A Preference-Based Framework for Updating Logic Programs

James P. Delgrande[1], Torsten Schaub[2,1], and Hans Tompits[3]

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

[2] Institut für Informatik, Universität Potsdam,
August-Bebel-Straße 89, D-14482 Potsdam, Germany
torsten@cs.uni-potsdam.de

[3] Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9–11, A–1040 Wien, Austria
tompits@kr.tuwien.ac.at

Abstract. We present a framework for updating logic programs under the answer-set semantics that builds on existing work on preferences in logic programming. The approach is simple and general, making use of two distinct complementary techniques: defaultification and preference. While defaultification resolves potential conflicts by inducing more answer sets, preferences then select among these answer sets, yielding the answer sets generated by those rules that have been added more recently. We examine instances of the framework with respect to various desirable properties; for the most part, these properties are satisfied by instances of our framework. Finally, the proposed framework is also easily implementable by off-the-shelf systems.

1 Introduction

Over the last decade, *answer-set programming* (ASP) [1] has become a major approach for knowledge representation and reasoning. Given that knowledge is always subject to change, there has been a substantial effort in developing approaches to updating logic programs under the answer-set semantics [2–10]. Unfortunately, the problem of update appears to be intrinsically more difficult in a nonmonotonic setting (such as in ASP) than in a monotonic one, such as in propositional logic [11]. As a consequence, many approaches are quite involved and the set of approaches is rather heterogeneous.

In contrast to this, we propose a simple and general framework for updating logic programs that is based on two well-known parameterisable techniques in ASP: *defaultification* [12, 13] and *preference handling* [14–16]. This is based on the following idea: The primary purpose of updating mechanisms is to resolve conflicts among more recent and less recent rules. To this end, we first need to detect potential conflicts between newer and older rules. Second, we need to prevent them from taking place. And finally, we need to resolve conflicts in favour of more recent updating rules. The two last steps are accomplished by defaultification and preferences. While defaultification resolves potential conflicts by inducing complementary answer sets, preferences then are used to select among these answer sets, producing those answer sets generated by rules that have been added more recently. As a result, our approach is easily implementable by appeal to existing off-the-shelf technology for preference handling, such as the front-end tool `plp`[4] (used in conjunction with standard ASP-solvers, like `smodels`[5] or `dlv`[6]), the genuine preference-handling ASP-solver `nomore`$^{<}$[7], or meta-interpretation methods [17].

Our techniques have further advantages: First, defaultification also allows for the elimination of incoherent situations, even in an updating program or in intermediate programs in an updating sequence. Second, preferences provide a modular way of capturing an update history, rather than an explicit program transformation, as done for instance in the approaches of Eiter *et al.* [9] or Zhang and Foo [4].

After giving some background, we introduce our framework in Section 3, along with an evaluation according to update principles proposed by Eiter *et al.* [9] in the context of ASP. Section 4 gives a more detailed comparison to the latter approach and shows how our approach deals with two well-known examples from the literature. The paper concludes with a discussion in Section 5.

2 Background

Given an alphabet $\mathcal{P}$, an *extended logic program*, or simply a *program*, is a finite set of rules of form

$$l_0 \leftarrow l_1, \ldots, l_m, \mathit{not}\ l_{m+1}, \ldots, \mathit{not}\ l_n, \qquad (1)$$

where $n \geq m \geq 0$ and each l_i $(0 \leq i \leq n)$ is a *literal*, that is, an *atom* $a \in \mathcal{P}$ or its negation $\neg a$. The set of all literals is given by $\mathcal{L} = \mathcal{P} \cup \{\neg a \mid a \in \mathcal{P}\}$. For a literal l, define $\bar{l} = a$ if $l = \neg a$, and $\bar{l} = \neg a$ if $l = a$. The set of atoms occurring in a program Π is denoted by $\mathit{atom}(\Pi)$. For a rule r of form (1), let $\mathit{head}(r) = l_0$ be the *head* of r and $\mathit{body}(r) = \{l_1, \ldots, l_m, \mathit{not}\ l_{m+1}, \ldots, \mathit{not}\ l_n\}$ the *body* of r. Furthermore, let $\mathit{body}^+(r) = \{l_1, \ldots, l_m\}$ and $\mathit{body}^-(r) = \{l_{m+1}, \ldots, l_n\}$. Rule r is *positive*, if $\mathit{body}^-(r) = \emptyset$.

A set of literals X is *consistent* if it does not contain a complementary pair a, $\neg a$ of literals. 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 Π of positive rules is denoted by $Cn(\Pi)$. The *reduct*, Π^X, of Π relative to a set X of literals is defined by $\Pi^X = \{\mathit{head}(r) \leftarrow \mathit{body}^+(r) \mid r \in \Pi,\ \mathit{body}^-(r) \cap X = \emptyset\}$ [18]. A set X of atoms is an *answer set* of an extended logic program Π if $Cn(\Pi^X) = X$. Two programs Π_1 and Π_2 are said to be *equivalent*, written $\Pi_1 \equiv \Pi_2$, if both programs have the same answer sets.

