On Properties of Update Sequences Based on Causal Rejection

THOMAS EITER, MICHAEL FINK,
GIULIANA SABBATINI, and HANS TOMPITS
Technische Universität Wien,
Institut für Informationssysteme,
Abt. Wissensbasierte Systeme 184/3,
Favoritenstraße 9-11, A-1040 Vienna, Austria
(*e-mail:* `[eiter,michael,giuliana,tompits]@kr.tuwien.ac.at`)

Abstract

In this paper, we consider an approach to update nonmonotonic knowledge bases represented as extended logic programs under the answer set semantics. In this approach, new information is incorporated into the current knowledge base subject to a causal rejection principle, which enforces that, in case of conflicts between rules, more recent rules are preferred and older rules are overridden. Such a rejection principle is also exploited in other approaches to update logic programs, notably in the method of dynamic logic programming, due to Alferes *et al.*

One of the central issues of this paper is a thorough analysis of various properties of the current approach, in order to get a better understanding of the inherent causal rejection principle. For this purpose, we review postulates and principles for update and revision operators which have been proposed in the area of theory change and nonmonotonic reasoning. Moreover, some new properties for approaches to updating logic programs are considered as well. Like related update approaches, the current semantics does not incorporate a notion of *minimality of change*, so we consider refinements of the semantics in this direction. As well, we investigate the relationship of our approach to others in more detail. In particular, we show that the current approach is semantically equivalent to inheritance programs, which have been independently defined by Buccafurri *et al.*, and that it coincides with certain classes of dynamic logic programs. In view of this analysis, most of our results about properties of the causal rejection principle apply to each of these approaches as well. Finally, we also deal with computational issues. Besides a discussion on the computational complexity of our approach, we outline how the update semantics and its refinements can be directly implemented on top of existing logic programming systems. In the present case, we implemented the update approach using the logic programming system `DLV`.

1 Introduction

1.1 Motivation and Context

Logic programming has been conceived as a computational logic paradigm for problem solving and offers a number of advantages over conventional programming languages. In particular, it is a well-suited tool for declarative knowledge representation and common-sense reasoning (Baral & Gelfond, 1994), and possesses thus a high potential as a key technology to equip software agents with advanced reasoning capabilities in order to make those agents behave intelligently (cf., e.g., (Sadri & Toni, 2000)).

It has been realized, however, that further work is needed on extending the current methods and techniques to fully support the needs of agents. In a simple (but, as for currently deployed agent systems, realistic) setting, an agent's knowledge base, KB, may be modeled as a logic program, which the agent may evaluate to answer queries that arise. Given various approaches to semantics, the problem of evaluating a logic program is quite well-understood, and (beside Prolog) provers for semantics with more sophisticated treatment of negation may be used. Currently available provers include the systems `DeRes` (Cholewiński *et al.*, 1996), `DLV` (Eiter *et al.*, 1997a), `smodels` (Niemelä & Simons, 1996), and `XSB` (Rao *et al.*, 1997).

An important aspect, however, is that an agent is situated in an environment which is subject to change. This requests the agent to adapt over time, and to adjust its decision making. An agent might be prompted to adjust its knowledge base KB after receiving new information in terms of an *update* U, given by a clause or a set of clauses that need to be incorporated into KB. Simply adding the rules of U to KB does not give a satisfactory solution in practice, even in simple cases. For example, if KB contains the rules $a \leftarrow b$ and $b \leftarrow$, and U consists of the rule $\neg a \leftarrow$ stating that a is false, then the union $KB \cup U$ is not consistent under predominant semantics such as the answer set semantics (Gelfond & Lifschitz, 1991) or the well-founded semantics (Van Gelder *et al.*, 1991). However, by attributing higher priority to the update $\neg a \leftarrow$, a result is intuitively expected which has a consistent semantics, where the emerging conflict between old and new information is resolved.

