to as the *AGM paradigm*.

1. **Expansion**: Given a set of propositions A and a new proposition x, the *expansion* of A by x is equal to $Cn(A \cup \{x\})$. Expansion of A by x is therefore the closure under logical consequence of the set-theoretic addition of x to A.

2. **Contraction:** Given a set of propositions A and a proposition x, the *contraction* of A by x, denoted $A \dot{-} x$, is equal to a maximal subset of A that fails to imply x (Alchourron and Makinson, 1982). So when contracting A by x, in addition to removing x from A, we remove additional propositions as necessary to ensure that x may not be rederived under logical consequence. (Gärdenfors, 1982) presents rationality postulates for contraction functions:

   $(\dot{-}1)$ $A \dot{-} x$ is a theory whenever A is a theory (closure).

   $(\dot{-}2)$ $A \dot{-} x \subseteq A$ (inclusion).

   $(\dot{-}3)$ If $x \notin Cn(A)$, then $A \dot{-} x = A$ (vacuity).

   $(\dot{-}4)$ If $x \notin Cn(\varnothing)$, then $x \notin Cn(A \dot{-} x)$ (success).

   $(\dot{-}5)$ If $Cn(x) = Cn(y)$, then $A \dot{-} x = A \dot{-} y$ (preservation).

   $(\dot{-}6)$ $A \subseteq Cn((A \dot{-} x) \cup \{x\})$ whenever A is a theory (recovery).

   The closure postulate $(\dot{-}1)$ states that the contraction of a theory is still a theory. The inclusion postulate $(\dot{-}2)$ states that contraction of a belief set cannot yield a belief set with any new beliefs. The vacuity postulate $(\dot{-}3)$ states that contraction of a belief set by a proposition which the belief set neither contains, nor implies as a result of logical consequence, yields the original belief set. The success postulate $(\dot{-}4)$ states that the belief set resulting from contraction by a non-tautological proposition will not imply that proposition as a result of logical

consequence. The preservation postulate $(\dot{-}5)$ states that if we may conclude the same things from two propositions by logical consequence, then the belief sets resulting from contraction of a belief set by either one of those propositions will be the same.

Finally we come to the recovery postulate $(\dot{-}6)$. This postulate states that if we contract a proposition from a belief set, then expand the resulting belief set by that same proposition to form a third belief set, there will be nothing in the original belief set that cannot be concluded from the information in the third belief set. There is extensive argument in the literature about the rationality of this particular postulate. I will discuss some particulars in §1.1.3.

3. **Revision:** Given a set of propositions A and a proposition x, the *revision* of A by x, denoted $A \dot{+} x$, is equal to
$Cn((A \dot{-} \neg x) \cup \{x\})$. Revision of A by x therefore accomplishes the addition of x to A in a fashion that is guaranteed not to produce an inconsistent theory. The mathematical reduction of revision to the form of contraction expressed above is called the Levi Identity (Levi, 1977). (Gärdenfors, 1982) presents rationality postulates for revision functions:

   $(\dot{+}1)$ $A \dot{+} x$ is always a theory.

   $(\dot{+}2)$ $x \in A \dot{+} x$.

   $(\dot{+}3)$ If $\neg x \notin Cn(A)$, then $A \dot{+} x = Cn(A \cup \{x\})$.

   $(\dot{+}4)$ If $\neg x \notin Cn(\varnothing)$, then $A \dot{+} x$ is consistent under Cn.

   $(\dot{+}5)$ If $Cn(x) = Cn(y)$, then $A \dot{+} x = A \dot{+} y$.

   $(\dot{+}6)$ $(A \dot{+} x) \cap A = A \dot{-} \neg x$, whenever A is a theory.

Postulate $(\dot{+}1)$ is self-explanatory. Postulate $(\dot{+}2)$ states that the revision of a belief set by a proposition results in a belief set containing that proposition. Postulate $(\dot{+}3)$ states that when the negation of a belief is not a consequence of a belief set, then revision of that belief set

by that belief will be the theory resulting from the closure under logical consequence of the expansion of the belief set by that belief. Postulate $(\dot{+}4)$ states that the revision of a belief set by a non-contradictory proposition results in a consistent belief set. It should be noted that this would hold even when the original belief set is contradictory, since the contradictory information would be contracted. Otherwise $\neg x$ could be rederived, since every belief is derivable from a contradictory set, and so the contraction seen in the Levi Identity would not really have occurred. Postulate $(\dot{+}5)$ states that if we may conclude the same things from two propositions by logical consequence, then the belief set resulting from revision of a belief set by either one those propositions will be the same. Postulate $(\dot{+}6)$ states that the common elements of A and the revision of A by x are in fact the elements of the contraction of A by $\neg x$ when A is a theory. This follows trivially from the Levi Identity.

The type of revision accomplished by the above operator may be thought of as *prioritized*; its RHS argument is added to the LHS belief set at the possible expense of pre-existing beliefs. That is, it prioritizes the RHS over other beliefs in the belief set. This follows from postulate $(\dot{+}2)$. I will discuss this notion in more detail in section 1.1.9.

4. **Partial Meet Contraction and Revision:** Let $A\perp x$ be the set of all maximal subsets B of A such that $B \not\vdash x$ (Alchourron et al., 1985). We say that a maxichoice contraction function $\dot{-}_{maxi}$ is one such that $A \dot{-}_{maxi} x$ is an arbitrary member of $A\perp x$ when the latter is nonempty, or else equal to A itself (Alchourron and Makinson, 1982). The full meet contraction $A \dot{-}_{fm} x$ is defined as $\bigcap(A\perp x)$ when A is nonempty, or else A itself. In (Alchourron and Makinson, 1982) it is shown that the result of full meet contraction is generally far too small to be of use.

Let γ be a function such that for all A, x, $\gamma(A\perp x)$ is a nonempty subset of $A\perp x$ when the latter is nonempty, or else the former is equal to $\{A\}$. γ is called a *selection function*. The operation $\dot{-}_{\gamma}$ defined by $A \dot{-}_{\gamma} x = \bigcap \gamma(A\perp x)$ is called the *partial meet contraction over A determined by γ* (Alchourron et al., 1985). Note that full meet contraction is equivalent to the

special case of partial meet contraction where $\gamma(A\perp x) = A\perp x$, and maxichoice contraction is the special case where $\gamma(A\perp x)$ is a singleton set. Partial meet revision is simply revision using partial meet contraction as the contraction function seen in the Levi Identity. The selection function γ is used as a way to choose the most desirable members of $A\perp x$. Such a selection function could possibly be used in determining how to proceed from the situation presented in section 1.1.1. However it is rather impractical to attempt to use any of the pure AGM operations presented in this section, since they tend to involve infinite theories. The next section brings us closer to something practical. I conclude by noting that partial meet contraction and revision satisfy all of the above postulates for AGM contraction and revision respectively (Alchourron et al., 1985).

1.1.3 Theory Change on Finite Bases

> It is widely accepted that agents, because of their limited resources, believe some but by no means all of the logical consequences of their beliefs. (Lakemeyer, 1991)

A major issue with the AGM paradigm is that it defines operations to be performed on infinite sets (theories). A more practical model would include operations to be performed on finite belief sets, or *belief bases*. Such operators would be useful in supporting computer-based implementations of revision systems (Williams, 1994).

(Hansson, 1997, 1999b) discusses finite-base analogues to the AGM operators. An important difference between these two paradigms is the fact that the finite base operators cannot satisfy the closure and recovery postulates for AGM contraction (Dixon, 1993; Williams, 1994). Obviously, satisfaction of the closure postulate is not desirable since that would result in infinite theories. Some argue that satisfaction of the recovery postulate is not desirable because it is inextricably

tied to the consequence relation (Williams, 1994; Makinson, 1987). Defense of recovery, on the other hand, is predicated upon the notion that contractions of a belief set should cause a minimal removal of information — so minimal, in fact, that expansion by a contracted belief should be sufficient to recapture the original beliefs. In this vein, Johnson makes a compelling point that recovery is desirable in the case of belief revision with reconsideration (Johnson, 2006, 35–42), i.e. iterated belief revision whereby a belief removed at one step no longer has a reason to be absent after the last step. My new procedures do not satisfy the recovery postulate (see §2.4).

1.1.4 Epistemic Entrenchment

Consider again the four scenarios in section 1.1.9. It is worthwhile to examine the mechanism by which you decided exactly which belief(s) to retract in the scenarios presented. In fact there is a similar mechanism at play in all of them. Let us assume that the decision on which beliefs to retract from a belief set is made is based on the relative importance of each belief, which is called its degree of *epistemic entrenchment* (Gärdenfors, 1988). Then we need an ordering $\leqslant$ with which to compare the entrenchment of individual beliefs. This ordering may be used to determine a selection function γ with which to define a partial meet revision function. It is because a different epistemic ordering is used in scenarios 2 and 3 from the one used in scenarios 4 and 5 that different choices are made on which belief to retract. In fact prioritized revision is just the special case where new beliefs are always more entrenched than all existing beliefs.

(Gärdenfors, 1988) presents rationality postulates for an epistemic ordering $\leqslant$.
(K is a belief set, and A, B, and C are beliefs. $\perp$ represents the contradictory belief set.) Such an ordering is a noncircular total preorder on *all propositions*.

(EE1) For any A,B, and C, if $A \leqslant B$ and $B \leqslant C$, then $A \leqslant C$. (Transitivity)

(EE2) For any A and B, if $A \vdash B$, then $A \leqslant B$. (Dominance)

(EE3) For all A and B in K, $(A \leqslant A \wedge B)$ or $(B \leqslant A \wedge B)$. (Conjunctiveness)

(EE4) When $K \neq \bot$, $A \notin K$ iff $A \leqslant B$ for all B. (Minimality)

(EE5) If $B \leqslant A$ for all B, then $\vdash A$.

(EE1) is self-explanatory. (EE2) states that we should prefer to remove a belief from which a second belief may be derived, than to remove the second derived belief. The reason is that the derived belief may be rederived otherwise. (EE3), when combined with (EE1) and (EE2), states that we must remove either A or B to remove $A \wedge B$. (EE4) states that propositions that are less entrenched than all other propositions are not in K (unless K is contradictory). (EE5) states that propositions with maximal entrenchment are theorems.

Postulate (EE3) does not apply to SNeBR (see §2.5) due to the distinction between hypotheses and derived beliefs. Postulate (EE4) does not apply to SNeBR since only propositions *within* a belief base are ever considered for removal when an inconsistency is detected.

1.1.5 Ensconcements

Ensconcements, introduced in (Williams, 1994), consist of a set of forumlae together with a total preorder on that set. They can be used to construct epistemic entrenchment orderings, and determine theory base change operators.

1.1.6 Safe Contraction

In (Alchourron and Makinson, 1985), the operation *safe contraction* is introduced. Let $<$ be a non-circular relation over a belief set A. An element a of A is *safe* with respect to x iff a is not a minimal element of any minimal subset B of A such that $x \in Cn(B)$. Let A/x be the set of all

elements of A that are safe with respect to x. Then the safe contraction of A by x, denoted $A \dot{-}_s x$, is defined to be $A \cap Cn(A/x)$.

1.1.7 Assumption-based Truth Maintenance Systems

In an assumption-based truth maintenance system (ATMS), the system keeps track of the assumptions (base beliefs) underlying each belief (de Kleer, 1986). One of the roles of an conventional TMS is to keep the database contradiction-free. In an assumption-based ATMS, contradictions are removed as they are discovered. When a contradiction is detected in an ATMS, then there will be one or more minimally-inconsistent sets of assumptions underlying the contradiction. Such sets are called *no-goods*. (Martins and Shapiro, 1988) presented SNeBR, an early implementation of an ATMS that uses the logic of SNePS. In that paper, sets of assumptions supporting a belief are called *origin sets*. They correspond to *antecedents* of a *justification* from (de Kleer, 1986). The focus of this thesis is modifications to the modern version of SNeBR.