An *ordered logic program* is a pair $(\Pi, <)$, where Π is a logic program and $< \subseteq \Pi \times \Pi$ is a strict partial order. Given $r_1, r_2 \in \Pi$, the relation $r_1 < r_2$ expresses that r_2 has *higher priority* than r_1. This informal interpretation can be made

[4] `http://www.cs.uni-potsdam.de/~torsten/plp`.
[5] `http://www.tcs.hut.fi/Software/smodels`.
[6] `http://www.dlvsystem.com`.
[7] `http://www.cs.uni-potsdam.de/wv/nomorepref`.

precise in different ways. In what follows, we consider three such interpretations: B-*preference* [14], D-*preference* [15], and W-*preference* [16]. Given $(\Pi, <)$, all of these approaches use $<$ for selecting preferred answer sets among the standard answer sets of Π. The approaches are defined as follows. Let X be a consistent set of literals and define $\Pi_X = \{r \in \Pi \mid \mathit{body}^+(r) \subseteq X \text{ and } \mathit{body}^-(r) \cap X = \emptyset\}$. Then:

1. X is $<_D$-*preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of Π_X such that, for every $i, j \in I$, we have that:
   (a) if $r_i < r_j$, then $j < i$;
   (b) $\mathit{body}^+(r_i) \subseteq \{\mathit{head}(r_k) \mid k < i\}$; and
   (c) if $r_i < r'$ and $r' \in \Pi \setminus \Pi_X$, then
      i. $\mathit{body}^+(r') \not\subseteq X$ or
      ii. $\mathit{body}^-(r') \cap \{\mathit{head}(r_k) \mid k < i\} \neq \emptyset$.
2. X is $<_W$-*preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of Π_X such that, for every $i, j \in I$, we have that:
   (a) if $r_i < r_j$, then $j < i$;
   (b) i. $\mathit{body}^+(r_i) \subseteq \{\mathit{head}(r_k) \mid k < i\}$ or
      ii. $\mathit{head}(r_i) \in \{\mathit{head}(r_k) \mid k < i\}$; and
   (c) if $r_i < r'$ and $r' \in \Pi \setminus \Pi_X$, then
      i. $\mathit{body}^+(r') \not\subseteq X$ or
      ii. $\mathit{body}^-(r') \cap \{\mathit{head}(r_k) \mid k < i\} \neq \emptyset$ or
      iii. $\mathit{head}(r') \in \{\mathit{head}(r_k) \mid k < i\}$.
3. X is $<_B$-*preserving*, if there exists an enumeration $\langle r_i \rangle_{i \in I}$ of Π_X such that, for every $i, j \in I$, we have that:
   (a) if $r_i < r_j$, then $j < i$; and
   (b) if $r_i < r'$ and $r' \in \Pi \setminus \Pi_X$, then
      i. $\mathit{body}^+(r') \not\subseteq X$ or
      ii. $\mathit{body}^-(r') \cap \{\mathit{head}(r_k) \mid k < i\} \neq \emptyset$ or
      iii. $\mathit{head}(r') \in X$.

As shown by Schaub and Wang [16], the three strategies yield an increasing number of preferred answer sets. That is, D–preference is stronger than W–preference, which is stronger than B–preference, which is stronger than the empty preference (i.e., having no preference).

Analogously to the unordered case, we call two ordered logic programs *equivalent* iff they have the same order-preserving answer sets, and we use again "$\equiv$" as a symbol for program equivalence. Note that an unordered program Π is trivially equivalent to the program $(\Pi, \emptyset)$ having an empty order relation, as every answer set of Π is a $<_\sigma$-preserving answer set of $(\Pi, <)$, for $<= \emptyset$ and $\sigma \in \{D, B, W\}$. Hence, allowing a slight abuse of notation, we sometimes identify an unordered program Π with $(\Pi, \emptyset)$.

3 The Basic Framework

3.1 Update Programs

The primary purpose of updating mechanisms is to resolve conflicts among newer and older rules. As mentioned, our approach is to first detect potential conflicts, second, to stop them from taking place, and third, to resolve these conflicts in favour of the updating rules.

A potential conflict manifests itself by complementary head literals. Two rules r_1, r_2 are said to be *conflicting* if $head(r_1) = \overline{head(r_2)}$. We represent potential conflicts among rules within two programs Π_1 and Π_2 in terms of the set

$$C(\Pi_1, \Pi_2) = \{(r_1, r_2) \mid r_1 \in \Pi_1, r_2 \in \Pi_2, head(r_1) = \overline{head(r_2)}\}.$$

For avoiding conflicts, we weaken rules by turning them into *defaultised* rules, as originally used by Sadri and Kowalski [12]. For rule r, we define

$$r^d = head(r) \leftarrow body(r), not\ \overline{head(r)}.$$

Similarly, for a program Π, we define $\Pi^d = \{r^d \mid r \in \Pi\}$ and call it the *defaultification* of Π. For example, program $\{a \leftarrow, \neg a \leftarrow\}$ has the answer set $\mathcal{L}$, while $\{a \leftarrow, \neg a \leftarrow\}^d = \{a \leftarrow not\ \neg a, \neg a \leftarrow not\ a\}$ has two answer sets, $\{a\}$ and $\{\neg a\}$. Note that, given that bodies of rules are *sets*, we have $r^d = (r^d)^d$, for every rule r.

The next result shows how the aforementioned “weakening” is to be understood.

Theorem 1. *Let Π be a logic program.*
Every consistent answer set of Π is an answer set of Π^d.

