An Approach to Revising Logic Programs under the Answer Set Semantics

James P. Delgrande
School of Computing Science
Simon Fraser University
Burnaby, B.C.
Canada V5A 1S6
jim@cs.sfu.ca

Abstract

An approach to the *revision* of logic programs under the answer set semantics is presented. For programs P and Q, the goal is to determine the answer sets that correspond to the revision of P by Q, denoted $P * Q$. A fundamental principle of classical (AGM) revision, and the one that guides the approach here, is the *success postulate*. In AGM revision, this stipulates that $\alpha \in K * \alpha$. By analogy with the success postulate, for programs P and Q, this means that the answer sets of Q will in some sense be contained in those of $P * Q$. The essential idea is that for $P * Q$, a three-valued answer set for Q, consisting of positive and negative literals, is first determined. The positive literals constitute a regular answer set, while the negated literals make up a minimal set of naf literals required to produce the answer set from Q. These literals are propagated to the program P, along with those rules of Q that are not *decided* by these literals. The approach differs from work in *update logic programs* in two main respects: first, we ensure that the revising logic program has higher priority, and so we satisfy the success postulate; second, for the preference implicit in a revision $P * Q$, the program Q as a whole takes precedence over P, unlike update logic programs, since answer sets of Q are propagated to P. We show that a core group of the AGM postulates are satisfied, as are the postulates that have been proposed for update logic programs. As well, a prototype implementation is available.

Introduction

Answer set programming (ASP) (Baral 2003) has proven to be well-suited to problems in knowledge representation and reasoning (KR). The advent of efficient provers (Niemelä and Simons 1997; Eiter *et al.* 1997; Gebser *et al.* 2007) has led to the successful application of ASP in both KR and constraint satisfaction problems. However, an important consideration is that in any nontrivial domain, an agent's knowledge of the domain will most likely be incomplete or inaccurate, or it may become out of date as the domain evolves. Thus, over time an agent will need to adjust its knowledge after receiving new information concerning the domain.

In ASP there has been a substantial effort in developing approaches to updating a knowledge base, where a knowledge base is expressed as a logic program under the answer set semantics. In general, one is given a sequence of logic programs $(P_1, \ldots, P_n)$ where informally rules in P_i in some fashion or other take precedence over rules in P_j for $j < i$. However, as we argue in the next section, it isn't clear that such approaches capture a notion of *revision* or *update* of logic programs, so much as they capture a notion of *priority* or *preference* between rules in a program. Thus such approaches generally fail to satisfy properties that would be expected to hold for revision in classical logic. Part of the reason is that revision appears to be intrinsically more difficult in a nonmonotonic setting (such as in ASP) than in a monotonic one, such as in propositional logic. However, we also suggest that part of the problem is that extant approaches enforce a notion of priority at the level of the individual rule; instead we propose that the notion of priority given in a revision is a *program level* notion, in that for a revision specified at $P_1 * P_2$, program P_2, taken as a whole, has priority over P_1.

In this paper, an approach to the *revision* of logic programs is presented. A logic program is taken as a representation of an agent's epistemic state, while the corresponding answer sets are taken as a representation of the agent's beliefs. The approach describes revision, in that the postulate of *success* is adhered to; the idea is that for a revision of P_1 by P_2, beliefs (viz. elements of an answer set) given by P_2 should overrule those in P_1. This is carried out by first determining 3-valued answer sets of P_2. Each such answer set is a pair (X^+, X^-), where X^+ is a regular answer set of P_2, and X^- is a minimum set of negation as failure literals necessary to produce the answer set X^+. The information in each such 3-valued answer set, together with the rules of P_2 not used in the definition of the answer set, and along with the program P_1, is used to define an answer set (or answer sets) of $P_1 * P_2$.

The assumption of *success* leads to an approach with a quite different emphasis from previous approaches. In particular, for the revision of P_1 by P_2, the *program* P_2 is treated as having higher priority than the program P_1; this is in contrast with previous work, wherein the *rules* in P_2 are treated as having higher priority than (some or all) of the rules in P_1. We suggest that this distinction separates approaches addressing *priorities* in logic programs from *revision*. As well, it leads to an approach with different, and arguably more appropriate, properties from earlier work. For example, the approach is syntax independent, in that if two programs are uniform equivalent, then they behave the same with respect to revision. As well, a prototype has been implemented.

We argue that this approach is an appropriate interpretation for a notion of *revision* in logic programs. As well, the approach may be applied in cases where a problem is expressed as a sequence of NP-complete problems; for example, it allows the natural specification of a problem in which a 3-colouring of a graph is to be found, followed by a Hamiltonian cycle among the yellow vertices.

We discuss these issues in more detail in the next section, after formal preliminaries have been presented. After this, intuitions are given, and the following section presents the formal details and properties of the approach. We conclude with a brief discussion. Complete proofs are deferred to the full paper.