1.1.8 Kernel Contraction

In (Hansson, 1994), the operation *kernel contraction* is introduced. A *kernel set* $A \Subset \alpha$ is defined to be the set of all minimal subsets of A that imply α. A kernel set is like a set of *origin sets* from (Martins and Shapiro, 1988). Let σ be an *incision function* for A. Then for all α, $\sigma(A \Subset \alpha) \subseteq \cup(A \Subset \alpha)$, and if $\varnothing \neq X \in A \Subset \alpha$, then $X \cap \sigma(A \Subset \alpha) \neq \varnothing$. The *kernel contraction* of A by α based on σ, denoted $A \sim_\sigma \alpha$, is equal to $A \backslash \sigma(A \Subset \alpha)$.

1.1.9 Prioritized Versus Non-Prioritized Belief Revision

> In the AGM model of belief revision (Alchourron et al., 1985) ... the input sentence is always accepted. This is clearly an unrealistic feature, and ... several models of belief change have been proposed in which no absolute priority is assigned to the new information due to its novelty. ... One way to construct non-prioritized belief revision is to base it on the following two-step process: First we decide whether to accept or reject the input. After that, if the input was accepted, it is incorporated into the belief state (Hansson, 1999a).

Hansson goes on to describe several other models of nonprioritized belief revision, but they all have one unifiying feature distinguishing them from prioritized belief revision: the input, i.e. the RHS argument to the revision operator, is not always accepted. To reiterate: *Prioritized belief revision* is revision in which the proposition by which the set is revised is always present in the result (as long as it is not a contradiction). *Non-prioritized belief revision* is revision in which the RHS argument to the revision operator is not always present in the result (even if it is not a contradiction).

The closest approximation of revision in SNePS by Hansson is the operation of *semi-revision* (Hansson, 1997). Semi-revision is a type of non-prioritized belief revision that may be applied to belief bases. So it is a bridge between the AGM paradigm and SNePS revision, as we shall see in §1.2.

Let us revisit the example from §1.1.1:

You believe that:

(*Your belief set S consists of*:)

- Blackbeard was a pirate.

  $Pirate(Blackbeard)$

- All pirates are uneducated

$\forall x[Pirate(x) \rightarrow \neg Educated(x)]$

- Consequently, Blackbeard was not educated

  $\neg Educated(Blackbeard)$

Now consider these five scenarios:

1. You are told that Blackbeard was educated. You *weigh this notion against your previous beliefs*, end up deciding that it is not reasonable, and reject it. Your belief set S has not changed.

2. You are told that Blackbeard was educated. You *weigh this notion against your previous beliefs*, end up deciding that it is reasonable, and accept it as fact. As a result, you abandon your prejudice and discontinue holding the belief that all pirates are uneducated. That is, you retract $\neg Educated(Blackbeard)$ from your belief set S, using a selection function γ, as in §1.1.2 part 4, that removes $\forall x[Pirate(x) \rightarrow \neg Educated(x)]$ over $Pirate(Blackbeard)$.

3. You are told that Blackbeard was educated. You *weigh this notion against your previous beliefs*, end up deciding that it is reasonable, and accept it as fact. You are a persistent bigot, so you discontinue holding the belief that Blackbeard was ever a real pirate (you prefer to believe that no pirate is educated). That is, you retract $\neg Educated(Blackbeard)$ from your belief set S, using a selection function γ as in section 1.1.2 part 4, that removes $Pirate(Blackbeard)$ over $\forall x[Pirate(x) \rightarrow \neg Educated(x)]$.

4. You are told that Blackbeard was educated. You assume that this is true *because it is the last thing you have been told*. As a result, you abandon your prejudice and discontinue holding the belief that all pirates are uneducated. That is, you retract $\neg Educated(Blackbeard)$ from your belief set S, using a selection function γ as in section 1.1.2 part 4, that removes $\forall x[Pirate(x) \rightarrow \neg Educated(x)]$ over $Pirate(Blackbeard)$.

5. You are told that Blackbeard was educated. You assume that this is true *because it is the last thing you have been told*. You are a persistent bigot, so you discontinue holding the belief that Blackbeard was ever a real pirate (you prefer to believe that no pirate is educated). That is, you retract $\neg Educated(Blackbeard)$ from your belief set S, using a selection function γ as in section 1.1.2 part 4, that removes $Pirate(Blackbeard)$ over $\forall x[Pirate(x) \rightarrow \neg Educated(x)]$.

In scenarios 1, 2 and 3 you considered whether the new information you were just told was true or false. You did not *automatically* prioritize the new information over your existing beliefs. In scenarios 2 and 3 in particular, the decision to keep the new belief was made *after weighing it against all the previously held beliefs, rather than on the basis of its novelty*. So you were performing *non-prioritized belief revision* on your belief set (Hansson, 1999a). In scenarios 4 and 5 you took as fact the new information you were given without considering its veracity. You automatically prioritized the new information over your existing beliefs. So you were performing *prioritized belief revision* on your belief set.

1.2 SNePS

(Except where otherwise noted, all material in this section comes from (Shapiro and The SNePS Implementation Group, 2010).)

1.2.1 Description of the System

"SNePS is a logic-, frame-, and network- based knowledge representation, reasoning, and acting system. Its logic is based on Relevance Logic (Shapiro, 1992), a paraconsistent logic (in which a contradiction does not imply anything whatsoever)" (Shapiro and Johnson, 2000). SNePS is divided into several packages, a few of which I discuss here:

SNePSLOG is an interface to SNePS in which information is entered in a predicate logic notation. It is the interface through which my demonstrations are performed (see §4).

SNeRE, the SNePS Rational Engine, provides an acting system for SNePS-based agents, whose beliefs must change to keep up with a changing world. Of particular interest is the *believe* action, which is used to introduce beliefs that take priority over all other beliefs at the time of their introduction.

1.2.2 Belief Change in SNePS

Every belief in a SNePS belief base has one or more *support sets*, each of which consists of an *origin tag* and an *origin set*. The origin tag will identify a belief as either being introduced as a hypothesis, or derived (note that it is possible for a belief to be both introduced as a hypothesis and derived from other beliefs). The origin set contains those *hypotheses* that were used to derive the belief. In the case of the origin tag denoting a hypothesis, the corresponding origin set would be a singleton set containing only the belief itself. The contents of the origin set of a derived belief are computed by the implemented rules of inference at the time the inference is drawn (Martins and Shapiro, 1988; Shapiro, 1992).

The representation of beliefs in SNePS lends itself well to the creation of processes for con-

traction and revision. Specifically, in order to contract a belief, one must merely remove at least one hypothesis from each of its origin sets. Similarly, prioritized revision by a belief b (where $\neg b$ is already believed) is accomplished by removing at least one belief from each origin set of $\neg b$. Non-prioritized belief revision under this paradigm is a bit more complicated. I discuss both types of revision in more detail in §2.

1.2.3 SNeBR

SNeBR, The SNePS Belief Revision subsystem, is responsible for resolving inconsistencies in the knowledge base as they are discovered. In the current release of SNePS (version 2.7.1), SNeBR is able to *automatically* resolve contradictions under a limited variety of circumstances. Otherwise "assisted culprit choosing" is performed, where the user must manually select culprits for removal. After belief revision is performed, the knowledge base might still be inconsistent, but every *known* derivation of an inconsistency has been eliminated.

> If the SNeRE believe act is performed on the proposition p, it is assumed that belief in p is to take priority over any contradictory belief. Therefore, SNeBR behaves as follows.
> 1. If andor(0,0){p, ...}[1] is believed as an hypothesis, it is chosen as the culprit. If it is a derived belief, assisted culprit choosing is done.
> 2. If andor(i,1){p, q, ...}[2] (i is either 0 or 1) is believed and q is believed as an hypothesis, then q is chosen as the culprit. If q is a derived belief, assisted culprit choosing is done. (Shapiro and The SNePS Implementation Group, 2010)

So automatic belief revision in this incarnation of SNeBR is heavily dependent on syntax. Additionally, it is not extensible. My modified SNeBR addresses these limitations.

[1] andor(0,0){...} means that every proposition within the braces is false. $\neg p$ is represented in SNePS as andor(0,0){p}.

[2] andor(i,1){...} means that at least i and at most 1 of the propositions within the braces is true.

2 New Belief Revision Algorithms

2.1 Problem Statement

2.1.1 Nonprioritized Belief Revision

Suppose we have a knowledge base that is not known to be inconsistent, and suppose that at some point we add a contradictory belief to that knowledge base. Either that new belief directly contradicts an existing belief, or we derive a belief that directly contradicts an existing one as a result of performing forward and/or backward inference on the new belief. Now the knowledge base is known to be inconsistent. We will refer to the contradictory beliefs as p and $\neg p$. Contraction of either p or $\neg p$ from the knowledge base will resolve the contradiction.

Since SNePS tags each belief with one or more origin sets, or sets of supporting hypotheses, we can identify the underlying beliefs that support each of the two contradictory beliefs. In the case where p and $\neg p$ each have one origin set, OS_p and $OS_{\neg p}$ respectively, we may resolve the contradiction by removing at least one hypothesis from OS_p, thereby removing p, or at least one hypothesis from $OS_{\neg p}$, thereby removing $\neg p$. Since we only need to remove one of these propositions, the contradiction may be resolved by removing at least one belief from $OS_p \cup OS_{\neg p}$, a *no-good*. If there are m origin sets for p, and n origin sets for $\neg p$, then there will be at most $m \times n$ distinct no-goods (some unions may be duplicates of others). To resolve a contradiction in this case, we must retract at least one hypothesis from each no-good (Sufficiency).

I will present an algorithm that will select the hypotheses for removal from the set of no-goods. The first priority will be that the hypotheses selected should be minimally-epistemically-entrenched (Minimal Entrenchment) according to some total preorder $\leqslant$. The second priority will be not to

remove any more hypotheses than are necessary in order to resolve the contradiction (Information Preservation), while still satisfying priority one.

2.1.2 Prioritized Belief Revision

The process of Prioritized Belief Revision in SNePS occurs when a contradiction is discovered after a belief is asserted explicitly using the *believe* act of SNeRE. The major difference from non-prioritized belief revision is that a subtle change is made to the entrenchment ordering $\leqslant$. If $\leqslant_{nonpri}$ is the ordering used for nonprioritized belief revision, then for prioritized belief revision we use an ordering $\leqslant_{pri}$ as follows:
Let P be the set of beliefs asserted by a *believe* action. Then

$\forall e_1, e_2 [e_1 \in P \wedge e_2 \notin P \rightarrow \neg(e_1 \leqslant_{pri} e_2) \wedge e_2 \leqslant_{pri} e_1]$

$\forall e_1, e_2 [e_1 \notin P \wedge e_2 \notin P \rightarrow (e_1 \leqslant_{pri} e_2 \leftrightarrow e_1 \leqslant_{nonpri} e_2)]$

$\forall e_1, e_2 [e_1 \in P \wedge e_2 \in P \rightarrow (e_1 \leqslant_{pri} e_2 \leftrightarrow e_1 \leqslant_{nonpri} e_2)]$

That is, a proposition asserted by a *believe* action takes priority over any other proposition. When either both or neither propositions being compared have been asserted by the *believe* action, then we use the same ordering as we would for nonprioritized revision.