We propose to use preferences among rules for resolving inconsistencies. This provides us with several degrees of freedom: First, one can choose among different preference-handling strategies; and second, these strategies can be imposed in different ways on the rules. As well, defaultification can be applied universally or selectively to rules. We next detail three specific ways of applying the framework.

To begin with, we give the following very basic definition of an update operator on logic programs:

Definition 1. *The update program obtained for updating program Π_1 by the program Π_2 via update operator $*_0$ is given by the ordered logic program*

$$\Pi_1 *_0 \Pi_2 = (\Pi_1^d \cup \Pi_2^d, \Pi_1^d \times \Pi_2^d).$$

Thus, the ordered logic program is over $\Pi_1^d \cup \Pi_2^d$, and $<$ is defined so that every rule in Π_1^d has less priority than every rule in Π_2^d. We do not suggest that this is a good update operator; in fact this operator is usually too strict, since it establishes a preference between *all* rules in programs Π_1 and Π_2 even though they may not conflict. However, it provides a simple, basic instance of our approach.

A conflict-oriented approach is the following.

Definition 2. *The update program obtained for updating program Π_1 by the program Π_2 via update operator $*_1$ is given by the ordered logic program*

$$\Pi_1 *_1 \Pi_2 = (\Pi_1^d \cup \Pi_2^d, C(\Pi_1^d, \Pi_2^d)).$$

Operator $*_1$ globally weakens the rules in the program and imposes preferences along potential conflicts.

A refinement of the above approach is the following:

Definition 3. *The update program obtained for updating program Π_1 by the program Π_2 via update operator $*_2$ is given by the ordered logic program*

$$\Pi_1 *_2 \Pi_2 = (\Pi_c^d \cup ((\Pi_1 \cup \Pi_2) \setminus \Pi_c)), C(\Pi_1^d, \Pi_2^d)),$$

where $\Pi_c = \{r_1, r_2 \mid (r_1, r_2) \in C(\Pi_1, \Pi_2)\}$.

Operator $*_2$ restricts defaultification to conflicting rules. Unlike the previous update operations, this necessitates program transformations whenever conflicting rules are encountered upon iterated updates (see below). Further definitions are possible.

As well, one must also specify a preference-handling strategy, which we consider next.

Definition 4. *Let $\Pi_1 * \Pi_2$ be an update program for some update operator $*$, $\sigma \in \{B, D, W\}$ a preference-handling strategy, and X a set of literals.*

Then, X is an answer set of $\Pi_1 * \Pi_2$ with respect to σ *iff X is a $<_\sigma$-preserving answer set of $\Pi_1 * \Pi_2$.*

Depending on the chosen preference-handling strategy, update programs may admit several, one, or no answer sets. The latter is worth illustrating because it motivates the increasing restriction of preferences among rules when defining our update operators. For instance, the update program $\{b \leftarrow \mathit{not}\ a\} *_0 \{c \leftarrow \mathit{not}\ b\}$ has no answer set with respect to either the B-, D-, or W-strategy. In contrast, we obtain answer set $\{b\}$ when applying $*_1$ and $*_2$. For another example, consider $\{a \leftarrow\} *_0 \{b \leftarrow a\}$. We obtain answer set $\{a, b\}$ with respect to the B strategy, but no answer set with the D or W strategy. In contrast to this, we get answer set $\{a, b\}$ when applying $*_1$ and $*_2$ no matter which preference-handling strategy we chose.

For illustrating the different behaviour of $*_0$, $*_1$, and $*_2$ with respect to inconsistent programs, consider $\Pi_1 = \{r_1 : a \leftarrow,\ r_2 : \neg a \leftarrow\}$ and $\Pi_2 = \{r_3 : b \leftarrow a\}$. Here, we have

$$r_1^d = a \leftarrow \mathit{not}\ \neg a, \qquad r_2^d = \neg a \leftarrow \mathit{not}\ a,$$
$$r_3^d = b \leftarrow a, \mathit{not}\ \neg b,$$

and $C(\Pi_1^d, \Pi_2^d) = \emptyset$. Hence, we get the following update programs:

$$\begin{aligned}
\Pi_1 *_0 \Pi_2 &= (\{r_1^d,\ r_2^d,\ r_3^d\}, \{r_1^d < r_3^d, r_2^d < r_3^d\}),\\
\Pi_1 *_1 \Pi_2 &= \Pi_1^d \cup \Pi_2^d = \{r_1^d,\ r_2^d,\ r_3^d\}, \quad \text{and}\\
\Pi_1 *_2 \Pi_2 &= \Pi_1 \cup \Pi_2 = \{r_1,\ r_2,\ r_3\}.
\end{aligned}$$

Clearly, Π_1 is inconsistent, i.e., it has the single answer set $\mathcal{L}$. Under B-preference, $\Pi_1 *_0 \Pi_2$ has $\{\neg a\}$ and $\{a, b\}$ as its answer sets, whereas under D- and W-preference, only $\{\neg a\}$ is an answer set of $\Pi_1 *_0 \Pi_2$. Roughly speaking, $\{a, b\}$ is not an answer set under D- and W-preference because of the "prescriptive" nature of these preference strategies. For $\Pi_1 *_1 \Pi_2$, we also get the two answer sets $\{\neg a\}$ and $\{a, b\}$, while $\Pi_1 *_2 \Pi_2$ again yields the inconsistent answer set $\mathcal{L}$.

The common factor, however, is that each selection criterion chooses its preferred answer sets among those of the defaultification of the union of the original programs.