To address this problem, some approaches for updating logic programs with (sets of) rules have been proposed recently (Alferes *et al.*, 1998; Alferes *et al.*, 1999; Inoue & Sakama, 1999; Zhang & Foo, 1998). In this paper, we consider an approach which is based on a *causal rejection principle*. According to this principle, a rule r is only discarded providing there is a "reason" for doing so, in terms of another, more recent rule r' which contradicts r. That is, if both r and r' are applicable (i.e., their bodies are satisfied) and have opposite heads, then only r' is applied while r is discarded. In the example from above, the rule $r: \ a \leftarrow b$ in the current knowledge base KB (whose body is true given rule $b \leftarrow$) is rejected by the new rule $r': \ \neg a \leftarrow$ in the update (whose body is also true), and thus in the updated knowledge base, r is not applied.

The causal rejection principle is not novel—in fact, it constitutes a major ingredient of the well-known dynamic logic programming approach (Alferes *et al.*, 1998; Alferes *et al.*, 2000). Furthermore, it underlies, in slightly different forms, the related approaches of inheritance logic programs (Buccafurri *et al.*, 1999a) and ordered logic programs (Laenens *et al.*, 1990; Buccafurri *et al.*, 1996). We provide here a simple and rigorous realization of this principle, in terms of "founded" rejection: a rule r may only be rejected by some other rule r' *which itself is not rejected.* While this foundedness condition, as it appears, plays in effect no role in the particular semantics we consider, it can do so for more involved semantics based on causal rejection, such as the one by Alferes *et al.* (1998; 2000).

Starting from a simple formalization of a semantics for updating logic programs based on causal rejection, which offers the advantage of a clear declarative characterization and of a syntactical realization at the same time, the main goal of this paper is to investigate properties of this semantics, as well as to analyze the relationship to other semantics for updating logic programs, in particular to dynamic logic programming. Notice that, although uses and extensions of dynamic logic programming have been discussed (cf. (Alferes *et al.*, 1999; Alferes & Pereira, 2000; Leite *et al.*, 2000)), its properties and relations to other approaches and related formalisms have been less explored so far (but see (Alferes & Pereira, 2000)).

1.2 Main Contributions

Inspired by ideas in (Alferes *et al.*, 1998; Alferes *et al.*, 2000), we consider a semantics for sequences $\boldsymbol{P} = (P_1, \ldots, P_n)$ of extended logic programs, in terms of a syntactic transformation to an *update program*, which is a single extended logic program in an extended language. The semantics properly generalizes the answer set semantics (Gelfond & Lifschitz, 1991) of single logic programs. The readable syntactic representation of the semantical results—which is useful from a computational perspective—is complemented, as in (Alferes *et al.*, 1998; Alferes *et al.*, 1999), by an elegant semantical characterizations in terms of a modified Gelfond-Lifschitz reduction, resulting from the usual construction by removal of rejected rules. The transformation we describe is similar to the one by Alferes *et al.*, but involves only a few types of rules and new atoms. For capturing the rejection principle, information about rule rejection is explicitly represented at the object level through rejection atoms; this is similar to an implementation of the related inheritance logic program approach proposed by Buccafurri *et al.* (1999a). Though not new in spirit, the approach we suggest offers a more accessible definition and is suitable for studying general properties of updates by causal rejection, providing insight in the mechanism of the rejection principle itself.

The main contributions of this paper can be summarized as follows.

(1) We extensively investigate, from different points of view, properties of update programs and answer set semantics for update sequences. We first analyze them from a belief revision perspective, and evaluate various (sets of) postulates for revision and iterated revision from the literature (Alchourrón *et al.*, 1985; Katsuno & Mendelzon, 1991; Darwiche & Pearl, 1997; Lehmann, 1995). To this end, we discuss possible interpretations of update programs as change operators for nonmonotonic logical theories. As it turns out, update programs (and thus equivalent approaches) do not satisfy many of the properties defined in the literature. This is partly explained by the nonmonotonicity of logic programs and the causal rejection principle embodied in the semantics, which strongly depends on the syntax of rules.

Furthermore, we consider properties from a nonmonotonic reasoning perspective, by naturally interpreting update programs as nonmonotonic consequence relations, and review postulates and principles which have been analyzed by Kraus, Lehmann, and Magidor (1990), and Makinson (1993).