2.2 Common Requirements for a Rational Belief Revision Algorithm

2.2.1 Primary Requirements

The inputs to the algorithm are:

- A set of formulae Φ: the current belief set, which is known to be inconsistent

- A total preorder $\leqslant$ on Φ: an epistemic entrenchment ordering that can be used to compare the relative desirability of each belief in the current belief set
- Minimally-inconsistent sets of formulae $\sigma_1, \ldots, \sigma_n$, each of which is a subset of Φ: the no-goods
- A set $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$: the set of all the no-goods

The algorithm should produce a *culprit set* T that satisfies the following conditions:

$(EE_{SNePS}1)$ $\forall\sigma[\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T \cap \sigma)]$ (Sufficiency)

$(EE_{SNePS}2)$ $\forall\tau[\tau \in T \rightarrow \exists\sigma[\sigma \in \Sigma \wedge \tau \in \sigma \wedge \forall w[w \in \sigma \rightarrow \tau \leqslant w]]]$ (Minimal Entrenchment)

$(EE_{SNePS}3)$ $\forall T'[T' \subset T \rightarrow \neg\forall\sigma[\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T' \cap \sigma)]]]$ (Information Preservation)

Condition $(EE_{SNePS}1)$ states that T contains at least one formula from each set in Σ. Condition $(EE_{SNePS}2)$ states that every formula in T is a minimally-entrenched formula of some set in Σ. Condition $(EE_{SNePS}3)$ states that if any formula is removed from T, then Condition $(EE_{SNePS}1)$ will no longer hold. In addition to the above conditions, the algorithm must terminate on all possible inputs, i.e. it must be a decision procedure.

2.2.2 Supplementary Requirement

In any case where queries must be made of the user in order to determine the relative epistemic ordering of propositions, the number of such queries must be kept to a minimum.

2.3 Implementation

I present algorithms to solve the problem as stated:

Where I refer to $\leqslant$ below, I am using the *prioritized* entrenchment ordering from §2.1. In the case of nonprioritized revision we may assume that $P = \varnothing$

2.3.1 Using a well preorder

Let $\trianglelefteqslant_{\leqslant}$ be the output of a function f whose input is a total preorder $\leqslant$, such that $\trianglelefteqslant_{\leqslant} \subseteq \leqslant$ The idea is that f creates the well preorder $\trianglelefteqslant_{\leqslant}$ from $\leqslant$ by removing some pairs from the total preorder $\leqslant$. Note that in the case where $\leqslant$ is already a well preorder, $\trianglelefteqslant_{\leqslant} = \leqslant$. Then we may use Algorithm 1 to solve the problem.

Algorithm 1 Algorithm to compute T given a well preorder

Input: $\Sigma, \trianglelefteqslant_{\leqslant}$
Output: T

1: $T \Leftarrow \varnothing$
2: **for all** $(\sigma \in \Sigma)$ **do**
3: Move minimally entrenched belief in σ to first position in σ, using $\trianglelefteqslant_{\leqslant}$ as a comparator
4: **end for**
5: Sort elements of Σ into descending order of the values of the first element in each σ using $\trianglelefteqslant_{\leqslant}$ as a comparator
6: *AddLoop* :
7: **while** $(\Sigma \neq \varnothing)$ **do**
8: $currentCulprit \Leftarrow \sigma_{1_1}$
9: $T \Leftarrow T \cup \{currentCulprit\}$
10: *DeleteLoop* :
11: **for all** $(\sigma_{current} \in \Sigma)$ **do**
12: **if** $(currentCulprit \in \sigma_{current})$ **then**
13: $\Sigma \Leftarrow \Sigma \backslash \sigma_{current}$
14: **end if**
15: **end for**
16: **end while**
17: **return** T

2.3.2 Using a total preorder

Unfortunately it is easy to conceive of a situation in which the supplied entrenchment ordering is a total preorder, but not a well preorder. For instance, let us say that, when reasoning about a changing world, propositional fluents (propositions that are only true of a specific time or situation) are abandoned over non-propositional fluents. It is not clear then how we should rank two distinct propositional fluents, nor how to rank two distinct non-propositional fluents. If we can arbitrarily specify a well preorder that is a subset of the total preorder we are given, then algorithm 1 will be suitable. Otherwise, we can simulate a well preorder $\leqslant$ through an iterative construction by querying the user for the minimally-entrenched proposition of a particular set of propositions at appropriate times in the belief-revision process. Algorithm 2 accomplishes just this.

Algorithm 2 Algorithm to compute T given a total preorder

Input: $\Sigma, \leqslant$

Output: T

$T \Leftarrow \varnothing$
MainLoop:
loop
  ListLoop:
  for all $(\sigma_i \in \Sigma, 1 \leqslant i \leqslant |\Sigma|)$ **do**
    Make a list l_{σ_i} of all minimally-entrenched propositions, i.e. propositions that are not strictly more entrenched than any other, among those in σ_i, using $\leqslant$ as a comparator.
  end for
  RemoveLoop:
  for all $(\sigma_i \in \Sigma, 1 \leqslant i \leqslant |\Sigma|)$ **do**
    if (According to l_{σ_i}, σ has exactly one minimally-entrenched proposition p AND the other propositions in σ_i are not minimally-entrenched in any other no-good via an l_{σ_j},$(1 \leqslant j \leqslant |\Sigma|, i \neq j)$) **then**
      $T \Leftarrow T \cup \{p\}$
      for all $(\sigma_{current} \in \Sigma)$ **do**
        if $(p \in \sigma_{current})$ **then**
          $\Sigma \Leftarrow \Sigma \backslash \sigma_{current}$
        end if
      end for
      if $(\Sigma = \varnothing)$ **then**
        return T
      end if
    end if
  end for
  ModifyLoop:
  for all $(\sigma \in \Sigma)$ **do**
    if (σ has multiple minimally-entrenched propositions) **then**
      query which proposition l of the minimally-entrenched propositions is least desired.
      Modify $\leqslant$ so that l is strictly less entrenched than those other propositions.
      break out of *ModifyLoop*
    end if
  end for
end loop

2.3.3 Characterization

These algorithms perform an operation similar to *incision functions* (Hansson, 1994), since they select one or more propositions to be removed from each minimally-inconsistent set. Their output seems analogous to $\sigma(\Phi \between (p \wedge \neg p))$, where σ is the incision function, $\between$ is the kernel-set operator from (Hansson, 1994), and p is a proposition. But we are actually incising Σ, the set of *known* no-goods. The known no-goods are of course a subset of all no-goods, i.e. $\Sigma \subseteq \Phi \between (p \wedge \neg p)$. This happens because SNeBR resolves contradictions as soon as they are discovered, rather than performing inference first to discover all possible sources of contradictions.

The type of contraction eventually performed is similar to safe contraction (Alchourron and Makinson, 1985), except that there are fewer restrictions on our epistemic ordering.

2.4 The Recovery Postulate

When a contradiction occurs between a proposition p and another proposition $\neg p$, and $\neg p$ is retracted, we also retract one or more beliefs from which $\neg p$ is derived, as well as all beliefs that are derived from $\neg p$. If we were to immediately reassert $\neg p$ (let us ignore for the moment that this will trigger belief revision again), then while the beliefs derived from $\neg p$ may be reasserted, any beliefs from which $\neg p$ was derived that were removed previously will not automatically be reasserted (recovered). There is simply no mechanism in place to do this. So the procedures above do not in general satisfy the recovery postulate for belief revision. Recovery would only be possible in the case where $\neg p$ was originally asserted solely as a hypothesis. In this case the only beliefs lost during the initial revision would be those derived from $\neg p$. Those beliefs would be recovered as soon as $\neg p$ is reasserted (as long as the rest of their origin sets are intact).

2.5 Conjunctiveness

Since only hypotheses are considered for removal by these procedures, the conjunctiveness postulate (EE3) is unnecessary. If we have *a* and *b* as hypotheses, and from these we derive $a \wedge b$, then it is meaningless to compare *a* and $a \wedge b$, because we do not consider the entrenchment of derived beliefs. This is similarly true if we believe $a \wedge b$ as a hypothesis, and use it to derive *a* and to derive *b*.

3 Changes to SNePS Manual

3.1 New SNePSLOG Commands

3.1.1 br-mode

Syntax: **br-mode [auto|manual]**

The *br-mode* command is used to query or set the default behavior of belief revision.

- **Default Setting**:

  In modes 1 and 2, the default setting for *br-mode* is *manual*.

  In mode 3, the the default setting for *br-mode* is *auto*.

- *br-mode auto* will cause SNeBR to always attempt automatic belief revision when a contradiction is detected. While this mode is active, SNeBR will always resolve contradictions using the currently-selected automated belief revision algorithm. Manual belief revision will not be available.

- *br-mode manual* will cause SNeBR to present the user with a list of options specifying how

to proceed when a contradiction is detected. The user will be able to select manual belief revision, ignore the contradiction and proceed with an inconsistent knowledge base, or use the currently-selected automated belief revision algorithm.

- *br-mode* without any arguments will inform the user which of the above two modes is currently active.

3.1.2 set-order

Syntax: **set-order** <**function-name**>

The *set-order* command is used to choose the epistemic ordering function used by SNeBR during automated belief revision.

function-name must be the name of a lisp function visible to the *snepslog* package.

The function referred to by *function-name* must be a function of two arguments, *lhs* and *rhs*, as follows:

$func(lhs, rhs) = true$ iff $lhs \leqslant rhs$, i.e. *lhs* is not strictly more entrenched than *rhs*.

lhs and *rhs* are wffs.

The user may define his/her own ordering functions for use with *set-order*, or use one of the several built-in functions, described below:

- **Default Setting**:

  In modes 1 and 2, the default order is *null-order*.

  In mode 3, the the default order is *fluent*.

- *set-order null-order* will cause all propositions to be equally entrenched.

```
;;; An order in which all propositions are equally entrenched
(defun null-order (lhs rhs)
```

```
(declare (ignore lhs rhs))
t)
```

- *set-order explicit* will cause all propositions to be equally entrenched, except as follows:

  If *IsLessEntrenched(x,y)* is asserted in the knowledge base, then *x* will be strictly less entrenched than *y*. Note that this ordering function uses statements in the object language of SNePS to determine relative entrenchment of propositions.

```
;;; Description: An ordering function relying on explicit statements of
;;;              relative entrenchment of propositions, using the
;;;              IsLessEntrenched predicate for checks
;;; [IsLessEntrenched(x.y)] = [x] is strictly less entrenched than [y]
(defun explicit (lhs rhs)
  (not (tell "ask IsLessEntrenched(~A, ~A)" rhs lhs)))
```

- *set-order source* will cause all propositions to be equally entrenched, except as follows:
  If *HasSource(x,s1)*, *HasSource(y,s2)*, and *IsBetterSource(s2,s1)* are asserted in the knowledge base, then *x* will be strictly less entrenched than *y*. Note that this ordering function uses statements in the object language of SNePS to determine relative entrenchment of propositions.

```
;;;; Description: An ordering function that causes propositions with more
;;;              reliable sources to be more epistemically entrenched than
;;;              propositions with less reliable sources. Also, unsourced
;;;              propositions are more entrenched than sourced ones.
;;; [HasSource(x,y)] = The source of [x] is [y]
;;; [IsBetterSource(x,y)] = [x] is a better source than [y]
(defun source (lhs rhs)
  (or
    (and
      (let ((source-lhs (tell "askwh HasSource(~A, ?x)" lhs))
            (source-rhs (tell "askwh HasSource(~A, ?x)" rhs)))
        (cond ((and source-lhs source-rhs)
               (let ((source-lhs-term
                       (cdr (assoc 'x (first source-lhs))))
                     (source-rhs-term
                       (cdr (assoc 'x (first source-rhs)))))
                 (not (tell "ask IsBetterSource(~A, ~A)"
                            source-lhs-term source-rhs-term))))
              (source-rhs nil)
              (t t))))))
```

- *set-order fluent* will cause all propositions to be equally entrenched, except as follows:

  If *pred-sym* is contained in the $*fluents*$ list in lisp, then any proposition whose predicate symbol is *pred-sym* will be strictly less entrenched than every proposition whose predicate symbol is not contained in the $*fluents*$ list. Note that this ordering uses meta-information outside of the object language of SNePS in order to determine relative entrenchment of propositions.

```
;;; Description:  An ordering function that causes propositional fluents to
;;;               be less epistemically entrenched than non-fluent
;;;               propositions is a fluent or the rhs argument is not.
(defun fluent (lhs rhs)
  (or (is-fluent lhs)
    (not (is-fluent rhs))))

;;; Description:  Returns t iff the function symbol for n is a fluent
;;; Arguments:    n - a node
(defun is-fluent (n)
  (let ((pred (relation-predicate n)))
    (when (and pred (listp pred))
      (setf pred (get-node-name (car pred))))
    (member pred *fluents*)))
```

3.1.3 br-tie-mode

Syntax: **br-tie-mode [auto|manual]**

The *br-tie-mode* command is used to select the algorithm used for automated belief revision.