Theorem 2. *Let $\Pi_1 *_i \Pi_2$ be an update program for some $i = 0, 1, 2$ and let $\sigma \in \{B, D, W\}$ be a preference-handling strategy.*

Then, every answer set of $\Pi_1 *_i \Pi_2$ with respect to σ is an answer set of $(\Pi_1 \cup \Pi_2)^d$.

Iterated updates are easily defined.

Definition 5. *Let $(\Pi_1, \ldots, \Pi_n)$ be a sequence of logic programs, for $n \geq 2$, and let $*$ be a binary update operator.*

Then, $*(\Pi_1, \ldots, \Pi_n)$, the update program obtained from $(\Pi_1, \ldots, \Pi_n)$, *is the ordered logic program given as follows:*

$$*(\Pi_1, \ldots, \Pi_n) = \begin{cases} \Pi_1 * \Pi_2, & \text{if } n = 2; \\ ([*(\Pi_1, \ldots, \Pi_{n-1})] * \Pi_n), & \text{if } n > 2. \end{cases}$$

Definition 6. *Let $*(\Pi_1, \ldots, \Pi_n)$ be an update program for some update operator $*$, let $\sigma \in \{B, D, W\}$ be a preference-handling strategy, and let X be a set of literals.*

Then, X is an answer set of $*(\Pi_1, \ldots, \Pi_n)$ with respect to σ *iff X is a $<_\sigma$-preserving answer set of $*(\Pi_1, \ldots, \Pi_n)$.*

Whenever convenient, we write $(\Pi_1 * \ldots * \Pi_n)$ instead of $*(\Pi_1, \ldots, \Pi_n)$. As in Theorem 2, every answer set of $(\Pi_1 * \ldots * \Pi_n)$ is selected among the ones of $(\Pi_1 * \ldots * \Pi_n)^d$.

3.2 Properties of Updates

Different instantiations of our framework yield different properties. To this end, we examine some properties proposed by Eiter *et al.* [9]. We focus below on the slightly more elaborate operators $*_1$ and $*_2$. For comparison, we also mention whether a property at hand is satisfied by the update operation defined by Eiter *et al.* [9], which we denote by $\circ_e$ (the operator $\circ_e$ is formally defined in Section 4).

The first property is the following:[8]

Initialisation: $\emptyset * \Pi \equiv \Pi$. (Fulfilled by $\circ_e$.)

While this property holds for $*_2$ over all preference strategies, it is not satisfied by $*_1$, no matter which preference-handling strategy is used. To see this, consider the program $\{a \leftarrow, \neg a \leftarrow not\ a\}$. While $\emptyset *_2 \Pi = \Pi$ and hence $\emptyset *_2 \Pi \equiv \Pi$, we get

$$\emptyset *_1 \Pi = \Pi^d = \{a \leftarrow not\ \neg a, \neg a \leftarrow not\ a\} \neq \Pi.$$

Π has the single answer set $\{a\}$, but Π^d admits two answer sets, $\{a\}$ and $\{\neg a\}$.

A similar situation is encountered when regarding the following property:

Idempotency: $\Pi * \Pi \equiv \Pi$. (Fulfilled by $\circ_e$.)

For analogous reasons as above, $*_1$ fails to satisfy this property, while it is satisfied by $*_2$, whenever Π has consistent answer sets (see below).

[8] Henceforth we understand a property to hold for all strategies unless otherwise mentioned.

In fact, despite the lack of the previous two properties, $*_1$ yields only consistent answer sets, even if an update is inconsistent. For instance, in a variation of one of the above examples, updating program $\Pi_1 = \{a \leftarrow\}$ by $\Pi_2 = \{b \leftarrow, \neg b \leftarrow\}$ yields

$$\Pi_1 *_1 \Pi_2 = \Pi_1^d \cup \Pi_2^d = \{a \leftarrow \mathit{not}\ \neg a\} \cup \{b \leftarrow \mathit{not}\ \neg b, \neg b \leftarrow \mathit{not}\ b\},$$

from which we obtain two answer sets, $\{a, b\}$ and $\{a, \neg b\}$. Unlike this, $\Pi_1 *_2 \Pi_2 = \Pi_1 \cup \Pi_2$ has the inconsistent answer set $\mathcal{L}$ (as in the above example).

Another property deals with the addition of tautologies (or more generally, the influence of redundant information).

Tautology: If $\mathit{head}(r) \in \mathit{body}^+(r)$, for all $r \in \Pi_2$, then $\Pi_1 * \Pi_2 \equiv \Pi_1$. (Violated by $\circ_e$.)

This property is violated by most update approaches in the literature. For example, let us update $\Pi_1 = \{a \leftarrow \mathit{not}\ \neg a, \neg a \leftarrow\}$ by $\Pi_2 = \{a \leftarrow a\}$. No matter which of the above update operators we take, we obtain a single answer set $\{a\}$ from the update program, which is generated by rule $a \leftarrow \mathit{not}\ \neg a$ in Π_1. The conclusion of $\neg a$ is prohibited by the single rule in Π_2, taking precedence over the second one in Π_1.