Finally, we present and discuss some further general properties relevant for update programs. Among them is an *iterativity property*, which informally states equivalence of nesting $((P_1, P_2), P_3)$ and sequences (P_1, P_2, P_3) of updates. A possible interpretation of this property is that an *immediate update strategy*, which incorporates new information immediately into the knowledge base, is equivalent to *demand-driven evaluation*, where the actual knowledge base KB is built on demand of particular queries, and full information about KB's update history is known. As we shall see, the property does not hold in general, but for certain classes of programs.

(2) As it appears, update answer sets—like related concepts based on causal rejection—do not respect minimality of change. We thus refine the semantics of update sequences and introduce *minimal answer sets* and *strictly minimal answer sets*. Informally, in minimal answer sets, the set of rules that need to be rejected is minimized. This means that a largest set of rules should be respected if an answer set is built; in particular, if all rules can be satisfied, then no answer sets would be adopted, which request the rejection of any rule. The notion of strict minimality further refines minimality by enforcing that rejection of older rules should be preferred to rejection of newer rules, thus performing hierarchic minimization.

The refined semantics come at the cost of higher computational complexity, and increase the complexity of update answer sets for propositional programs by one level, namely from the first to the second level in the polynomial hierarchy. This parallels similar results for the update semantics by Sakama and Inoue (1999), which employs a notion of minimality in the basic definition.

(3) We conduct a comparison between update programs and alternative approaches for updating logic programs (Alferes *et al.*, 1998; Alferes *et al.*, 2000; Zhang & Foo, 1998; Inoue & Sakama, 1999; Leite & Pereira, 1997; Leite, 1997; Marek & Truszczyński, 1994) and related approaches (Buccafurri *et al.*, 1999a; Delgrande *et al.*, 2000). We find that for some of these formalisms, syntactic subclasses are semantically equivalent to update programs. Thus, update programs provide a (different) characterization of these fragments, and by their simplicity, contribute to better understanding on the essential working of these formalisms on these fragments. Furthermore, our results on properties of update answer set semantics carry over to the equivalent fragments, and establish also semantical results for these formalisms, which have not been analyzed much in this respect so far. Finally, equivalent fragments of different formalisms are identified via update programs.

First, we show that update programs are, on the language we consider, equivalent to inheritance logic programs. More precisely, our notion of an answer set for an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ coincides with the notion of an answer set for a corresponding inheritance program $P^<$ in the approach by Buccafurri *et al.* (1999a), where $P^<$ results from $\boldsymbol{P}$ by interpreting more recent updates in the sequence $(P_1, \ldots, P_n)$ (i.e., programs with higher index) as programs containing more specific information. Thus, update programs (and classes of dynamic logic programs) may semantically be regarded as fragment of the inheritance framework of Buccafurri *et al.* (1999a). We then compare our update programs to revision programming by Marek and Truszczyński (1994) and the related approach of Leite and Pereira (1997), which has been extended to sequences of programs in (Leite, 1997). It appears that the fragment of this formalism where programs merely use weak negation is, apart from extra conditions on sequences of more than two programs, semantically stronger than update programs. Furthermore, we give a thorough analysis of the dynamic logic programming approach by Alferes *et al.* (1998; 2000). Their notion of model of an update sequence $\boldsymbol{P}$, which we refer to as *dynamic answer set*, semantically imposes extra conditions compared to our update answer set.[1] Note that syntactic conditions for classes of programs can be found on which dynamic answer sets and update answer sets coincide. Furthermore, by this correspondence, some results for update principles and computational complexity derived for our update programs carry over to dynamic logic programs as well. Further inspection, which we do not carry out here, suggests the same results beyond the corresponding fragments.

To the best of our knowledge, no investigation of approaches to updating logic programs from the perspectives of belief revision and nonmonotonic consequences relations has been carried out so far. In view of our results about the relationship between update programs and other approaches, in particular to inheritance logic programs and fragments of dynamic logic programming, our investigations apply to these formalisms as well.