- **Default Setting**:

  In modes 1 and 2, the default setting for *br-tie-mode* is *manual*.

  In mode 3, the the default setting for *br-tie-mode* is *auto*.

- *br-tie-mode auto* will cause SNeBR to arbitrarily break entrenchment ties when multiple propositions are minimally entrenched within a minimally-inconsistent set. This mode uses a more time- and space- efficient algorithm to perform belief revision than *manual* mode. As such it is preferable whenever the user-supplied ordering is known to be a well preorder (no arbitrary decisions will be made), or when the user simply does not care how entrenchment ties are broken.

  In the current implementation of SNePS, a proposition whose name (e.g. wff**1**, wff**2**, etc.) has minimal lexicographic rank will be considered to be *strictly* less entrenched than all other propositions within a minimally inconsistent set, as long as no other propositions within the set are *strictly* less entrenched according to the user-supplied ordering. This has the effect of making propositions that were conceived of later more entrenched than those that are otherwise equally entrenched, but conceived of earlier. The order of conception should not be confused with the order of assertion, which may differ.

- *br-tie-mode manual* will cause SNeBR to query the user which proposition is *strictly* minimally entrenched when multiple propositions are minimally entrenched within a minimally-inconsistent set.

- *br-tie-mode* without any arguments will inform the user which algorithm is currently being used for automated belief revision.

3.2 New SNePSUL Commands

3.2.1 br-mode

Syntax: **(br-mode t**|**nil)**

- **Default Setting**: The default setting for *br-mode* in SNePSUL is *nil*
- *(br-mode t)* in SNePSUL will do the equivalent of entering *br-mode auto* in SNePSLOG
- *(br-mode nil)* in SNePSUL will do the equivalent of entering *br-mode manual* in SNePSLOG

3.2.2 set-order

Syntax: **(set-order** <**function-name**>**)**
function-name must be the name of a lisp function visible to the *sneps* package.
In order to use one of the built-in ordering functions, *function-name* must be prefixed with *snepslog:* (e.g. *snepslog:fluent*).

- **Default Setting**: The default setting for *set-order* in SNePSUL is *snepslog:null-order*
- *(set-order function-name)* will do the equivalent of entering *set-order function-name* in SNePSLOG.

3.2.3 br-tie-mode

Syntax: **(br-tie-mode t**|**nil)**

- **Default Setting**: The default setting for *br-tie-mode* in SNePSUL is *nil*
- *(br-tie-mode t)* in SNePSUL will do the equivalent of entering *br-tie-mode auto* in SNePSLOG
- *(br-tie-mode nil)* in SNePSUL will do the equivalent of entering *br-tie-mode manual* in SNePSLOG

3.3 Updates

> If the SNeRE believe act is performed on the proposition p, it is assumed that belief in p is to take priority over any contradictory belief. Therefore, SNeBR behaves as follows.
> 1. If andor(0,0){p, ...} is believed as an hypothesis, it is chosen as the culprit. If it is a derived belief, assisted culprit choosing is done.
> 2. If andor(i,1){p, q, ...} (i is either 0 or 1) is believed and q is believed as an hypothesis, then q is chosen as the culprit. If q is a derived belief, assisted culprit choosing is done.

§8.5.1 of the SNePS 2.7.1 manual (Shapiro and The SNePS Implementation Group, 2010, 76), quoted above, is no longer an accurate description of how SNeBR works. This text will be removed, and replaced with the text below:

> If the SNeRE believe act is performed on the proposition p, it is assumed that belief in p is to take priority over any contradictory belief. Thus if andor(0,0){p, ...} is believed, it is chosen as the culprit. Therefore, SNeBR behaves as follows:
> 1. If *br-mode* is set to *manual*, then the user will be given a choice to perform assisted culprit choosing, to attempt automated belief revision, or to continue in an inconsistent context.
> 2. If *br-mode* is set to *auto*, then automated belief revision will be attempted.
>
> Automated belief revision can proceed in one of two ways:
> 1. If *br-tie-mode* is set to *manual*, then SNeBR will query the user which proposition is *strictly* minimally entrenched whenever multiple propositions are minimally entrenched within a minimally-inconsistent set during the revision process.
> 2. If *br-tie-mode* is set to *auto*, then SNeBR will arbitrarily choose a proposition to be *strictly* minimally entrenched whenever multiple propositions are minimally entrenched within a minimally-inconsistent set during the revision process.
>
> For more information on how to set the epistemic ordering of propositions, see the *set-order* SNePSLOG command.

The portion of §8.5.2 (Shapiro and The SNePS Implementation Group, 2010, 76) that reads as follows

2. The user may choose to continue in an inconsistent context. That situation is not disastrous, since SNePS uses a paraconsistent logica contradiction does not imply irrelevant other propositions.

will be modified to read:

2. The user may choose to continue in an inconsistent context. That situation is not disastrous, since SNePS uses a paraconsistent logica contradiction does not imply irrelevant other propositions. Once the user chooses to continue in an inconsistent context, the SNePSLOG prompt will be prefixed with an asterisk (*) until consistency is restored.

4 Annotated Demonstrations

4.1 Old Belief Revision Behavior

4.1.1 Manual Revision Example 1

Up until now, belief revision in SNePS was performed manually. A typical interaction might have gone like this:

```
;;; Show supports
: expert

;;; Ask the user what to do when belief revision is triggered
: br-mode manual
Automatic belief revision must now be manually selected.

;;; All pirates are uneducated.
```

```
: all(x)(Pirate(x)=>~Educated(x)).

;;; All criminals are uneducated.
: all(x)(Criminal(x)=>~Educated(x)).

;;; Blackbeard was a pirate and a criminal.
: and{Pirate(Blackbeard),Criminal(Blackbeard)}!

;;; The knowledge base thus far
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}

;;; Blackbeard was educated (triggers belief revision)
Educated(Blackbeard).

  A contradiction was detected within context default-defaultct.
  The contradiction involves the proposition you want to assert:
      wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}

  and the previously existing proposition:
      wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}

  You have the following options:
   1. [a] to attempt to resolve the contradiction automatically
   2. [c] to continue anyway, knowing that a contradiction is derivable;
   3. [r] to revise the inconsistent part of the context manually
   4. [d] to discard this contradictory new assertion from the context
  (please type a, c, r or d)
```

```
=><= r

  In order to make the context consistent you must delete at least
  one hypothesis from each of the following sets of hypotheses:
    (wff6 wff5 wff1)
    (wff6 wff5 wff2)

   The hypotheses listed below are included in more than
   one set. Removing one of these will make more than one
   set consistent.
    (wff6 wff5)

 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 3 :   wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}
          (2 supported propositions: (wff7 wff1) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions
```

```
  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 3

The consistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (3 supported propositions: (wff5 wff4 wff3) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= q

 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (2 supported propositions: (wff8 wff6) )
```

```
 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (5 supported propositions: (wff8 wff7 wff5 wff4 wff3) )

 3 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (2 supported propositions: (wff7 wff2) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 2

The consistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

 2 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (1 supported proposition: (wff2) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
```

```
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= q

  Do you want to add a new hypothesis?  no

;;; The revised knowledge base
list-asserted-wffs
  wff9!:  nand{Criminal(Blackbeard),Pirate(Blackbeard)}  {<ext,{wff2,wff6}>}
  wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
```

We see that the user is given total control of how to manipulate the knowledge base to restore consistency. Notice that in the example above the user was given advice about which propositions to remove to minimize information loss, but that the advice was ignored. Also notice that there was no information whatsoever about the relative entrenchment of the culprit hypotheses.

4.1.2 Manual Revision Example 2

Now see the example below:

```
;;; Show supports
: expert

;;; Ask the user what to do when belief revision is triggered
: br-mode manual
Automatic belief revision must now be manually selected.

;;; All pirates are uneducated.
```

```
: all(x)(Pirate(x)=>~Educated(x)).

;;; All criminals are uneducated.
: all(x)(Criminal(x)=>~Educated(x)).

;;; Blackbeard was a pirate and a criminal.
: and{Pirate(Blackbeard),Criminal(Blackbeard)}!

;;; The knowledge base thus far
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}

;;; Blackbeard was educated (triggers belief revision)
: Educated(Blackbeard).

  A contradiction was detected within context default-defaultct.
  The contradiction involves the proposition you want to assert:
      wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}

  and the previously existing proposition:
      wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}

  You have the following options:
   1. [a] to attempt to resolve the contradiction automatically
   2. [c] to continue anyway, knowing that a contradiction is derivable;
   3. [r] to revise the inconsistent part of the context manually
   4. [d] to discard this contradictory new assertion from the context
  (please type a, c, r or d)
```

```
=><= r

  In order to make the context consistent you must delete at least
  one hypothesis from each of the following sets of hypotheses:
    (wff6 wff5 wff1)
    (wff6 wff5 wff2)

   The hypotheses listed below are included in more than
   one set. Removing one of these will make more than one
   set consistent.
    (wff6 wff5)

 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 3 :   wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}
          (2 supported propositions: (wff7 wff1) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions
```

```
  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 3

The consistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (3 supported propositions: (wff5 wff4 wff3) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= q

 In order to make the context consistent you must delete
  at least one hypothesis from the set listed below.

An inconsistent set of hypotheses:

 1 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (2 supported propositions: (wff8 wff6) )
```

```
 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (5 supported propositions: (wff8 wff7 wff5 wff4 wff3) )

 3 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (2 supported propositions: (wff7 wff2) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 1

The consistent set of hypotheses:

 1 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 2 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (2 supported propositions: (wff7 wff2) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
```

```
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= d

  Enter the list number of a hypothesis to discard,
  [c] to cancel this discard, or [q] to quit revising this set.
=><= 2

The consistent set of hypotheses:

 1 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (3 supported propositions: (wff5 wff4 wff3) )

  Enter the list number of a hypothesis to examine or
  [d] to discard some hypothesis from this list,
  [a] to see ALL the hypotheses in the full context,
  [r] to see what you have already removed,
  [q] to quit revising this set, or
  [i] for instructions

  (please type a number OR d, a, r, q or i)
=><= q

  Do you want to add a new hypothesis?  no

;;; The revised knowledge base
list-asserted-wffs
  wff10!:  nand{Educated(Blackbeard),
                all(x)(Criminal(x) =>  (~Educated(x)))}  {<ext,{wff5}>}
  wff5!:  and{Pirate(Blackbeard),Criminal(Blackbeard)}  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
```

This example is even more problematic. The user removed far more than was necessary in order to resolve the contradiction. Further on that note, the user chose to remove wff6 from the second no-good after removing a proposition from the first no-good. Had the no-goods been presented in the opposite order, and wff6 was removed initially, then it would not have been necessary to even look at the other no-good; both would have been made consistent by the removal of wff6.