Background

Formal Preliminaries

Our language is built from a finite set of atoms $\mathcal{P} = \{a, b, \ldots\}$. A literal is an atom a or its negation $\neg a$; $\mathcal{L}$ is the set of literals. For a set X of literals, $not(X) = \{not\, a \mid a \in X\}$. For a literal l, $not\ l$ is sometimes referred to as a naf (negation as failure) literal or *weakly negated* literal. For $l \in \mathcal{L}$, $atom(l)$ is the atom corresponding to l; for a set X of literals, $atom(X) = \{atom(l) \mid l \in X\}$. A rule r is of the form

$$L_0 \leftarrow L_1, \ldots, L_n, not\ L_{n+1}, \ldots not\ L_m. \qquad (1)$$

where $L_0, \ldots, L_m$ are literals and $0 \leq n \leq m$. If $n = m$ then r is *positive*. If $m = 0$, then r is called a *fact*. We also allow the situation where L_0 is absent, in which case we denote the head by $\bot$; and r is called a *constraint*. The literal L_0 is called the *head* of r, and the set $\{L_1, \ldots, L_n, not\ L_{n+1}, \ldots, not\ L_m\}$ is the *body* of r. We use $head(r)$ to denote the head of rule r, and $body(r)$ to denote the body of r. Furthermore, $body^+(r) = \{L_1, \ldots, L_n\}$ and $body^-(r) = \{L_{n+1}, \ldots, L_m\}$. An (*extended*) *logic program*, or simply a *program*, is a finite set of rules.

A set of literals X is *consistent* if it does not contain a complementary pair a, $\neg a$ of literals and does not contain $\bot$. We say that X is *logically closed* iff it is either consistent or equals $\mathcal{L}$. The smallest set of literals being both logically closed and closed under a set P of positive rules is denoted by $Cn(P)$. The *reduct*, P^X, of P relative to a set X of literals is defined by $P^X = \{head(r) \leftarrow body^+(r) \mid r \in P,\ body^-(r) \cap X = \emptyset\}$ (Gelfond and Lifschitz 1990). A set X of literals is an *answer set* of a logic program P if $Cn(P^X) = X$. A program P is consistent just if it has an answer set not equal to $\mathcal{L}$. Thus a program with no answer sets is also counted as inconsistent. For example, the program $P = \{a \leftarrow, \quad b \leftarrow a, not\ c, \quad c \leftarrow not\ b\}$ has answer sets $AS(P) = \{\{a, b\}, \{a, c\}\}$.

Two programs P_1 and P_2 are *equivalent*, written $P_1 \equiv P_2$, if both programs have the same answer sets. Two programs are strongly equivalent (Lifschitz *et al.* 2001), written $P_1 \equiv_s P_2$, just if $P_1 \cup P_3 \equiv P_2 \cup P_3$ for every logic program P_3. Two programs are uniform equivalent (Eiter and Fink 2003), written $P_1 \equiv_u P_2$, just if $P_1 \cup F \equiv P_2 \cup F$ for every set of facts F.

Belief revision

Belief revision is the area of KR that is concerned with how an agent may incorporate new information about a domain into its knowledge base. In revision, a formula α is to be incorporated into the agent's set of beliefs K, so that the resulting knowledge base is consistent when α is. Since α may be inconsistent with K, revision may also necessitate the removal of beliefs from K in order to retain consistency. By a principle of *informational economy*, as many beliefs as possible are retained from K. A common approach in addressing belief revision is to provide a set of *rationality postulates* for belief change functions. The *AGM approach* (Alchourrón *et al.* 1985; Gärdenfors 1988) provides the best-known set of such postulates. An agent's beliefs are modelled by a set of sentences, called a *belief set*, closed under the logical consequence operator of a logic that includes classical propositional logic.

Subsequently, various researchers have argued that it is more appropriate to consider *epistemic states* as objects of revision. Epistemic state K effectively includes information regarding how the revision function itself changes following a revision. The belief set corresponding to epistemic state K is denoted $Bel(K)$. Formally, a revision operator $*$ maps an epistemic state K and new information α to a revised epistemic state $K * \alpha$. For set of formulas Ψ, define $\Psi + \alpha$ as $Cn(\Psi \cup \{\alpha\})$. Then, the basic AGM postulates for revision can be given as follows:

($K * 1$) $Bel(K * \alpha) = Cn(Bel(K * \alpha))$

($K * 2$) $\alpha \in Bel(K * \alpha)$

($K * 3$) $Bel(K * \alpha) \subseteq Bel(K) + \alpha$

($K * 4$) If $\neg\alpha \notin Bel(K)$ then $Bel(K) + \alpha \subseteq Bel(K * \alpha)$

($K * 5$) $Bel(K * \alpha)$ is inconsistent, only if $\vdash \neg\alpha$

($K * 6$) If $\alpha \equiv \psi$ then $Bel(K * \alpha) = Bel(K * \psi)$

Thus, the result of revising K by α is an epistemic state in which α is believed in the corresponding belief set ($K * 1$, $K * 2$); whenever the result is consistent, the revised belief set consists of the expansion of $Bel(K)$ by α ($K * 3$, $K * 4$); the only time that $Bel(K)$ is inconsistent is when α is inconsistent ($K * 5$); and revision is independent of the syntactic form of the formula for revision ($K * 6$).