1.3 Structure of the Paper

The paper is organized as follows. After providing some necessary preliminaries in the next section, we introduce in Section 3 update programs and answer sets for such programs, and establish some characterization results. In Section 4, we embark on our study of general principles of update programs based on causal rejection from various perspectives. The refinements of answer sets to minimal and strictly minimal answer sets are considered in Section 5. Section 6 is devoted to computational issues of our approach. After an investigation of the computational complexity of update programs under the semantics introduced, we discuss an implementation of our approach based on the `DLV` logic programming tool (Eiter *et al.*, 1997a; Eiter *et al.*, 1998). In Section 7, relations to other and related approaches are investigated. The paper concludes with Section 8, containing a short summary and a discussion of further work and open issues. Some proofs and further results, which are omitted here for space reasons, can be found in (Eiter *et al.*, 2000b).

[1] In a preliminary version of this paper (Eiter *et al.*, 2000a), we erroneously reported, due to misunderstanding notation in (Alferes *et al.*, 1998; Alferes *et al.*, 2000), that dynamic logic programs and update programs are equivalent in general. This view was supported by the examples discussed in (Alferes *et al.*, 1998; Alferes *et al.*, 2000) and many others we considered.

2 Preliminaries

We deal with extended logic programs (Gelfond & Lifschitz, 1991), which consist of rules built over a set $\mathcal{A}$ of propositional atoms where both default negation *not* and strong negation $\neg$ is available. A *literal*, L, is either an atom A (a *positive literal*) or a strongly negated atom $\neg A$ (a *negative literal*). For a literal L, the *complementary literal*, $\neg L$, is $\neg A$ if $L = A$, and A if $L = \neg A$, for some atom A. For a set S of literals, we define $\neg S = \{\neg L \mid L \in S\}$, and denote by $Lit_{\mathcal{A}}$ the set $\mathcal{A} \cup \neg\mathcal{A}$ of all literals over $\mathcal{A}$. A literal preceded *not* is called a *weakly negated literal*.

A *rule*, r, is an ordered pair $L_0 \leftarrow B(r)$, where L_0 is a literal and $B(r)$ is a finite set of literals or weakly negated literals. We also allow the case where L_0 may be absent. We call L_0 the *head* of r, denoted $H(r)$, and $B(r)$ the *body* of r. For $B(r) = \{L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n\}$, we define $B^+(r) = \{L_1, \ldots, L_m\}$ and $B^-(r) = \{L_{m+1}, \ldots, L_n\}$. The elements of $B^+(r)$ are referred to as the *prerequisites* of r. We employ the usual conventions for writing rules like $L_0 \leftarrow B_1 \cup B_2$ or $L_0 \leftarrow B_1 \cup \{L\}$ as $L_0 \leftarrow B_1, B_2$ and $L_0 \leftarrow B_1, L$, respectively. Generally, rule r with $B(r)$ as above will simply be written as

$$L_0 \leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n.$$

If r has an empty head, then r is a *constraint*; if the body of r is empty, then r is a *fact*; if $n = m$ (i.e., if r contains no default negation), then r is a *basic rule*. We denote by $\mathcal{L}_{\mathcal{A}}$ the set of all rules constructible using the literals in $Lit_{\mathcal{A}}$.

An *extended logic program* (ELP), P, is a (possibly infinite) set of rules. If all rules in P are basic, then P is a *basic program*. Usually, $\mathcal{A}$ will simply be understood as the set of *all* atoms occurring in P.

An *interpretation* I is a set of literals which is *consistent*, i.e., I does not contain complementary literals A and $\neg A$. A literal L is *true* in I (symbolically $I \models L$) iff $L \in I$, and *false* otherwise. Given a rule r, the body $B(r)$ of r is true in I, denote $I \models B(r)$, iff (i) each $L \in B^+(r)$ is true in I and (ii) each $L \in B^-(p)$ is false in I. Rule r is true in I, denoted $I \models r$, iff $H(r)$ is true in I whenever $B(r)$ is true in I. In particular, a constraint r is true in I iff $I \not\models B(r)$. For a program P, I is a *model* of P, denoted $I \models P$, if $I \models r$ for all $r \in P$.