4.2 New Belief Revision Behavior

We shall see that the problems with manual belief revision discussed in §4.1 are remedied by the new procedures.

4.2.1 Algorithm 2 Example 1

The example below corresponds to that in §4.1.1, but now automatic belief revision is performed, using Algorithm 2.

```
;;; Show supports
: expert

;;; Use algorithm 2
: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.

;;; Use an entrenchment ordering where every proposition is
;;; minimally-entrenched
: set-order null-order

;;; All pirates are uneducated.
```

```
: all(x)(Pirate(x)=>~Educated(x)).

;;; All criminals are uneducated.
: all(x)(Criminal(x)=>~Educated(x)).

;;; Blackbeard was a pirate and a criminal.
: and{Pirate(Blackbeard),Criminal(Blackbeard)}!

;;; The knowledge base thus far
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}

;;; Blackbeard was educated (triggers belief revision)
Educated(Blackbeard).

  A contradiction was detected within context default-defaultct.
  The contradiction involves the proposition you want to assert:
      wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}

  and the previously existing proposition:
      wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}

  You have the following options:
   1. [a] to attempt to resolve the contradiction automatically
   2. [c] to continue anyway, knowing that a contradiction is derivable;
   3. [r] to revise the inconsistent part of the context manually
   4. [d] to discard this contradictory new assertion from the context
  (please type a, c, r or d)
```

```
=><= a

 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (2 supported propositions: (wff7 wff2) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 3 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

=><= 1

 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}
          (2 supported propositions: (wff7 wff1) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 3 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

=><= 1

;;; The revised knowledge base
list-asserted-wffs
  wff9!:  ~(all(x)(Criminal(x) => (~Educated(x))))  {<ext,{wff5,wff6}>}
```

```
wff8!:  ~(all(x)(Pirate(x) => (~Educated(x))))  {<ext,{wff5,wff6}>}
wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
```

Notice that the user was merely asked which proposition was minimally-entrenched within each no-good, not to actually remove propositions. As a result, SNeBR was able to correctly identify wff1 and wff2 as the culprits, remove them, and assert their negations. The user did not get an opportunity to unnecessarily remove more propositions. Also note that wff6, the last proposition to be asserted, was an option. Therefore this is an example of *nonprioritized* belief revision.

4.2.2 Algorithm 2 Example 2

The following example shows a vast improvement in behavior over the example in §4.1.2, using Algorithm 2.

```
;;; Show supports
: expert

;;; Always attempt automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.

;;; Use algorithm 2
: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.

;;; Use an entrenchment ordering where every proposition is
```

```
;;; minimally-entrenched
: set-order null-order

;;; All pirates are uneducated.
: all(x)(Pirate(x)=>~Educated(x)).

;;; All criminals are uneducated.
: all(x)(Criminal(x)=>~Educated(x)).

;;; Blackbeard was a pirate and a criminal.
: andor{Pirate(Blackbeard),Criminal(Blackbeard)}!

;;; The knowledge base thus far
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}

;;; Blackbeard was educated (triggers belief revision)
: Educated(Blackbeard).

 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
          (2 supported propositions: (wff7 wff2) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )
```

```
 3 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

=><= 1

 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}
          (2 supported propositions: (wff7 wff1) )

 2 :   wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
          (4 supported propositions: (wff7 wff5 wff4 wff3) )

 3 :   wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
          (1 supported proposition: (wff6) )

=><= 3

;;; The revised knowledge base
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}
```

Again the user was only asked to provide enough information necessary to resolve the contradiction, and was not given complete power to alter the knowledge base. More importantly, SNeBR was able to recognize after the *second* query that it was not necessary to remove wff2, the proposition selected as minimal in the first query. This is because wff6, the proposition selected as minimal

in the second query, was present in both no-goods. Also note that wff6, the last proposition to be asserted, was contracted. This is a therefore a prime example of *nonprioritized* belief revision in action.

4.2.3 Algorithm 1 Example

When algorithm 1 is used, the user is not consulted when there is a question about which proposition is uniquely minimally-entrenched in a no-good. Algorithm 1 attempts to figure this out using the supplied ordering (none is provided here), and when a question remains it arbitrarily (but deterministically) chooses a proposition to be minimally-entrenched. In the current implementation, SNePS will use the lexicographic rank of the name of the proposition (wff**1**, wff**2**, etc.) to determine its relative epistemic ordering. Lexicographic comparison yields a well preorder.

```
;;; Show supports
: expert

;;; Always use automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.

;;; Use algorithm 1
: br-tie-mode auto
Entrenchment ties will now be automatically broken

;;; Use an entrenchment ordering where every proposition is
;;; minimally-entrenched
: set-order null-order

;;; All pirates are uneducated.
: all(x)(Pirate(x)=>~Educated(x)).
```

```
;;; All criminals are uneducated.
: all(x)(Criminal(x)=>~Educated(x)).

;;; Blackbeard was a pirate and a criminal.
: and{Pirate(Blackbeard),Criminal(Blackbeard)}!

;;; The knowledge base as thus far
: list-asserted-wffs
  wff7!:  ~Educated(Blackbeard)  {<der,{wff2,wff5}>,<der,{wff1,wff5}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
  wff2!:  all(x)(Criminal(x) => (~Educated(x)))  {<hyp,{wff2}>}
  wff1!:  all(x)(Pirate(x) => (~Educated(x)))  {<hyp,{wff1}>}

;;; Blackbeard was educated (triggers belief revision)
: Educated(Blackbeard).

;;; The revised knowledge base
: list-asserted-wffs
  wff9!:  ~(all(x)(Criminal(x) => (~Educated(x))))  {<ext,{wff5,wff6}>}
  wff8!:  ~(all(x)(Pirate(x) => (~Educated(x))))  {<ext,{wff5,wff6}>}
  wff6!:  Educated(Blackbeard)  {<hyp,{wff6}>}
  wff5!:  Pirate(Blackbeard) and Criminal(Blackbeard)  {<hyp,{wff5}>}
  wff4!:  Criminal(Blackbeard)  {<der,{wff5}>}
  wff3!:  Pirate(Blackbeard)  {<der,{wff5}>}
```

We see that wff1 and wff2, being of minimal lexicographic rank, were also least epistemically entrenched. As a result, they were chosen as culprits to restore consistency to the no-goods in which they appeared. Though it is not necessarily apparent, *nonprioritized* belief revision was performed here as well.

4.2.4 Prioritized Belief Revision Example

In the following example, prioritized belief revision is demonstrated. Algorithm 2 is used.

```
;;; Show supports
: expert

;;; Use mode 3 of SNePSLOG (see manual)
: set-mode-3

;;; Always attempt automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.

;;; Use algorithm 2
: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.

;;; Initialize believe action (see manual)
: ^(attach-primaction believe believe)

;;; Use an order in which every proposition is minimally-entrenched
: set-order null-order

;;; [Walk(prep, obj)] = It walks [prep] a [obj].
: define-frame Walk(verb prep obj)

;;; [Talk(prep, obj]] = It talks [prep] a [obj].
: define-frame Talk(verb prep obj)

;;; We will use x to range over the above frames
: define-frame x(verb prep obj)
```

```
;;; [Isa(obj)] = It is a [obj].
: define-frame Isa(verb obj)

;;; If it [x]s like a [y], it is a [y].
: all(x,y)(x(like,y)=>Isa(y)).

;;; It walks like a duck.
: Walk(like, duck)!

;;; It talks like a duck.
: Talk(like, duck)!

;;; The knowledge base thus far
: list-asserted-wffs
  wff4!:  Talk(like,duck)  {<hyp,{wff4}>}
  wff3!:  Isa(duck)  {<der,{wff1,wff2}>,<der,{wff1,wff4}>}
  wff2!:  Walk(like,duck)  {<hyp,{wff2}>}
  wff1!:  all(y,x)(x(like,y) => Isa(y))  {<hyp,{wff1}>}

;;; It's not a duck (triggers belief revision).
: perform believe(~Isa(duck))

 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff1!:  all(y,x)(x(like,y) => Isa(y))  {<hyp,{wff1}>}
          (2 supported propositions: (wff3 wff1) )

 2 :   wff2!:  Walk(like,duck)  {<hyp,{wff2}>}
          (2 supported propositions: (wff3 wff2) )

=><= 2
```

```
 Please choose from the following hypotheses the one that you would
 least like to keep:

 1 :   wff1!:  all(y,x)(x(like,y) => Isa(y))  {<hyp,{wff1}>}
          (2 supported propositions: (wff3 wff1) )

 2 :   wff4!:  Talk(like,duck)  {<hyp,{wff4}>}
          (2 supported propositions: (wff4 wff3) )

=><= 2

;;; The revised knowledge base
: list-asserted-wffs
  wff8!:  ~Walk(like,duck)  {<ext,{wff1,wff5}>}
  wff7!:  ~Talk(like,duck)  {<ext,{wff1,wff5}>}
  wff5!:  ~Isa(duck)  {<hyp,{wff5}>}
  wff1!:  all(y,x)(x(like,y) => Isa(y))  {<hyp,{wff1}>}
```

Note that the final assertion was made by performing a *believe* act. This invoked prioritized belief revision, causing the statement that was just believed to be strictly more entrenched than every other proposition. This is why wff5, the assertion that it's not a duck, did not even appear as an option during queries to the user; SNeBR knew it could not be minimally-entrenched in any no-good. As a result, it was guaranteed to remain asserted at the conclusion of the revision process.

4.3 Capturing Old Results

A significant feature of my work is that it generalizes previous published work on belief revision in SNePS (Cravo and Martins, 1993; Johnson and Shapiro, 1999; Shapiro and Johnson, 2000; Shapiro

and Kandefer, 2005). The following demonstrations showcase the new features I have introduced to SNeBR, and capture the essence of belief revision as seen in the papers mentioned above by using well-specified epistemic ordering functions. The demos have been edited for formatting and clarity. The commands *br-tie-mode auto* and *br-tie-mode manual* indicate that Algorithm 1 and Algorithm 2 should be used respectively. A *wff* is a well-formed formula. A wff followed by a period (.) indicates that the wff should be asserted, i.e. added to the knowledge base. A wff followed by an exclamation point (!) indicates that the wff should be asserted, and that forward inference should be performed on it.

4.3.1 SNePSwD

I present a demonstration on how the automated belief revision behavior of (Cravo and Martins, 1993) can easily be duplicated by my new system. (Cravo and Martins, 1993) uses a predetermined manual ordering of propositions to do belief revision. The ordering was constructed using a separate command that recorded metainformation about existing propositions outside of the object language of SNePS. That behavior is replicated by the ordering function *explicit*.