It can be observed that two of the postulates, ($K * 3$) and ($K * 4$), are inappropriate in a system governed by a notion nonmonotonic consequence. As a simple example, consider where the agent believes that a particular individual is a bird and that it can fly. If it is subsequently learned that the bird was a penguin, the agent would also modify its knowledge base so that it believed that the individual did not fly. This example then violates both ($K * 3$) and ($K * 4$). Note that we can't circumvent this counterexample by simply excluding flying-penguin states of affairs, since we would want to allow the *possibility* that a penguin (perhaps an extremely fit penguin) flies, even though penguins, by default, do not fly. In consequence we focus on postulates ($K * 1$), ($K * 2$), ($K * 5$) and ($K * 6$), which we refer to as the *core* AGM postulates.

Note that nonmonotonic formalisms can nonetheless be treated from the standpoint of classical (AGM) revision; the issue is to express revision in terms of a monotonic foundation. Thus (Delgrande *et al.* 2008) addresses revision in ASP from the standpoint of the *SE models* of a program. This is in contrast to the work here, and previous work, which addresses belief change at the level of a logic program, rather than with respect to the underlying models.

Logic Program Updates

Previous work in ASP that addresses an agent's evolving knowledge base has generally been termed *logic program update* or *update logic programs*. In such approaches one begins with an *ordered logic program*, comprised of a sequence of logic programs $P = (P_1, \ldots, P_n)$. Rules in higher-ranked sets are, in some fashion or another, preferred over those in lower-ranked sets. Commonly this is implemented by using the ordering on rules to adjudicate which of two rules should be applied when both are applicable and their respective heads conflict; see for example (Inoue and Sakama 1999; Alferes *et al.* 2000; Eiter *et al.* 2002). Alternatively, other approaches use the ordering to "filter" rules, e.g. (Zhang and Foo 1998). Hence, in one fashion or another, some rules are selected over others, and these selected rules are used to determine the resulting answer sets. The following example, due to Patrick Krümpelmann, is illuminating.

$$P_1 = \{b \leftarrow\}, \quad P_2 = \{\neg a \leftarrow b\}, \quad P_3 = \{a \leftarrow\}.$$

In update logic programs, the rule given in P_2 is dropped, since its head conflicts with that of the rule in P_3. Yet disregarding the rule ing P_1 yields a consistent result, and moreover, this is the lowest-ranked rule, so arguably it *should* be disregarded.

A major stream of research in ASP has addressed *prioritised* or *preference* logic programs, where a prioritised logic program is a pair $(P, <)$ in which $<$ is some ordering over rules in program P. The intuition here is that some rules take precedence over (or override or are more important than) other rules. Syntactically, the *form* of an update logic program, given as a total order on programs, is of course an instance of a prioritised logic program. We suggest that an update logic program is in fact best regarded as a prioritised logic program. This is most clearly seen in those approaches where the focus is on rules whose heads conflict. Thus, for example in the approach of (Eiter *et al.* 2002), preferences come into play only between two rules when the head of one is the complementary literal of the other.

We can summarise the preceding by suggesting that previous work is essentially based at the *rule level*, in that higher-ranked rules preempts lower-ranked rules. In contrast, the approach here is based at the *program level*; that is for a revision $P_1 * P_2$, the program P_2 is considered *as a whole* to have priority over P_1. This is effected in a revision $P_1 * P_2$ by first determining answer sets of P_2, and then augmenting, as appropriate, these answer sets with additional information via P_1. (There is more to it than this, as described in the next section.) Arguably this is the appropriate level of granularity for revision: If an agent learns new information given in a program P, it is the program *as a whole* that comprises the agent's (new) knowledge. That is, a rule $r \in P$ isn't an isolated piece of knowledge, but rather, given possible negation as failure literals in $body(r)$, the potential instantiation of r depends non-locally on the entire program P.