For another example, consider $((\Pi_1 * \Pi_2) * \Pi_3)$ where

$$\Pi_1 = \{a \leftarrow\},\ \Pi_2 = \{\neg a \leftarrow\},\ \Pi_3 = \{a \leftarrow a\}\ .$$

Clearly, $(\Pi_1 * \Pi_2)$ induces a single answer set $\{\neg a\}$ in all of the above approaches, including $\circ_e$. Unlike this, update operation $\circ_e$ results in two answer sets, $\{\neg a\}$ and $\{a\}$, once Π_3 has been added and so does each update operation in our framework when using the preference-handling strategy B. This is different, however, when using strategy D, in which case we obtain only a single answer set $\{\neg a\}$ from $((\Pi_1 * \Pi_2) * \Pi_3)$. It remains for future work to see how the addition of tautologies can be counterbalanced by stronger preference-handling strategies. A general approach to overcome this deficiency is proposed by Alferes *et al.* [19]; it also remains future work whether that technique applies in our framework as well.

The next property deals with iterated updates.

Associativity: $(\Pi_1 * (\Pi_2 * \Pi_3)) \equiv ((\Pi_1 * \Pi_2) * \Pi_3)$. (Violated by $\circ_e$.)[9]

This property holds for all instances of our framework. In fact, one can show that both updates yield the same update programs.

Absorption: If $\Pi_2 = \Pi_3$, then $((\Pi_1 * \Pi_2) * \Pi_3) \equiv (\Pi_1 * \Pi_2)$. (Fulfilled by $\circ_e$.)

This property is also satisfied by all instances of our framework. This is also the case with the following generalisation of absorption:

Augmentation: If $\Pi_2 \subseteq \Pi_3$, then $((\Pi_1 * \Pi_2) * \Pi_3) \equiv (\Pi_1 * \Pi_3)$. (Violated by $\circ_e$.)

[9] Strictly speaking, in the approach of Eiter *et al.* [9], the associativity principle is formulated not in terms of the update operation itself, but in terms of the associated update program $P_\triangleleft$ (see Section 4); the same applies for the disjointness and parallelism properties below.

Table 1. Summary of update properties.

	$*_1$	$*_2$	$\circ_e$
Initialisation		$\surd$	$\surd$
Idempotency		$\surd$	$\surd$
Tautology			
Associativity	$\surd$	$\surd$	
Absorption	$\surd$	$\surd$	$\surd$
Augmentation	$\surd$	$\surd$	
Disjointness	$\surd$	$\surd$	$\surd$
Non-Interference	$\surd$	$\surd$	$\surd$

The next property captures update with disjoint programs.

Disjointness: If $atom(\Pi_1) \cap atom(\Pi_2) = \emptyset$, then $(\Pi_1 \cup \Pi_2) * \Pi_3 \equiv (\Pi_1 * \Pi_3) \cup (\Pi_2 * \Pi_3)$. (Fulfilled by $\circ_e$.)

This principle is satisfied by all instances of our framework.

The next property is a variant of the previous.

Parallelism: If $atom(\Pi_2) \cap atom(\Pi_3) = \emptyset$, then $\Pi_1 * (\Pi_2 \cup \Pi_3) \equiv (\Pi_1 * \Pi_2) \cup (\Pi_1 * \Pi_3)$. (Violated by $\circ_e$.)

This property does not hold in our approach. To see this, let $\Pi_3 = \emptyset$. Clearly, we obtain different results from $\Pi_1 * \Pi_2$ and $(\Pi_1 * \Pi_2) \cup \Pi_1$. Arguably, given this example, unrestricted parallelism is not a desirable property.

The last property deals with commutativity when dealing with non-interacting update programs.

Non-Interference: If $atom(\Pi_2) \cap atom(\Pi_3) = \emptyset$, then $(\Pi_1 * \Pi_2) * \Pi_3 \equiv (\Pi_1 * \Pi_3) * \Pi_2$. (Fulfilled by $\circ_e$.)

This property is satisfied by all instances of our framework.

These properties are summarised in Table 1 (with the exception of parallelism, which we feel is undesirable). We note that $*_2$ satisfies the most properties, followed by $\circ_e$, and then $*_1$.

Up to now, we have ignored the treatment of integrity constraints (cf. Baral [1]) in updating logic program. In this respect, we simply follow the approach taken by Eiter *et al.* [9] by handling them as global constraints that are discarded in the defaultification and preference-handling process. Updating a program Π_1 by the program Π_2 in the presence of integrity constraints Π_c then amounts to computing the order-preserving answer sets of $(\Pi_1 * \Pi_2) \cup \Pi_c$. Although we do not detail it here, we mention that our approach allows for accommodating the update of integrity constraints just as well by making them subject to an appropriately adapted defaultification and preference-handling mechanism.

4 Examples and Properties vis-à-vis $\circ_e$

In what follows, we discuss some examples comparing the present update approach with the update approach due to Eiter *et al.* [9]. For simplicity, we focus on our operator $*_1$ under the weakest (reasonable) preference handling strategy, B, given that it is closest to $\circ_e$ (cf. Theorem 3 below).