The following demo is an adaptation of the automated belief revision demo from (Cravo and Martins, 1993). The descriptions are by and large taken from that article. Formulae stating entrenchment orderings have been omitted from the output for clarity.

```
;;; Show supports
: expert

;;; Always use automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.
```

```
;;; Use algorithm 2
: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.

;;; Use an entrenchment ordering where relative entrenchment is manually
;;; specified
: set-order explicit

;;; There are four kinds of meetings: official meetings, classes, work
;;; meetings, and social meetings. Any meeting will be of exactly one of
;;; these types.
: all(m)(meeting(m) => xor{official(m),class(m),work(m),social(m)}).

;;; Philip, Peter, and John will attend work meetings.
: all(m)(work(m) => and{attends(m,Philip),attends(m,Peter),attends(m,John)}).

;;; Any meeting will be either in the morning or the afternoon, but not both.
: all(m)(meeting(m) => xor{in-morning(m), in-afternoon(m)}).

;;; If the same person attends two different meetings, then one of the
;;; meetings has to be in the morning, and the other in the afternoon.
: all(m1,m2,p)({attends(m1,p),attends(m2,p)} &=>
                (in-morning(m1) <=> in-afternoon(m2))).

;;; Peter prefers meetings in the morning.
: all(m)(attends(m,Peter) => in-morning(m)).

;;; John prefers meetings in the afternoon.
: all(m)(attends(m,John) => in-afternoon(m)).

;;; Philip prefers meetings in the morning.
: all(m)(attends(m,Philip) => in-morning(m)).
```

```
;;; w is a meeting.
: meeting(w).

;;; w is a work meeting.
: work(w).

;;; Philip's preference is less important than John's.
: IsLessEntrenched(wff7,wff6).

;;; John's preference is less important than Peter's.
: IsLessEntrenched(wff6,wff5).

;;; The preferences of people are less important than the remaining
;;; information in the problem.
: IsLessEntrenched(wff5,wff1).
: IsLessEntrenched(wff5,wff2).
: IsLessEntrenched(wff5,wff3).
: IsLessEntrenched(wff5,wff4).
: IsLessEntrenched(wff5,wff8).
: IsLessEntrenched(wff5,wff9).

;;; IsLessEntrenched is a transitive relation
: all(x,y,z)({IsLessEntrenched(x,y),IsLessEntrenched(y,z)} &=>
              IsLessEntrenched(x,z)).

;;; The knowledge base thus far
list-asserted-wffs
  wff18!:  all(z,y,x)({IsLessEntrenched(y,z),IsLessEntrenched(x,y)} &=>
               {IsLessEntrenched(x,z)})  {<hyp,{wff18}>}
  wff9!:  work(w)  {<hyp,{wff9}>}
  wff8!:  meeting(w)  {<hyp,{wff8}>}
  wff7!:  all(m)(attends(m,Philip) => in-morning(m))  {<hyp,{wff7}>}
  wff6!:  all(m)(attends(m,John) => in-afternoon(m))  {<hyp,{wff6}>}
```

```
  wff5!:  all(m)(attends(m,Peter) => in-morning(m))  {<hyp,{wff5}>}
  wff4!:  all(p,m2,m1)({attends(m2,p),attends(m1,p)} &=>
                  {in-afternoon(m2) <=> in-morning(m1)})  {<hyp,{wff4}>}
  wff3!:  all(m)(meeting(m) => (xor{in-afternoon(m),in-morning(m)}))
         {<hyp,{wff3}>}
  wff2!:  all(m)(work(m) => (attends(m,John) and attends(m,Peter) and
                             attends(m,Philip)))  {<hyp,{wff2}>}
  wff1!:  all(m)(meeting(m) => xor{social(m),work(m),class(m),official(m)}))
          {<hyp,{wff1}>}

;;; When will the meeting w take place?
?when(w)?

;;; The revised knowledge base
list-asserted-wffs
  wff93!:  ~(all(m)(attends(m,John) => in-afternoon(m)))
           {<ext,{wff2,wff3,wff5,wff8,wff9}>}
  wff38!:  attends(w)  {<der,{wff2,wff9}>}
  wff37!:  attends(w,John) and attends(w,Peter) and attends(w,Philip)}
           {<der,{wff2,wff9}>}
  wff36!:  xor{social(w),class(w),official(w),work(w)} {<der,{wff1,wff8}>}
  wff35!:  xor{in-afternoon(w),in-morning(w)}  {<der,{wff3,wff8}>}
  wff31!:  in-morning(w)  {<der,{wff2,wff5,wff9}>,<der,{wff2,wff7,wff9}>}
  wff29!:  attends(w,John)  {<der,{wff2,wff9}>}
  wff27!:  attends(w,Peter)  {<der,{wff2,wff9}>}
  wff25!:  attends(w,Philip)  {<der,{wff2,wff9}>}
  wff24!:  ~social(w)  {<der,{wff1,wff8,wff9}>}
  wff22!:  ~class(w)  {<der,{wff1,wff8,wff9}>}
  wff20!:  ~official(w)  {<der,{wff1,wff8,wff9}>}
  wff18!:  all(z,y,x)({IsLessEntrenched(y,z),IsLessEntrenched(x,y)} &=>
                  {IsLessEntrenched(x,z)})  {<hyp,{wff18}>}
  wff9!:  work(w)  {<hyp,{wff9}>}
  wff8!:  meeting(w)  {<hyp,{wff8}>}
```

```
wff5!:  all(m)(attends(m,Peter) => in-morning(m))  {<hyp,{wff5}>}
wff4!:  all(p,m2,m1)({attends(m2,p),attends(m1,p)} &=>
                {in-afternoon(m2) <=> in-morning(m1)})  {<hyp,{wff4}>}
wff3!:  all(m)(meeting(m) =>
                   xor{in-afternoon(m),in-morning(m)}))  {<hyp,{wff3}>}
wff2!:  all(m)(work(m) => (attends(m,John) and attends(m,Peter) and
                           attends(m,Philip)})  {<hyp,{wff2}>}
wff1!:  all(m)(meeting(m) => (xor{social(m),work(m),class(m), official(m)}))
        {<hyp,{wff1}>}
```

As in (Cravo and Martins, 1993), we see that the meeting takes place in the morning, according to Peter's preference. Here, unlike in (Cravo and Martins, 1993), the information about entrenchment orderings is contained in propositions in the object language of SNePS, namely propositions of the form *IsLessEntrenched(...,...)*. This was done to take advantage of SNePS's reasoning capabilities when determining orderings.

4.3.2 Says Who?

I present a demonstration on how the source-credibility-based revision behavior from (Johnson and Shapiro, 1999) is generalized by my changes to SNeBR. In the following example, the command *set-order source* sets the epistemic ordering used by SNeBR to be a lisp function that compares two propositions based on the relative credibility of their sources. Unsourced propositions are assumed to have maximal credibility. The sources, as well as their relative credibility, are represented as meta-knowledge in the SNePS knowledge base. This was also done in (Johnson and Shapiro, 1999) and (Shapiro and Johnson, 2000). As with *explicit*, the *source* function makes SNePSLOG queries to determine sources of propositions and credibility of sources, using the *askwh* and *ask* commands (Shapiro and The SNePS Implementation Group, 2010) seen in quotes. This allows it to perform inference in making these determinations.

Here we see that the nerd and the sexist make the generalizations that all jocks are not smart and all females are not smart respectively, while the holy book and the professor state that all old people are smart, and all grad students are smart respectively. Since Fran is an old female jock graduate student, there are two sources that would claim she is smart, and two that would claim she is not. However, the sources claiming she is smart are more credible than those claiming otherwise. So the generalizations about jocks and females are discarded. In fact, their negations are asserted, since *belief revision in SNePS provides a mechanism for negation introduction.*

```
;;; Show supports
: expert

: br-mode auto
Automatic belief revision will now be automatically selected.

: br-tie-mode manual
The user will be consulted when an entrenchment tie occurs.

;;; Use source credibilities as epistemic ordering criteria.
set-order source

;;; The holy book is a better source than the professor.
IsBetterSource(holybook, prof).

;;; The professor is a better source than the nerd.
IsBetterSource(prof, nerd).

;;; The nerd is a better source than the sexist.
IsBetterSource(nerd, sexist).

;;; Fran is a better source than the nerd.
IsBetterSource(fran, nerd).
```

```
;;; Better-Source is a transitive relation
all(x,y,z)({IsBetterSource(x,y), IsBetterSource(y,z)} &=>
               IsBetterSource(x,z))!

;;; All jocks are not smart.
all(x)(jock(x)=>~smart(x)). ;wff10

;;; The source of the statement 'All jocks are not smart' is the
;;; nerd.
HasSource(wff10, nerd).

;;; All females are not smart.
all(x)(female(x)=>~smart(x)). ;wff12

;;; The source of the statement 'All females are not smart' is the
;;; sexist.
HasSource(wff12, sexist).

;;; All graduate students are smart.
all(x)(grad(x)=>smart(x)). ;wff14

;;; The source of the statement 'All graduate students are smart'
;;; is the professor.
HasSource(wff14, prof).

;;; All old people are smart.
all(x)(old(x)=>smart(x)). ;wff16

;;; The source of the statement 'All old people are smart' is the
;;; holy book.
HasSource(wff16, holybook).
```

```
;;; The source of the statement 'Fran is an old female jock who is
;;; a graduate student' is fran.
HasSource(and{jock(fran),grad(fran),female(fran),old(fran)},fran).

;;; The KB thus far
list-asserted-wffs
wff23!: HasSource(old(fran) and female(fran) and grad(fran) and
        jock(fran),fran)  {<hyp,{wff23}>}
wff17!: HasSource(all(x)(old(x) => smart(x)),holybook) {<hyp,{wff17}>}
wff16!: all(x)(old(x) => smart(x))  {<hyp,{wff16}>}
wff15!: HasSource(all(x)(grad(x) => smart(x)),prof)  {<hyp,{wff15}>}
wff14!: all(x)(grad(x) => smart(x))  {<hyp,{wff14}>}
wff13!: HasSource(all(x)(female(x) => (~smart(x))),sexist)
        {<hyp,{wff13}>}
wff12!: all(x)(female(x) => (~smart(x)))  {<hyp,{wff12}>}
wff11!: HasSource(all(x)(jock(x) => (~smart(x))),nerd)
        {<hyp,{wff11}>}
wff10!: all(x)(jock(x) => (~smart(x)))  {<hyp,{wff10}>}
wff9!:  IsBetterSource(fran,sexist)  {<der,{wff3,wff4,wff5}>}
wff8!:  IsBetterSource(prof,sexist)  {<der,{wff2,wff3,wff5}>}
wff7!:  IsBetterSource(holybook,sexist)  {<der,{wff1,wff2,wff3,wff5}>}
wff6!:  IsBetterSource(holybook,nerd)  {<der,{wff1,wff2,wff5}>}
wff5!:  all(z,y,x)({IsBetterSource(y,z),IsBetterSource(x,y)} &=>
            {IsBetterSource(x,z)})  {<hyp,{wff5}>}
wff4!:  IsBetterSource(fran,nerd)  {<hyp,{wff4}>}
wff3!:  IsBetterSource(nerd,sexist)  {<hyp,{wff3}>}
wff2!:  IsBetterSource(prof,nerd)  {<hyp,{wff2}>}
wff1!:  IsBetterSource(holybook,prof)  {<hyp,{wff1}>}

;;; Fran is an old female jock who is a graduate student (asserted
;;; with forward inference).
and{jock(fran),grad(fran),female(fran),old(fran)}!
wff50!: ~(all(x)(jock(x) => (~smart(x))))  {<ext,{wff16,wff22}>,
```

```
        <ext,{wff14,wff22}>}
wff24!: smart(fran)  {<der,{wff16,wff22}>,<der,{wff14,wff22}>}

;;; The resulting knowledge base
list-asserted-wffs
wff50!: ~(all(x)(jock(x) => (~smart(x))))  {<ext,{wff16,wff22}>,
        <ext,{wff14,wff22}>}
wff37!: ~(all(x)(female(x) => (~smart(x))))  {<ext,{wff16,wff22}>}
wff24!: smart(fran)  {<der,{wff16,wff22}>,<der,{wff14,wff22}>}
wff23!: HasSource(old(fran) and female(fran) and grad(fran) and
        jock(fran),fran)  {<hyp,{wff23}>}
wff22!: old(fran) and female(fran) and grad(fran) and jock(fran)
        {<hyp,{wff22}>}
wff21!: old(fran)  {<der,{wff22}>}
wff20!: female(fran)  {<der,{wff22}>}
wff19!: grad(fran)  {<der,{wff22}>}
wff18!: jock(fran)  {<der,{wff22}>}
wff17!: HasSource(all(x)(old(x) => smart(x)),holybook) {<hyp,{wff17}>}
wff16!: all(x)(old(x) => smart(x))  {<hyp,{wff16}>}
wff15!: HasSource(all(x)(grad(x) => smart(x)),prof)  {<hyp,{wff15}>}
wff14!: all(x)(grad(x) => smart(x))  {<hyp,{wff14}>}
wff13!: HasSource(all(x)(female(x) => (~smart(x))),sexist)
        {<hyp,{wff13}>}
wff11!: HasSource(all(x)(jock(x) => (~smart(x))),nerd) {<hyp,{wff11}>}
wff9!:  IsBetterSource(fran,sexist)  {<der,{wff3,wff4,wff5}>}
wff8!:  IsBetterSource(prof,sexist)  {<der,{wff2,wff3,wff5}>}
wff7!:  IsBetterSource(holybook,sexist)  {<der,{wff1,wff2,wff3,wff5}>}
wff6!:  IsBetterSource(holybook,nerd)  {<der,{wff1,wff2,wff5}>}
wff5!:  all(z,y,x)({IsBetterSource(y,z),IsBetterSource(x,y)} &=>
            {IsBetterSource(x,z)})  {<hyp,{wff5}>}
wff4!:  IsBetterSource(fran,nerd)  {<hyp,{wff4}>}
wff3!:  IsBetterSource(nerd,sexist)  {<hyp,{wff3}>}
wff2!:  IsBetterSource(prof,nerd)  {<hyp,{wff2}>}
```

```
wff1!:  IsBetterSource(holybook,prof)  {<hyp,{wff1}>}
```