Let r be a rule. Then r^+ denotes the basic rule obtained from r by deleting all weakly negated literals in the body of r, i.e., $r^+ = H(r) \leftarrow B^+(r)$. Furthermore, we say that rule r is *defeated* by a set of literals S if some literal in $B^-(r)$ is true in S, i.e., if $B^-(r) \cap S \neq \emptyset$. As well, each literal in $B^-(r) \cap S$ is said to *defeat* r.

The *reduct*, P^S, of a program P *relative to* a set S of literals is defined by

$$P^S = \{r^+ \mid r \in \Pi \text{ and } r \text{ is not defeated by } S\}.$$

An interpretation I is an *answer set* of a program P iff it is a minimal model of P^I. By $\mathcal{S}(P)$ we denote the collection of all answer sets of P. If $\mathcal{S}(P) \neq \emptyset$, then P is said to be *satisfiable*.

We regard a logic program P as the *epistemic state* of an agent. The given semantics is used for assigning a *belief set* to any epistemic state P as follows.

Let $I \subseteq Lit_{\mathcal{A}}$ be an interpretation. Define

$$Bel_{\mathcal{A}}(I) = \{r \in \mathcal{L}_{\mathcal{A}} \mid I \models r\}.$$

Furthermore, for a class $\mathcal{I}$ of interpretations, let $Bel_{\mathcal{A}}(\mathcal{I}) = \bigcap_{I \in \mathcal{I}} Bel_{\mathcal{A}}(I)$.

Definition 1
For a logic program P, the belief set, $Bel_{\mathcal{A}}(P)$, of P is given by $Bel_{\mathcal{A}}(P) = Bel_{\mathcal{A}}(\mathcal{S}(P))$.

We write $P \models_{\mathcal{A}} r$ if $r \in Bel_{\mathcal{A}}(P)$, and for any program Q, we write $P \models_{\mathcal{A}} Q$ if $P \models_{\mathcal{A}} q$ for all $q \in Q$. Programs P_1 and P_2 are *equivalent* (modulo $\mathcal{A}$), symbolically $P_1 \equiv_{\mathcal{A}} P_2$, iff $Bel_{\mathcal{A}}(P_1) = Bel_{\mathcal{A}}(P_2)$. It can be seen that if either P_1 or P_2 involves only finitely many atoms, or if $\mathcal{A}$ is finite, then $P_1 \equiv_{\mathcal{A}} P_2$ is equivalent to the condition that P_1 and P_2 have the same answer sets modulo $\mathcal{A}$. We will drop the subscript "$\mathcal{A}$" in $Bel_{\mathcal{A}}(\cdot)$, $\models_{\mathcal{A}}$, and $\equiv_{\mathcal{A}}$ if no ambiguity can arise.

Belief sets enjoy the following natural properties:

Theorem 1
For every logic program P, we have that:

(i) $P \subseteq Bel(P)$;
(ii) $Bel(Bel(P)) = Bel(P)$;
(iii) $\{r \mid I \models r$, for every interpretation $I\} \subseteq Bel(P)$.

Proof
Properties (i) and (iii) hold trivially. Property (ii) can be seen as follows: $Bel(P) \subseteq Bel(Bel(P))$ follows directly from property (i), and $Bel(Bel(P)) \subseteq Bel(P)$ holds due to the fact that each answer set of P is also an answer set of $Bel(P)$. □

Clearly, the belief operator $Bel(\cdot)$ is nonmonotonic, i.e., in general, $P_1 \subseteq P_2$ does not imply $Bel(P_1) \subseteq Bel(P_2)$.