In the approach of Eiter *et al.* [9], the semantics of an n-fold update $\Pi_1 \circ_e \cdots \circ_e \Pi_n$ is given by the semantics of an (ordinary) program $P_\lhd$, for $P = (\Pi_1, \ldots, \Pi_n)$, containing the following elements:

1. all integrity constraints in Π_i, $1 \leq i \leq n$;
2. for each $r \in \Pi_i$, $1 \leq i \leq n$:
$$l_i \leftarrow \mathit{body}(r), \mathit{not}\ \mathit{rej}(r), \quad \text{where } \mathit{head}(r) = l;$$
3. for each $r \in \Pi_i$, $1 \leq i < n$:
$$\mathit{rej}(r) \leftarrow \mathit{body}(r), \neg l_{i+1}, \quad \text{where } \mathit{head}(r) = l;$$
4. for each literal l occurring in P $(1 \leq i < n)$:
$$l_i \leftarrow l_{i+1}; \qquad\qquad l \leftarrow l_1.$$

Here, for each rule r, $\mathit{rej}(r)$ is a new atom not occurring in $\Pi_1, \ldots, \Pi_n$. Intuitively, $\mathit{rej}(r)$ expresses that r is "rejected." Likewise, each l_i, $1 \leq i \leq n$, is a new atom not occurring in $\Pi_1, \ldots, \Pi_n$. Then, answer sets of $\Pi_1 \circ_e \cdots \circ_e \Pi_n$ are given by the answer sets of $P_\lhd$, modulo the original language. This is similar to compiling ordered logic programs to standard ones as done in our previous work [15].

Example 1. Consider the following programs:

$$\begin{aligned} \Pi_1 &= \{r_1 : \neg a \leftarrow\}, \\ \Pi_2 &= \{r_2 : a \leftarrow b, \mathit{not}\ \neg a\}, \\ \Pi_3 &= \{r_3 : b \leftarrow\}. \end{aligned}$$

The program Π_1 has a single answer set, namely $\{\neg a\}$. In updating Π_1 by Π_2, nothing changes because $r_2^d = r_2$ is not applicable (b is not derivable). A further update by Π_3 changes this situation: b becomes derivable and r_2^d can be applied. In fact, since $r_1^d < r_2^d$, rule r_2^d must be applied before r_1^d and so r_1^d is defeated. Thus, $\{a, b\}$ is the single answer set of $*_1 P$, for $P = (\Pi_1, \Pi_2, \Pi_3)$. Observe that $\{a, b\}$ is of course also an answer set of the unordered program $\Pi_1^d \cup \Pi_2^d \cup \Pi_3^d$, together with $\{\neg a, b\}$. In fact, the latter set is the unique answer set of $\Pi_1 \cup \Pi_2 \cup \Pi_3$, which shows that answer sets of update programs $*P$ are not selected among answer sets of the union of the constituents of P, but rather of the union of the defaultification of its constituents (cf. Theorem 2). Note, however, that $\Pi_1 \circ_e \Pi_2 \circ_e \Pi_3$ has both $\{a, b\}$ and $\{\neg a, b\}$ as answer sets. $\diamondsuit$

The operations $*_1$ and $\circ_e$ also yield different results on inconsistent programs. Consider:

$$\begin{aligned}
&\Pi_1 = \{b \leftarrow,\ \neg b \leftarrow\},\\
&\Pi_2 = \{a \leftarrow\},\\
&\Pi_3 = \{b \leftarrow\},
\end{aligned}$$

and let $P = (\Pi_1, \Pi_2)$. $P_\triangleleft$ has the set of all literals as its unique (inconsistent) answer set, but $*_1 P$ has $\{a, b\}$ and $\{a, \neg b\}$ as answer sets. On the other hand, both $\Pi_1 \circ_e \Pi_2 \circ_e \Pi_3$ and $\Pi_1 *_1 \Pi_2 *_1 \Pi_3$ have $\{a, b\}$ as unique answer set.

The same holds for programs which may become inconsistent due to new information:

$$\begin{aligned}
&\Pi_1 = \{b \leftarrow a,\ \neg b \leftarrow a\},\\
&\Pi_2 = \{a \leftarrow\},\\
&\Pi_3 = \{\neg a \leftarrow\}.
\end{aligned}$$

Π_1 has a consistent answer set, but $\Pi_1 *_1 \Pi_2$ has $\{a, b\}$ and $\{a, \neg b\}$ as answer sets, whereas $\Pi_1 \circ_e \Pi_2$ has the set of all literals as unique answer set. For the additional update with Π_3, both approaches yield $\{\neg a\}$ as unique answer set.

In general, we can formulate the following relation between the answer sets of $*_1$ and $\circ_e$:

Theorem 3. *Let $P = (\Pi_1, \ldots, \Pi_n)$ be a sequence of programs such that $P_\triangleleft$ has only consistent answer sets.*
Then, any answer set of $*_1 P$ is also an answer set of $\Pi_1 \circ_e \cdots \circ_e \Pi_n$.

The converse does not hold, as Example 1 illustrates. Actually, there is an even simpler counterexample: consider

$$\Pi_1 \;=\; \{a \leftarrow\} \quad \text{and} \quad \Pi_2 \;=\; \{\neg a \leftarrow \mathit{not}\ a\}.$$

Then, $\Pi \circ_e \Pi_2$ has two answer sets, viz. $\{a\}$ and $\{\neg a\}$, while $\Pi_1 *_1 \Pi_2$ has only $\{\neg a\}$ as answer set. Actually, $\{\neg a\}$ is the only answer set of $\Pi_1 *_1 \Pi_2$ under any of the three preference strategies B, D, and W.

Finally, let us consider two examples on updating logic programs that have been discussed in the literature, showing that $*_1$ and $\circ_e$ behave the same in these cases.