We see that the statements that all jocks are not smart and that all females are not smart are no longer asserted at the end. These statements supported the statement that Fran *is not* smart. The statements that all old people are smart and that all grad students are smart supported the statement that Fran *is* smart. The contradiction was resolved by contracting "Fran *is not* smart," since the sources for its supports were less credible than the sources for "Fran *is* smart."

4.3.3 Wumpus World

I present a demonstration on how the state-constraint-based revision behavior from (Shapiro and Kandefer, 2005) is generalized by my changes to SNeBR. The command *set-order fluent* says that propositional fluents are strictly less entrenched than non-propositional fluents. The *fluent* order was created specifically to replace the original belief revision behavior of the SNeRE *believe* act. In the version of SNeBR used in (Shapiro and Kandefer, 2005), propositions of the form $andor(<0|1>,1)(p_1,p_2,\ldots)$ were assumed to be state contraints, while the inner propositions, p_1, p_2, etc., were assumed to be fluents. The fluents were less entrenched than the state constraints. We see that the ordering was heavily syntax-dependent.

In my new version, the determination of which propositions are fluents is made by checking for membership of the predicate symbol of an atomic proposition in a list called `*fluents*`, which is defined by the user to include the predicate symbols of all propositional fluents. So the entrenchment ordering defined here uses metaknowledge about the knowledge base that is not represented in the SNePS knowledge base. The command *br-tie-mode manual* indicates that Algorithm 2 should be used. Note that the *xor* connective (Shapiro, 2010) used below replaces instances of *andor(1,1)(...)* from (Shapiro and Kandefer, 2005). The command `perform believe(wff)` is identical to the command `wff!`, except that the former causes `wff` to be strictly more entrenched

than every other proposition during belief revision. That is, `wff` is guaranteed to be *safe* (unless `wff` is itself a contradiction). So we would be using *prioritized* belief revision.

```
;;; Show supports
: expert

;;; Always use automatic belief revision
: br-mode auto
Automatic belief revision will now be automatically selected.

;;; Use algorithm 2
: br-tie-mode manual
Entrenchment ties will now be automatically broken
The user will be consulted when an entrenchment tie occurs.
2;;; [Facing(x)] = The agent is facing direction [x].
define-frame Facing(nil onfloor)

;;; Use an entrenchment ordering that favors non-fluents over
;;; fluents
set-order fluent

;;; Establish what kinds of propositions are fluents, specifically:
;;;   - That the agent is facing some direction is a fact that may
;;;     change over time.
^(setf *fluents* '(Facing))

;;; The agent is Facing west
Facing(west).

;;; At any given time, the agent is facing either north, south,
;;; east, or west (asserted with forward inference).
xor{Facing(north),Facing(south),Facing(east), Facing(west)}!
```

```
;;; The knowledge base as it stands
list-asserted-wffs
wff8!:  ~Facing(north)  {<der,{wff1,wff5}>}
wff7!:  ~Facing(south)  {<der,{wff1,wff5}>}
wff6!:  ~Facing(east)  {<der,{wff1,wff5}>}
wff5!:  xor{Facing(east),Facing(south),Facing(north), Facing(west)}
        {<hyp,{wff5}>}
wff1!:  Facing(west)  {<hyp,{wff1}>}

;;; Tell the agent to believe it is now facing east.
perform believe(Facing(east))

;;; The resulting knowledge base
list-asserted-wffs
wff10!:  ~Facing(west)  {<ext,{wff4,wff5}>}
wff8!:  ~Facing(north)  {<der,{wff1,wff5}>,<der,{wff4,wff5}>}
wff7!:  ~Facing(south)  {<der,{wff1,wff5}>,<der,{wff4,wff5}>}
wff5!:  xor{Facing(east),Facing(south),Facing(north),Facing(west)}
        {<hyp,{wff5}>}
wff4!:  Facing(east)  {<hyp,{wff4}>}
```

There are three propositions in the no-good when revision is performed: `Facing(west)`, `Facing,east`, and `xor(1,1)`{`Facing(...`}. `Facing(east)` is not considered for removal since it was prioritized by the believe action. The state-constraint `xor(1,1)`{`Facing...`} remains in the knowledge base at the end, because it is more entrenched than `Facing(west)`, a propositional fluent, which is ultimately removed.

5 Analysis of Algorithm 1

5.1 Proofs of Satisfaction of Requirements by Algorithm 1

I show that Algorithm 1 satisfies the requirements established in section 2:

5.1.1 $EE_{SNePS}1$ (Sufficiency)

During each iteration of *AddLoop* an element τ is added to T from some $\sigma \in \Sigma$. Then each set $\sigma \in \Sigma$ containing τ is removed from Σ. The process is repeated until Σ is empty. Therefore each removed set σ in Σ contains some τ in T (Note that each σ will be removed from Σ by the end of the process). So $\forall\sigma[\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T \cap \sigma)]$. Q.E.D.

5.1.2 $EE_{SNePS}2$ (Minimal Entrenchment)

From lines 8-9, we see that T is comprised solely of first elements of sets in Σ. And from lines 2-4, we see that those first elements are all minimal under $\leqq$ relative to the other elements in each set. Since $\forall e_1, e_2, \leqslant [e_1 \leqq e_2 \rightarrow e_1 \leqslant e_2]$, those first elements are minimal under $\leqslant$ as well. That is, $\forall\tau[\tau \in T \rightarrow \exists\sigma[\sigma \in \Sigma \wedge \tau \in \sigma \wedge \forall w[w \in \sigma \rightarrow \tau \leqslant w]]]$. Q.E.D.

5.1.3 $EE_{SNePS}3$ (Information Preservation)

From the previous proof we see that during each iteration of *AddLoop*, we are guaranteed that at least one set σ containing the current culprit is removed from Σ. And we know that the current culprit for that iteration is minimally-entrenched in σ. We also know from $(EE_{SNePS}2)$ that

each subsequently chosen culprit will be minimally entrenched in some set. From lines 2-5 and *AddLoop*, we know that subsequently chosen culprits will be less entrenched than the current culprit. From lines 2-5, we also see that all the other elements in σ have higher entrenchment than the current culprit. Therefore subsequent culprits cannot be elements in σ. So, they cannot be used to eliminate σ. Obviously, previous culprits were also not members of σ. Therefore, if we exclude the current culprit from T, then there will be a set in Σ that does not contain any element of T. That is,

$\forall T'[T' \subset T \rightarrow \exists\sigma[\sigma \in \Sigma \wedge \neg\exists\tau[\tau \in (T' \cap \sigma)]]]$

$\therefore \forall T'[T' \subset T \rightarrow \exists\sigma[\neg\neg(\sigma \in \Sigma \wedge \neg\exists\tau[\tau \in (T' \cap \sigma))]]]$

$\therefore \forall T'[T' \subset T \rightarrow \exists\sigma[\neg(\neg(\sigma \in \Sigma) \vee \exists\tau[\tau \in (T' \cap \sigma))]]]$

$\therefore \forall T'[T' \subset T \rightarrow \exists\sigma[\neg(\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T' \cap \sigma))]]]$

$\therefore \forall T'[T' \subset T \rightarrow \neg\forall\sigma[\sigma \in \Sigma \rightarrow \exists\tau[\tau \in (T' \cap \sigma)]]]$ Q.E.D.

5.1.4 Decidability

We see that *DeleteLoop* is executed once for each element in Σ, which is a finite set. So it always terminates. We see that *AddLoop* terminates when Σ is empty. And from lines 8 and 13 we see that at least one set is removed from Σ during each iteration of *AddLoop*. So *AddLoop* always terminates. Lines 2-4 involve finding a minimum element, which is a decision procedure. Line 5 performs sorting, which is also a decision procedure. Since every portion of Algorithm 1 always terminates, it is a decision procedure. Q.E.D.

5.1.5 Supplementary Requirement

Algorithm 1 is a fully-automated procedure that makes no queries of the user. Q.E.D.

5.2 Complexity of Algorithm 1

5.2.1 Space Complexity

Algorithm 1 can be run completely in-place, i.e. it can use only the memory allocated to the input, with the exception of the production of the set of culprits T. Let us assume that the space needed to store a single proposition is $O(1)$ memory units. Since we only need to remove one proposition from each no-good to restore consistency, algorithm 1 uses $O(|\Sigma|)$ memory units.

5.2.2 Time Complexity

The analysis for time complexity is based on a sequential-procesing system. Let us assume that we implement lists as array structures. Let us assume that we may determine the size of an array in $O(1)$ time. Let us also assume that performing a comparison using $\leqslant$ takes $O(1)$ time. Then in lines 2-4, for each array $\sigma \in \Sigma$ we find the minimum element σ and perform a swap on two elements at most once for each element in σ. If we let s_{max} be the cardinality of the largest σ in Σ, then lines 2-4 will take $O(|\Sigma| \cdot s_{max})$ time. In line 5, we sort the no-goods' positions in Σ using their first elements as keys. This takes $O(|\Sigma| \cdot log(|\Sigma|))$ time. Lines 7-16 iterate through the elements of Σ at most once for each element in Σ. During each such iteration, a search is performed for an element within a no-good. Also, during each iteration through all the no-goods, at least one σ is removed, though this does not help asymptotically. Since the no-goods are not sorted, the search takes linear time in s_{max}. So lines 7-16 take $O(|\Sigma|^2 \cdot s_{max})$ time. Therefore, the running time is $O(|\Sigma|^2 \cdot s_{max})$ time.