(Eiter *et al.* 2002) suggests a number of alternative postulates that may be considered for update program updates. For our use, they are given as follows:

Initialisation: $AS(\emptyset * P) = AS(P)$.
Idempotency: $AS(P * P) = AS(P)$.
Tautology: If $head(r) \in body^+(r)$, for all $r \in P_2$, then $AS(P_1 * P_2) = AS(P_1)$.
Associativity: $AS(P_1 * (P_2 * P_3)) = AS((P_1 * P_2) * P_3)$.
Absorption: if $AS(P_2) = AS(P_3)$ then $AS(P_1 * P_2 * P_3) = AS(P_1 * P_2)$.
Augmentation: If $AS(P_2) \subseteq AS(P_3)$, then $AS(P_1 * P_2 * P_3) = AS(P_1 * P_3)$.
Disjointness: If $atom(P_1) \cap atom(P_2) = \emptyset$, then $AS((P_1 \cup P_2) * P_3) = AS(P_1 * P_3) \cup AS(P_2 * P_3)$.
Parallelism: If $atom(P_2) \cap atom(P_3) = \emptyset$, then $AS(P_1 * (P_2 \cup P_3)) = AS(P_1 * P_2) \cup AS(P_1 * P_3)$.
Non-Interference: If $atom(P_2) \cap atom(P_3) = \emptyset$, then $AS(P_1 * P_2 * P_3) = AS(P_1 * P_3 * P_2)$.

Many of these postulates are elementary and expected, yet most extant approaches have problems with them. In particular, most approaches do not satisfy tautology. Moreover, those that do satisfy tautology most often do so by specifically addressing this principle. It seems reasonable to suggest that the reason for this lack of adherence to basic postulates is that belief change with respect to ASP is a program-level operation, and not a rule-level operation.

Logic Program Revision: Intuitions

The overall goal is to come up with an approach to revision in logic programs (call it *LP revision*) under the answer set semantics. The intent is that the approach adhere insofar as possible to intuitions underlying classical (AGM) revision. In AGM revision, a *belief set* K is revised by a formula α to give another belief set $K * \alpha$. As described earlier, we take a logic program P as specifying an agent's *epistemic state*. The answer sets of P, $AS(P)$, represent the beliefs of the agent, and so are analogous to a belief set in AGM revision.

A key characteristic of AGM revision, and one that guides the approach here, is the *success postulate*. Recall that in the AGM approach, the success postulate stipulates that $\alpha \in Bel(K*\alpha)$, or in terms of models, that $Mod(Bel(K*\alpha)) \subseteq Mod(\alpha)$. Informally, in a revision by α, the logical content of α is retained. By analogy with the success postulate, for a revision of P_1 by P_2, the content of P_2 is given by its answer sets, and so in the revision $P_1 * P_2$, the answer sets of P_2 should in some sense be contained in those of $P_1 * P_2$. This notion is fundamental; as well, it has very significant ramifications in an approach to LP revision.

For example, consider the following programs, where we want to determine $P_1 * P_2$:

Example 1

$$\begin{aligned} P_1 &= \{b \leftarrow,\;\; c \leftarrow not\ d\} \\ P_2 &= \{a \leftarrow not\ b\} \end{aligned}$$

By our interpretation of the success postulate, since $\{a\}$ is an answer set of P_2, it should appear in the answer sets of $P_1 * P_2$ (that is, $\{a\}$ should be a subset of some answer set of $P_1 * P_2$). However, a was derived by the failure of being able to prove b in P_1. Consequently, if the answer sets of P_2 are to appear among the answer sets of $P_1 * P_2$, then the *reasons* for the answer sets of P_2 should also be retained. Consequently b should not appear in the answer sets of $P_1 * P_2$. Hence we would want to obtain $\{a, c\}$ as the answer set of $P_1 * P_2$.

So adherence to a success postulate requires that, if a literal $not\ p$ is used in a higher-ranked set of rules, it should override positive occurrences in lower-ranked sets. This also is in keeping with our assertion in the previous section, that in a revision we consider a program as a whole, and not at the individual rule level. Moreover, this example serves to distinguish the present approach from previous work, in that in previous work the assertion of a fact overrides an assumption of negation as failure at any level. Thus in previous work on update logic programs for the above example one would obtain the answer set $\{b, c\}$.

Thus in working towards an answer set for a revision $P_1 * P_2$, we first determine an answer set for P_2. However, we need to keep track of not just those literals that are (positively) derivable, but also a set of not literals necessary for the construction of the answer set. Consequently, we deal with three-valued answer sets. Thus for Example 1, in considering P_1, we need to keep track of the fact that a was derived in P_2 and that moreover $not\ b$ was used in this derivation, thereby necessitating the blocking of any later deriving of b in lower ranked rule sets. We write the three valued answer set of P_2 in Example 1 as $(\{a\}, \{b\})$. The three value answer set for $P_1 * P_2$ then is $(\{a, c\}, \{b, d\})$; and the corresponding answer set for $P_1 * P_2$ is $\{a, c\}$.

Consider next a variation of Example 1 where again we are to determine answer sets for $P_1 * P_2$:

Example 2

$$\begin{aligned} P_1 &= \{b \leftarrow,\ \ c \leftarrow\} \\ P_2 &= \{a \leftarrow not\ b,\ \ a \leftarrow not\ c\} \end{aligned}$$

The atom a may be obtained by $not\ b$ or $not\ c$ in P_2. By appeal to a principle of *informational economy* (Section), in a three valued answer set we retain a minimum number of not literals sufficient to derive the answer set. In the present example, this means that P_2 has two three-valued answer sets: $(\{a\}, \{b\})$ and $(\{a\}, \{c\})$. This leads to the three valued answer sets for $P_1 * P_2$: $(\{a, c\}, \{b\})$ and $(\{a, b\}, \{c\})$, and corresponding answer sets $\{a, c\}$ and $\{a, b\}$.