Example 2 (Adapted from [5]). Consider the update of Π_1 by Π_2, where

$$\begin{aligned}
\Pi_1 = \{\ &r_1 : \ \mathit{sleep} \leftarrow \mathit{not}\ \mathit{tv\_on}, \quad r_2 : \ \mathit{night} \leftarrow\ ,\\
&r_3 : \ \mathit{tv\_on} \leftarrow, \quad r_4 : \ \mathit{watch\_tv} \leftarrow \mathit{tv\_on}\ \},\\
\Pi_2 = \{\ &r_5 : \ \neg \mathit{tv\_on} \leftarrow \mathit{power\_failure},\ r_6 : \ \mathit{power\_failure} \leftarrow\ \}.
\end{aligned}$$

The single answer set of both $\Pi_1 *_1 \Pi_2$ and $\Pi_1 \circ_e \Pi_2$ is

$$S = \{\mathit{power\_failure}, \neg \mathit{tv\_on}, \mathit{sleep}, \mathit{night}\}.$$

If new information arrives as program Π_3, given by

$$\Pi_3 = \{\ r_7 :\ \neg \mathit{power\_failure} \leftarrow\ \},$$

then again $\Pi_1 *_1 \Pi_2 *_1 \Pi_3$ and $\Pi_1 \circ_e \Pi_2 \circ_e \Pi_3$ have the unique answer set

$$T = \{\ \neg \mathit{power\_failure}, \mathit{tv\_on}, \mathit{watch\_tv}, \mathit{night}\ \}. \qquad \Diamond$$

Example 3 ([9]). An agent consulting different sources in search of a performance or a final rehearsal of a concert on a given weekend may be faced with the following situation. First, the agent is notified that there is no concert on Friday:

$$\Pi_1 = \big\{\ r_1 :\ \neg \mathit{concert\_friday} \leftarrow\ \big\}.$$

Later on, a second source reports that there is neither a final rehearsal on Friday nor a concert on Saturday:

$$\Pi_2 = \big\{\ r_2 :\ \neg \mathit{final\_rehearsal\_friday} \leftarrow, \quad r_3 :\ \neg \mathit{concert\_saturday} \leftarrow\ \big\}.$$

Finally, the agent is assured that there is a final rehearsal or a concert on Friday and that whenever there is a final rehearsal on Fridays, a concert on Saturday or Sunday follows:

$$\begin{aligned}
\Pi_3 = \big\{ r_4 &:\ \mathit{concert\_friday} \leftarrow \mathit{not}\ \mathit{final\_rehearsal\_friday},\\
r_5 &:\ \mathit{final\_rehearsal\_friday} \leftarrow \mathit{not}\ \mathit{concert\_friday},\\
r_6 &:\ \mathit{concert\_saturday} \leftarrow \mathit{final\_rehearsal\_friday}, \mathit{not}\ \mathit{concert\_sunday},\\
r_7 &:\ \mathit{concert\_sunday} \leftarrow \mathit{final\_rehearsal\_friday}, \mathit{not}\ \mathit{concert\_saturday}\ \big\}.
\end{aligned}$$

The update program $\Pi_1 *_1 \Pi_2 *_1 \Pi_3$ has three answer sets:

$$\begin{aligned}
S_1 &= \{\mathit{final\_rehearsal\_friday}, \neg \mathit{concert\_friday}, \mathit{concert\_saturday}\},\\
S_2 &= \{\mathit{final\_rehearsal\_friday}, \neg \mathit{concert\_friday}, \neg \mathit{concert\_saturday},\\
&\qquad \mathit{concert\_sunday}\},\\
S_3 &= \{\neg \mathit{final\_rehearsal\_friday}, \mathit{concert\_friday}, \neg \mathit{concert\_saturday}\},
\end{aligned}$$

where $\Pi_1 *_1 \Pi_2 *_1 \Pi_3$ is given as follows:

$$\begin{aligned}
&\Pi_1^d \cup \Pi_2^d \cup \Pi_3^d =\\
&\big\{\ r_1^d :\ \neg \mathit{concert\_friday} \leftarrow \mathit{not}\ \mathit{concert\_friday},\\
&\quad r_2^d :\ \neg \mathit{final\_rehearsal\_friday} \leftarrow \mathit{not}\ \mathit{final\_rehearsal\_friday},\\
&\quad r_3^d :\ \neg \mathit{concert\_saturday} \leftarrow \mathit{not}\ \mathit{concert\_saturday},\\
&\quad r_4^d :\ \mathit{concert\_friday} \leftarrow \mathit{not}\ \mathit{final\_rehearsal\_friday}, \mathit{not}\ \neg \mathit{concert\_friday},\\
&\quad r_5^d :\ \mathit{final\_rehearsal\_friday} \leftarrow \mathit{not}\ \mathit{concert\_friday},\\
&\qquad\qquad\qquad\qquad\qquad \mathit{not}\ \neg \mathit{final\_rehearsal\_friday},\\
&\quad r_6^d :\ \mathit{concert\_saturday} \leftarrow \mathit{final\_rehearsal\_friday}, \mathit{not}\ \mathit{concert\_sunday},\\
&\qquad\qquad\qquad\qquad\qquad \mathit{not}\ \neg \mathit{concert\_saturday},\\
&\quad r_7^d :\ \mathit{concert\_sunday} \leftarrow \mathit{final\_rehearsal\_friday}, \mathit{not}\ \mathit{concert\_saturday},\\
&\qquad\qquad\qquad\qquad\qquad \mathit{not}\ \neg \mathit{concert\_sunday}\ \big\},
\end{aligned}$$

together with the following order:

$$r_1^d < r_4^d, \quad r_2^d < r_5^d, \quad \text{and} \quad r_3^d < r_6^d.$$

The same answer sets are obtained in case of $\mathit{\Pi}_1 \circ_e \mathit{\Pi}_2 \circ_e \mathit{\Pi}_3$. $\Diamond$