Note that the situation changes slightly if we sort the no-goods instead of just placing the minimally-entrenched proposition at the front, as in lines 2-4. In this case, each search through a no-good will take $O(log(s_{max}))$ time, yielding a new total time of $O(|\Sigma| \cdot s_{max} \cdot log(s_{max}) + |\Sigma|^2 \cdot$

$log(s_{max})$).

6 Analysis of Algorithm 2

6.1 Proofs of Satisfaction of Requirements by Algorithm 2

I show that Algorithm 2 satisfies the requirements established in section 2:

6.1.1 $EE_{SNePS}1$ (Sufficiency)

Since every set of propositions must contain at least one proposition that is minimally entrenched, at least one proposition is added to the list in each iteration of *ListLoop*. In the worst case, assume that for each iteration of *MainLoop*, only either *RemoveLoop* or *ModifyLoop* do any work. We know that at least this much work is done for the following reasons: if *ModifyLoop* cannot operate on any no-good during an iteration of *MainLoop*, then all no-goods have only one minimally-entrenched proposition. So either *RemoveLoop*'s condition at line 10 would hold, or:

1. A no-good has multiple minimally-entrenched propositions, causing *ModifyLoop* to do work. This contradicts our assumption that *ModifyLoop* could not do any work during this iteration of *MainLoop*, so we set this possibility aside.

2. Some proposition p_1 is a non-minimally-entrenched proposition in some no-good σ_n, and a minimally-entrenched one in another no-good σ_m. In this case, either p_1 is removed during the iteration of *RemoveLoop* where σ_m is considered, or there is another proposition p_2 in σ_m that is not minimally-entrenched in σ_m, but is in $\sigma_{m'}$. This chaining must eventually terminate at a no-good $\sigma_{m_{final}}$ since $\leqslant$ is transitive and Σ is finite. And the final proposition in the chain p_{final} must be the sole minimally-entrenched proposition in σ_{final}, since otherwise *ModifyLoop* would

have been able to do work for this iteration of *MainLoop*, which is a contradiction. *ModifyLoop* can only do work once for each no-good, so eventually its work is finished. If *ModifyLoop* has no more work left to do, then *RemoveLoop* must do work at least once for each iteration of *MainLoop*. And in doing so, it will create a list of culprits of which each no-good contains at least one. Q.E.D.

6.1.2 $EE_{SNePS}2$ (Minimal Entrenchment)

Since propositions are only added to T when the condition in line 10 is satisfied, it is guaranteed that every proposition in T is a minimally-entrenched proposition in some no-good σ.

6.1.3 $EE_{SNePS}3$ (Information Preservation)

From line 10, we see that when a proposition p is removed, none of the other propositions in its no-good are minimally-entrenched in any other no-good. That means none of the other propositions could be a candidate for removal. So, the only way to remove the no-good in which p appears is by removing p. So if p were not removed, then $(EE_{SNePS}1)$ would not be satisfied. Q.E.D.

6.1.4 Decidability

ListLoop creates lists of minimal elements of lists. This is a decision procedure since the comparator is a total preorder. From the proof of $(EE_{SNePS}1)$ above, we see that either *RemoveLoop* or *ModifyLoop* must do work for each iteration of *MainLoop*. *ModifyLoop* cannot operate more than once on the same no-good, because there are no longer multiple minimally-entrenched propositions in the no-good after it does its work. Nor can *RemoveLoop* operate twice on the same no-good, since the no-good is removed when *ModifyLoop* does work. So, eventually *ModifyLoop* has no more work to do, and at that point *RemoveLoop* will remove at least one no-good for each iteration

of *MainLoop*. By lines 17-18, when the last no-good is removed, the procedure terminates. So it always terminates. Q.E.D.

6.1.5 Supplementary Requirement

RemoveLoop attempts to compute T each time it is run from *MainLoop*. If the procedure does not terminate within *RemoveLoop*, then we run *ModifyLoop* on *at most one* no-good. Afterwards, we run *RemoveLoop* again. Since the user is only queried when the procedure cannot automatically determine any propositions to remove, we argue that this means minimal queries are made of the user. Q.E.D.

6.2 Complexity of Algorithm 2

6.2.1 Space Complexity

As before, let s_{max} be the cardinality of the largest no-good in Σ. In the worst case all propositions are minimally entrenched, so *ListLoop* will recreate Σ. So *ListLoop* will use $O(|\Sigma| \cdot s_{max})$ space. RemoveLoop creates a culprit list, which we stated before takes $O(|\Sigma|)$ space. ModifyLoop may be implemented in a variety of ways. We will assume that it creates a list of pairs, of which the first and second elements range over propositions in the no-goods. In this case *ModifyLoop* uses $O(|\Sigma|^2 \cdot s_{max}^2)$ space. So the total space requirement is $O(|\Sigma|^2 \cdot s_{max}^2)$ memory units.

6.2.2 Time Complexity

The analysis for time complexity is based on a sequential-procesing system. For each no-good σ, in the worst case, *ListLoop* will have to compare each proposition in σ agains every other. So, for each iteration of *MainLoop*, *ListLoop* takes $O(|\Sigma| \cdot s_{max}^2)$ time. There are at most $O(s_{max})$ elements in each list created by *ListLoop*. So, checking the condition in line 10 takes $O(|\Sigma| \cdot s_{max}^2)$ time. Lines 12-16 can be executed in $O(|\Sigma| \cdot s_{max})$ time. Therefore, *RemoveLoop* takes $O(|\Sigma| \cdot s_{max}^2)$ time. We assume that all the work in lines 24-27 can be done in constant time. So, *ModifyLoop* takes $O(|\Sigma|)$ time. We noted earlier that during each iteration of *MainLoop*, *RemoveLoop* or *ModifyLoop* will do work. In the worst case, only one will do work each time. And they each may do work at most $|\Sigma|$ times. So the total running time for the procedure is $O(|\Sigma|^2 \cdot s_{max}^2)$.

7 Conclusions

My modified version of SNeBR provides decision procedures for belief revision in SNePS. By providing a single resulting knowledge base, these procedures essentially perform maxichoice revision for SNePS. These procedures work equally well for both prioritized and nonprioritized belief revision, with subtle changes to the epistemic ordering required to perform the latter. Using a well preorder, belief revision can be performed completely automatically. Given a total preorder, it may be necessary to consult the user in order to simulate a well preorder. The simulated well preorder need only be partially specified; it is only necessary to query the user when multiple beliefs are minimally-epistemically-entrenched within a no-good, and even then only in the case where no other belief in the no-good is already being removed. In any event, the epistemic ordering itself is *user-supplied*. My algorithm for revision given a well preorder uses asymptotically less time and space than the other algorithm, which uses a total preorder.

References

Alchourron, C. E., Gärdenfors, P., and Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic*, 20:510–530.

Alchourron, C. E. and Makinson, D. (1982). On the logic of theory change: Contraction functions and their associated revision functions. *Theoria*, 48:14–37.

Alchourron, C. E. and Makinson, D. (1985). On the logic of theory change: Safe contraction. *Studia Logica*, (44):405–422.

Cravo, M. R. and Martins, J. P. (1993). SNePSwD: A newcomer to the SNePS family. *Journal of Experimental and Theoretical Artificial Intelligence*, 5(2–3):135–148.

de Kleer, J. (1986). An assumption-based TMS. *Artificial Intelligence*, 28:127–162.

Dixon, S. (1993). A finite base belief revision system. In *Proceedings of ACSC-16: 16th Australian Computer Science Conference*, volume 15 of *Australian Computer Science Communications*, pages 445–451. Queensland University of Technology, Brisbane, Australia.

Gärdenfors, P. (1982). Rules for rational changes of belief. In Pauli, T., editor, *Philosophical Essays Dedicated to Lennart Åqvist on His Fiftieth Birthday*, number 34 in Philosophical Studies, pages 88–101, Uppsala, Sweden. The Philosophical Society and the Department of Philosophy, University at Uppsala.

Gärdenfors, P. (1988). *Knowledge in Flux: Modeling the Dynamics of Epistemic States*. The MIT Press, Cambridge, Massachusetts.

Gärdenfors, P. and Rott, H. (1995). Belief revision. In Gabbay, Hogger, and Robinson, editors, *Epistemic and Temporal Reasoning*, volume 4 of *Handbook of Logic in Artificial Intelligence and Logic Programming*, pages 35–131. Clarendon Press, Oxford.

Hansson, S. O. (1994). Kernel contraction. *The Journal of Symbolic Logic*, 59(3):845–859.

Hansson, S. O. (1997). Semi-revision. *Journal of Applied Non-Classical Logics*, 7(2):151–175.

Hansson, S. O. (1999a). A survey of non-prioritized belief revision. *Erkenntnis*, 50:413–427.

Hansson, S. O. (1999b). *A Textbook of Belief Dynamics: Theory Change and Database Updating*. Kluwer Academic Publishers, Dordrecht; Boston.

Johnson, F. L. (2006). *Dependency-directed reconsideration: an anytime algorithm for hindsight knowledge-base optimization*. PhD thesis, State University of New York at Buffalo, Buffalo, NY, USA.

Johnson, F. L. and Shapiro, S. C. (1999). Says Who? - Incorporating Source Credibility Issues into Belief Revision. Technical Report 99-08, Department of Computer Science and Engineering and Center for Multisource Information Fusion and Center for Cognitive Science, State University of New York at Buffalo, Buffalo, NY.

Lakemeyer (1991). On the relation between explicit and implicit beliefs. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, pages 368–375. Morgan Kaufmann.

Levi, I. (1977). Subjunctives, dispositions and changes. *Synthese*, 34:423–455.

Makinson, D. (1987). On the status of the postulate of recovery in the logic of theory change. *Journal of Philosophical Logic*, 16:383–394.

Martins, J. P. and Shapiro, S. C. (1988). A model for belief revision. *Artificial Intelligence*, 35(1):25–79.

Shapiro, S. C. (1992). Relevance logic in computer science. Section 83 of Alan Ross Anderson and Nuel D. Belnap, Jr. and J. Michael Dunn *et al. Entailment, Volume II*, pages 553–563. Princeton University Press, Princeton, NJ.

Shapiro, S. C. (2010). Set-oriented logical connectives: Syntax and semantics. In Lin, F., Sattler, U., and Truszczynski, M., editors, *Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning (KR2010)*, pages 593–595. AAAI Press.

Shapiro, S. C. and Johnson, F. L. (2000). Automatic belief revision in SNePS. In Baral, C. and Truszczynski, M., editors, *Proceedings of the 8th International Workshop on Non-monotonic Reasoning NMR2000*. unpaginated, 5 pages.

Shapiro, S. C. and Kandefer, M. (2005). A SNePS Approach to the Wumpus World Agent or Cassie Meets the Wumpus. In Morgenstern, L. and Pagnucco, M., editors, *IJCAI-05 Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC05): Working Notes*, pages 96–103.

Shapiro, S. C. and The SNePS Implementation Group (2010). *SNePS 2.7.1 USER'S MANUAL*. Department of Computer Science and Engineering, University at Buffalo, The State University of New York, 201 Bell Hall, Buffalo, NY 14260-2000.

Williams, M.-A. (1994). On the logic of theory base change. In MacNish, C., Pearce, D., and Pereira, L., editors, *Logics in Artificial Intelligence*, volume 838 of *Lecture Notes in Computer Science*, pages 86–105. Springer Berlin / Heidelberg.