Consider finally the programs:

Example 3

$$\begin{aligned} P_1 &= \{b \leftarrow\} \\ P_2 &= \{a \leftarrow b\} \end{aligned}$$

P_2 has answer set $\emptyset$ (and three-value answer set $(\emptyset, \emptyset)$). However in next considering P_1 in the revision $P_1 * P_2$, one should be able to use the non-satisfied rule $a \leftarrow b$ and obtain an answer set $\{a, b\}$ for $P_1 * P_2$. This is by way of an extended notion of informational economy, in which a maximal justifiable set of beliefs is desirable. So rules of P_2 that are neither applied nor refuted should nonetheless be available for later steps in the revision.

These examples have dealt with a single occurrence of revision. Clearly the process can be iterated to a sequence of programs. Informally, an answer set for a sequence of programs is determined by finding 3-valued answer sets for higher-ranked programs, and propagating these answer sets, along with *undecided* rules, to lower ranked programs. Consequently, answer sets are built incrementally, with literals at a higher level being retained at lower levels. In the next subsection, for generality, we work with sequences of programs rather than just pairs.

Logic Program Revision: Approach

This section describes an approach to LP revision based on the intuitions of the previous subsection. Consider by way of analogy, classical AGM revision: For a revision $K * \alpha$, the formula α is to be incorporated in K; since $Bel(K) \cup \{\alpha\}$ may well be inconsistent, formulas in $Bel(K)$ may be dropped in order to obtain a consistent result. Similarly in a revision of programs $P_1 * P_2$: we would like the result to be consistent if possible.

In outline, the goal is to determine answer sets for $P_1 * P_2$. To this end, an answer set X of P_2 is determined and it, along with the rules in P_1, say P_1', that do not take part in the definition of X, are propagated to P_2. Since the result should be consistent, we consider maximal subsets of $P_2 \cup P_1'$ that are consistent with X and use these to determine the resulting answer sets for the revision. We begin by defining the relevant notion of an answer set with respect to revision.

By a three-valued interpretation we will mean an ordered pair of sets $X = (X^+, X^-)$ where $X^+, X^- \subseteq \mathcal{L}$ and $X^+ \cap X^- = \emptyset$. The intuition is that members of X^+ constitute an answer set of some program, while X^- contains a minimum set of assumptions necessary for the derivation of X^+. A (canonical) program corresponding to a 3-valued interpretation is given by

$$Pgm(X) = \{a \leftarrow \mid a \in X^+\} \cup \{\bot \leftarrow a \mid a \in X^-\}.$$

The consequence relation $Cn(.)$ on definite programs is extended to arbitrary logic programs by the simple expedient of treating a weakly negated literal $not\ \ l$ as a new atom. Thus for example $Cn(\{a \leftarrow,\ \ b \leftarrow a,\ \ c \leftarrow not\ \ d\})$ is $\{a, b\}$.

The next definition extends the notion of reduct to 3-valued interpretations.

Definition 1 *Let P be a program and $X = (X^+, X^-)$ a 3-valued interpretation.*

P^X, the min-reduct of P wrt X*, is the program obtained from P by:*

1. *deleting every rule $r \in P$ where $body^-(r) \cap X^+ \neq \emptyset$, and*
2. *replacing any remaining rule $r \in P$, by*
   $head(r) \leftarrow body^+(r), not(body^-(r) \setminus X^-)$.

Part 1 above is the same as in the standard definition of reduct. In Part 2, just those naf literals appearing in X^- are deleted from the bodies of the remaining rules. The following definition extends the notion of an answer set to 3-valued interpretations.

Definition 2 *Let P be a program and $X = (X^+, X^-)$ a 3-valued interpretation.*

$X = (X^+, X^-)$ *is a* 3-valued answer set for P *if* $Cn(P^{X^+}) = Cn(P^X) = X^+$ *and for any* $Y = (X^+, Y^-)$ *where* $Y^- \subset X^-$ *we have that* $Cn(P^Y) \neq X^+$.

The set of 3-valued answer sets of program P is denoted $3AS(P)$.

Thus for 3-valued answer set $X = (X^+, X^-)$ of P, we have that X^+ is an answer set of P. As well, 3-valued answer sets include sufficient negation as failure literals for the derivation of the answer set. Thus, for $\{a \leftarrow not\ b,\ a \leftarrow not\ c\}$ there are two 3-valued answer sets $(\{a\}, \{b\})$ and $(\{a\}, \{c\})$ along with answer set $\{a\}$.