5 Discussion

We have presented a simple and general framework to updating logic programs under the answer-set semantics. Our approach is based on two well-known techniques in ASP: defaultification and preference handling. Depending on how we fix the interplay among both techniques, we obtain distinct update operations. An interesting feature is that inconsistencies are removed by defaultification and can subsequently be resolved through preferences. Also, the approach is iterative and the history of the continued updates is (mainly[10]) captured by preferences rather than explicit program transformations. Another advantage of this approach is that it is easily implementable by off-the-shelf systems developed for ASP.

More elaboration of the different instances of our framework is desirable. In particular, it will be interesting to see how the properties of the framework change with different ordering mechanisms, in particular, ones that are especially designed for update purposes. Another interesting question is in how far our framework is suitable as a uniform approach in which other approaches can be simulated. As well, although our focus has been on the development of a general framework, it appears that the approaches under strategy $*_2$ have good properties (as evidenced by our survey of properties, as well as the examples presented) and warrant fuller investigation in their own right.

Given the numerous approaches to updating logic programs, among them [2–10], a detailed comparison to the literature is beyond the scope of this paper. An excellent survey is given by Eiter *et al.* [9]. We have already compared our approach in some detail to the one of Eiter *et al.* [9]. We summarise here differences with some particularly related approaches. Alferes *et al.* [8] use similar techniques for combining update operations with preferences. However, in contrast to our approach, preferences are not used to implement updates but rather as an additional means for guiding the update. Zhang and Foo [4] were first in mapping update operations onto preference handling in answer-set programming. Unlike us, however, their approach is based on the elimination of conflicting rules and thus on rewriting logic programs. Furthermore, the resulting update program is given in terms of a set of programs, which prohibits iterated update operations as well as a comparison in view of the postulates discussed in Section 3. Interesting recent work [10] gives a framework for characterising update approaches in terms of the notion of *forgetting*.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)

[10] Operator $*_2$ necessitates modifications to the program whenever new conflicts arise.

2. Foo, N., Zhang, Y.: Towards generalized rule-based updates. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97). Volume 1. Morgan Kaufmann (1997) 82–88
3. Przymusinski, T., Turner, H.: Update by means of inference rules. Journal of Logic Programming **30** (1997) 125–143
4. Zhang, Y., Foo, N.Y.: Updating logic programs. In: Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI'98). (1998) 403–407
5. Alferes, J., Leite, J., Pereira, L., Przymusinska, H., Przymusinski, T.: Dynamic updates of non-monotonic knowledge bases. Journal of Logic Programming **45** (2000) 43–70
6. Alferes, J., Pereira, L., Przymusinska, H., Przymusinski, T.: LUPS - A language for updating logic programs. In: Proceedings of the Fifth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'99). Volume 1730 of Lecture Notes in Artificial Intelligence, Springer (1999) 162–176
7. Inoue, K., Sakama, C.: 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, Springer (1999) 147–161
8. Alferes, J.J., Dell'Acqua, P., Pereira, L.M.: A compilation of updates plus preferences. In: Proceedings of the Eighth International Conference on Logics in Artificial Intelligence (JELIA 2002). Volume 2424 of Lecture Notes in Computer Science, Springer (2002) 62–73
9. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: On properties of update sequences based on causal rejection. Theory and Practice of Logic Programming **2** (2002) 711–767
10. Zhang, Y., Foo, N.: A unified framework for representing logic program updates. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005), AAAI Press/The MIT Press (2005) 707–713
11. Winslett, M.: Reasoning about action using a possible models approach. In: Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88), AAAI Press/The MIT Press (1988) 89–93
12. Sadri, F., Kowalski, R.: A theorem-proving approach to database integrity. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publishers (1987) 313–362
13. Nieuwenborgh, D.V., Vermeir, D.: Preferred answer sets for ordered logic programs. Theory and Practice of Logic Programming **6** (2006) 107–167
14. Brewka, G., Eiter, T.: Preferred answer sets for extended logic programs. Artificial Intelligence **109** (1999) 297–356
15. Delgrande, J., Schaub, T., Tompits, H.: A framework for compiling preferences in logic programs. Theory and Practice of Logic Programming **3** (2003) 129–187
16. Schaub, T., Wang, K.: A semantic framework for preference handling in answer set programming. Theory and Practice of Logic Programming **3** (2003) 569–607
17. Eiter, T., Faber, W., Leone, N., Pfeifer, G.: Computing preferred answer sets by meta-interpretation in answer set programming. Theory and Practice of Logic Programming **3** (2003) 463–498
18. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Proceedings of the Seventh International Conference on Logic Programming (ICLP'90), MIT Press (1990) 579–597
19. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: Semantics for dynamic logic programming: A principle-based approach. In: Proceedings of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004). Volume 2923 of Lecture Notes in Computer Science, Springer (2004) 8–20