As suggested at the start of the section, in a revision $P_1 * P_2$ we need to isolate a subset of P_1 that is consistent with P_2. We give the necessary definitions next.

Definition 3 *Let P_1, P_2 be programs. Define $P_1 \downarrow P_2$ by:*
If P_2 is not consistent, then $P_1 \downarrow P_2 = \mathcal{L}$.
Otherwise:

$$\begin{aligned} P_1 \downarrow P_2 \quad = \quad \{ & P' \cup P_2 \mid P' \subseteq P_1 \text{ and} \\ & P' \cup P_2 \text{ is consistent and} \\ & \text{for } P' \subset P'' \subseteq P_2, \\ & \quad P'' \cup P_2 \text{ is inconsistent.}\} \end{aligned}$$

Thus for $P \in P_1 \downarrow P_2$, P consists of P_2 together with a maximal set of rules from P_1 such that the P is consistent.

Given a sequence of logic programs $(P_1, \ldots, P_n)$, the revision process can now be informally described as follows:

1. Let $X_n \in 3AS(P_n)$; that is, $X_n = (X_n^+, X_n^-)$, is a 3-valued answer set for P_n.
2. In the general case, one has a 3-valued answer set $X_{i+1} = (X_{i+1}^+, X_{i+1}^-)$ from the revision sequence $(P_{i+1}, \ldots, P_n)$. A maximal set of rules in $(P_i, \ldots, P_n)$ consistent with X_{i+1} is used to determine a 3-valued answer set $X_i = (X_i^+, X_i^-)$ for the revision sequence $(P_i, \ldots, P_n)$.
3. A 3-valued answer set $X_1 = (X_1^+, X_1^-)$ for the revision sequence $(P_1, \ldots, P_n)$ then yields the answer set X_1^+ for the full sequence $(P_1, \ldots, P_n)$.

With this setting, we can give the main definition for the answer sets of a revision sequence of programs. To this end, a revision problem is given by a sequence $(P_1, \ldots, P_n)$ of logic programs; the goal is to determine answer sets of the sequence under the interpretation that a higher-indexed program, taken as a single entity, takes priority over a lower-indexed program. We write the revision sequence in notation closer to that of standard belief revision, as $P_1 * \ldots * P_n$; our goal then is to characterise the resulting answer sets of $P_1 * \ldots * P_n$.

Definition 4 *Let $P = (P_1, \ldots, P_n)$ be a sequence of logic programs.*

$X \in AS(P_1 * \ldots * P_n)$ iff there is a sequence:

$$((P_1^r, X_1), \ldots, (P_n^r, X_n))$$

such that for $1 \leq i \leq n$, P_i^r is a logic program and X_i is a 3-valued interpretation,
and:

1. *i.)* $P_n^r = P_n$ *and*
   ii.) X_n *is a 3-valued answer set for* P_n.
2. *for* $i < n$*:*
   i.) $P_i^r \in P_i \downarrow (P_{i+1}^r \cup Pgm(X_{i+1}))$ *and*
   ii.) X_i *is a 3-valued answer set for* P_i^r.
3. $X = X_1^+$.

The case of binary revision is of course simpler. Though redundant, it is instructive, and so we give it next.

Definition 5 *Let P_1, P_2 be logic programs. $X \in AS(P_1 * P_2)$ is an answer set of $P_1 * P_2$ if:*

1. *there is a 3-valued answer set X_2 of P_2 and,*
2. *for some X', (X, X') is a 3-valued answer set of $P_1 \downarrow (P_2 \cup Pgm(X_2))$*

Examples

Consider the examples given earlier. For Example 1 we have:

$$P_1 = \{b \leftarrow,\ c \leftarrow not\ d\}, \qquad P_2 = \{a \leftarrow not\ b\}$$

For $P_1 * P_2$ we obtain:

$$\begin{aligned} P_2^r &= P_2, & X_2 &= (\{a\}, \{b\}) \\ P_1^r &= \{c \leftarrow not\ d\} & X_1 &= (\{a, c\}, \{b, d\}). \\ &\quad \cup \{a \leftarrow,\ \perp \leftarrow b\}, \end{aligned}$$

Thus, P_2 has 3-valued answer set $(\{a\}, \{b\})$: $\{a\}$ is a (standard) answer set for P_2, and it depends on, at minimum, the assumption of b being false by default. This in turn requires a commitment to the non-truth of b in next considering P_1. The program P_1^r given by rules of P_1 consistent with $(\{a\}, \{b\})$ consists of the single rule $c \leftarrow not\ d$ along with an encoding of the 3-valued interpretation $(\{a\}, \{b\})$. We obtain the 3-valued answer set $(\{a, c\}, \{b, d\})$, with corresponding answer set $\{a, c\}$.

Consider next Example 2:

$$P_1 = \{b \leftarrow,\ \ c \leftarrow\}, \qquad P_2 = \{a \leftarrow not\ b,\ \ a \leftarrow not\ c\}$$

P_2 has two 3-valued answer sets $(\{a\}, \{b\})$ and $(\{a\}, \{c\})$. Again, $\{a\}$ is a (standard) answer set for P_2, but it depends on, at minimum, the assumption of either b or c being false by default. This in turn requires a commitment to the falsity of one of b or c in next considering P_1. As a result, the 3-valued answer set $(\{a\}, \{b\})$ yields the program P_1^r given by $\{c \leftarrow,\ a \leftarrow,\ \perp \leftarrow b\}$, while the 3-valued answer set $(\{a\}, \{c\})$ yields the program $\{b \leftarrow,\ a \leftarrow,\ \perp \leftarrow c\}$. Consequently for the revision we obtain two 3-valued answer sets $(\{a, c\}, \{b\})$ and $(\{a, b\}, \{c\})$, with corresponding answer sets $\{a, c\}$ and $\{a, b\}$.

For Example 3, where $P_1 = \{b \leftarrow\}$ and $P_2 = \{a \leftarrow b\}$, we obtain:

$$\begin{aligned} P_2^r &= P_2, & X_2 &= (\emptyset, \emptyset) \\ P_1^r &= \{b \leftarrow\} \cup \{a \leftarrow b\}, & X_1 &= (\{a, b\}, \{\}). \end{aligned}$$

Thus there is one 3-valued answer set, $(\{a, b\}, \emptyset)$, with answer set $\{a, b\}$.

We consider two more small examples to further illustrate the approach.

$$P_1 = \{a \leftarrow, \;\; b \leftarrow\}, \qquad P_2 = \{\bot \leftarrow a, b\}$$

For $P_1 * P_2$, there are two 3-valued answer sets, $(\{a\}, \emptyset)$, $(\{b\}, \emptyset)$ with corresponding answer sets $\{a\}$, $\{b\}$. This is what would be desired: P_2 requires that a and b cannot be simultaneously true, while P_1 states that a and b are both true. In this case, P_2 is retained, with a "maximal" part of P_1 also held.

$$P_1 = \{a \leftarrow, \;\; d \leftarrow b\}, \qquad P_2 = \{b \leftarrow \mathit{not}\ a, \;\; c \leftarrow \mathit{not}\ \neg a\}$$

In this case, P_2 has three-valued interpretation $(\{b, c\}, \{a, \neg a\})$; hence the derivation of b and c relies on the possibility of a being true, and of a being false. There is one 3-valued answer set for $P_1 * P_2$, $(\{b, c, d\}, \{a, \neg a\})$, with answer set $\{b, c, d\}$.

Properties

We next consider formal properties of the approach.

Section suggested that four of the basic AGM postulates are appropriate in a nonmonotonic framework. With respect to these postulates, we obtain the following:

Theorem 1 *Let P_1, P_2, P_3 be logic programs.*

$(A * 1)$ $AS(P_1 * P_2) \subseteq 2^{\mathcal{L}}$.

$(A * 2)$ *If $X \in AS(P_2)$ then there is $X' \in AS(P_1 * P_2)$ such that $X \subseteq X'$.*

$(A * 5a)$ $AS(P_1 * P_2) = \mathcal{L}$ *only if* $AS(P_2) = \mathcal{L}$

$(A * 5b)$ $AS(P_1 * P_2) = \emptyset$ *only if* $AS(P_2) = \emptyset$

$(A * 6)$ *If $P_2 \equiv_u P_3$ then $P_1 * P_2 = P_1 * P_3$.*

Thus the result of revision is a set of answer sets $(A * 1)$, which is to say, if an agent's beliefs are given by a set of answer sets, corresponding to potential states of the world, then a revision sequence also yields a set of such beliefs. The key property of the approach is given by $(A * 2)$, corresponding to the success postulate: in a revision $P_1 * P_2$, beliefs as expressed in P_2 override those of P_1. The two parts of $(A * 5)$ hold by virtue of the fact that in a revision $P_1 * P_2$, only some consistent (with P_2) subset of P_1 is used in the revision. $(A * 6)$ is a version of independence of syntax. The postulate fails if $P_2 \equiv_u P_3$ is replaced by $AS(P_2) = AS(P_3)$: a counterexample is given by $P_1 = \{b \leftarrow\}$, $P_2 = \{a \leftarrow \mathit{not}\ b\}$, and $P_3 = \{a \leftarrow \mathit{not}\ c\}$. Consequently, appropriate versions of the core AGM postulates hold in the approach. As suggested in Section , the other two basic postulates are not appropriate in a nonmonotonic framework.

With regards to the postulates given in Section for logic program updates, we obtain the following.

Theorem 2 *Let P_1, P_2, P_3 be logic programs.*

Then P_1, P_2, P_3 satisfy initialisation, idempotency, tautology, *and* non-interference.

Programs P_1, P_2, P_3 also satisfy the principle:

UAbsorption: *if $P_2 \equiv_u P_3$ then*
$AS(P_1 * P_2 * P_3) = AS(P_1 * P_2)$.

These principles are elementary but nonetheless desirable; as mentioned, the majority of approaches in update logic programs fail to satisfy *tautology*.

The remaining principles can be argued to be undesirable, with the possible exception of *associativity*. Not surprisingly, *absorption* fails at the level of answer sets, though not at the level of uniform equivalence (as given by **UAbsorption**). *Augmentation* would seem to be related to a notion of monotonicity, and hence is undesirable. *Disjointness* and *parallelism* both clearly fail. Arguably both *should* fail; consider the the case where $P_2 = \emptyset$. Disjointness in this situation reduces to:

$$AS(P_1 * P_3) = AS(P_1 * P_3) \cup AS(P_3)$$

which is clearly undesirable.

Discussion

This paper has described an approach to logic program revision in which the focus is on *revision* as understood in the belief revision community. Consequently, the key *success* postulate is taken seriously. This leads to an approach with quite different properties than other approaches that have appeared in the literature. In particular, for a revision $P_1 * P_2$ the program P_2 is treated as a whole as having higher priority than P_1 in that answer sets of P_2 are propagated to P_1. This is in contrast to logic program update, where essentially once selects rules to apply, giving preference to rules in P_2, and then applying these selected rules. This distinction has an important consequence, and requires the use of three-valued interpretations, in which literals assumed to be true at a higher ranked program can override literals used as facts in a lower-ranked program. Thus, the (necessary) use of $\mathit{not}\ p$ in a higher ranked program blocks the assertion of p as a fact in a lower-ranked program. Again, this is a necessary consequence of a strong commitment to the success postulate.

Arguably the approach helps cast light on the logic-program-update landscape. We suggest here that approaches falling under the general heading of *update logic programs* are are more appropriately viewed as dealing with preferences or priorities over rules, rather than revision or update per se.

The approach at hand seems to fall somewhere between a syntactic approach and a strongly semantic one, such as (Delgrande *et al.* 2008). In (Delgrande *et al.* 2008), for example, revision is phrased in terms of the underlying SE models of a theory. Consequently, very appealing theoretical results are obtained, including the large majority of AGM postulates, as well as the appropriate logic program update postulates. The disadvantage is that, since revision is phrased in terms of SE models, it is not immediately clear how to obtain a *reasonable* logic program corresponding to the models given by the revision.

The present approach also does not yield a logic program that corresponds to a revision (but see below for a discussion); however, it does define the answer sets of a revision sequence. Moreover, the answer sets obtained via a revision exhibit reasonable properties – for example, the core AGM postulates are obtained, including syntax independence under uniform equivalence and a success postulate, as well as the appropriate set of logic program update postulates. The approach would seem to have some hope for practical implementation. It would be applicable, clearly, in problems wherein the answer sets of the most recent program are to hold sway over the answer sets of lower-ranked programs. As well, it can be directly applied to problems involving a sequence of NP complete problems, for example in a situation where the solution to one problem feeds as input into a second. A representative example described earlier involves finding a three colouring for a graph, and then basing another problem (such as Hamiltonian cycle) on the vertices of a specific colour.

References

C.E. Alchourrón, P. Gärdenfors, and D. Makinson. On the logic of theory change: Partial meet functions for contraction and revision. *Journal of Symbolic Logic*, 50(2):510–530, 1985.

José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1-3):43–70, 2000.

C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

J. Delgrande, T. Schaub, H. Tompits, and S. Woltran. Belief revision of logic programs under answer set semantics. In G. Brewka and J. Lang, editors, *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning*, pages 411–421, Sydney, Australia, 2008. AAAI Press.

T. Eiter and M. Fink. Uniform equivalence of logic programs under the stable model semantics. In Catuscia Palamidess, editor, *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP'03)*, volume 2916 of *Lecture Notes in Computer Science*, pages 224–238. Springer-Verlag, 2003.

T. Eiter, N. Leone, C. Mateis, G. Pfeifer, , and F. Scarcello. A deductive system for nonmonotonic reasoning. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the Fourth International Conference on Logic Programming and Non-Monotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 363–374. Springer Verlag, 1997.

Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6):711–767, 2002.

P. Gärdenfors. *Knowledge in Flux: Modelling the Dynamics of Epistemic States*. The MIT Press, Cambridge, MA, 1988.

M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press/The MIT Press, 2007.

M. Gelfond and V. Lifschitz. Logic programs with classical negation. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the International Conference on Logic Programming*, pages 579–597. MIT, 1990.

K. Inoue and C. Sakama. Updating extended logic programs through abduction. In *Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*), volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 147–161. Springer, 1999.

V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Artificial Intelligence*, pages 420–429. Springer-Verlag, 1997.

Yan Zhang and Norman Y. Foo. Updating logic programs. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence* (*ECAI'98*), pages 403–407, 1998.