3 Update programs

We introduce a framework to update logic programs based on a compilation technique to ELPs. The basic idea is the following. Given a sequence $(P_1, \ldots, P_n)$ of ELPs, each P_i is assumed to update the information expressed by the initial section $(P_1, \ldots, P_{i-1})$. The sequence $(P_1, \ldots, P_n)$ is translated into a single ELP P', respecting the successive update information, such that the answer sets of P' represent the answer sets of $(P_1, \ldots, P_n)$. The translation is realized by introducing new atoms $rej(\cdot)$ which control the applicability of rules with respect to the update information[2]. Informally, $rej(r)$ states that rule r is "rejected", in case a more recent rule r' asserts a conflicting information. This conflict is resolved by enabling $rej(r)$ to block the applicability of r, and so rule r' is given precedence over r.

In some sense, the proposed update mechanism can be seen as some form of

[2] This idea can be found elsewhere in the literature, e.g., (Kowalski & Toni, 1996; Inoue, 2000)

an *inheritance strategy*, where more recent rules are viewed as "more specific" information, which have to be given preference in case of a conflict. In Section 7.1, we will discuss the relationship between our update formalism and the inheritance framework introduced by Buccafurri *et al.* (1999a).

The general method of expressing update sequences in terms of single programs has already been discussed by Alferes *et al.* (1998; 2000). However, in that framework, applicability issues are realized in terms of newly introduced atoms referring to the derivability of *atoms* of the original programs, and not to the applicability of *rules* as in the present approach. A detailed comparison between our approach and the method of Alferes *et al.* (1998; 2000) is given in Section 7.3.

3.1 Basic Approach

By an *update sequence*, $\boldsymbol{P}$, we understand a series $(P_1, \ldots, P_n)$ of ELPs. We say that $\boldsymbol{P}$ is an update sequence *over* $\mathcal{A}$ iff $\mathcal{A}$ represents the set of atoms occurring in the rules of the constituting elements P_i of $\boldsymbol{P}$ $(1 \leq i \leq n)$.

Given an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over $\mathcal{A}$, we assume a set $\mathcal{A}^*$ extending $\mathcal{A}$ by new, pairwise distinct atoms $rej(r)$ and A_i, for each r occurring in $\boldsymbol{P}$, each atom $A \in \mathcal{A}$, and each i, $1 \leq i \leq n$. We further assume an injective *naming function* $N(\cdot, \cdot)$, which assigns to each rule r in a program P_i a distinguished name, $N(r, P_i)$, obeying the condition $N(r, P_i) \neq N(r', P_j)$ whenever $i \neq j$. With a slight abuse of notation we shall identify r with $N(r, P_i)$ as usual. Finally, for a literal L, we write L_i to denote the result of replacing the atomic formula A of L by A_i.

Definition 2
Given an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over a set of atoms $\mathcal{A}$, we define the update program $\boldsymbol{P}_{\triangleleft} = P_1 \triangleleft \ldots \triangleleft P_n$ over $\mathcal{A}^*$ consisting of the following items:

(i) all constraints in P_i, $1 \leq i \leq n$;

(ii) for each $r \in P_i$, $1 \leq i \leq n$:

$$L_i \quad \leftarrow \quad B(r), not\ rej(r) \qquad\qquad \text{if } H(r) = L;$$

(iii) for each $r \in P_i$, $1 \leq i < n$:

$$rej(r) \quad \leftarrow \quad B(r), \neg L_{i+1} \qquad\qquad \text{if } H(r) = L;$$

(iv) for each literal L occurring in $\boldsymbol{P}$ $(1 \leq i < n)$:

$$L_i \leftarrow L_{i+1}; \qquad\qquad L \leftarrow L_1.$$

Informally, this program expresses layered derivability of a literal L, beginning at the top layer P_n downwards to the bottom layer P_1. The rule r at layer P_i is only applicable if it is not refuted by a literal derived at a higher level that is incompatible with $H(r)$. Inertia rules propagate a locally derived value for L downwards to the first level, where the local value is made global. The transformation $\boldsymbol{P}_{\triangleleft}$ is modular in the sense that for $\boldsymbol{P'} = (P_1, \ldots, P_n, P_{n+1})$ it augments $\boldsymbol{P}_{\triangleleft} = P_1 \triangleleft \ldots \triangleleft P_n$ only with rules depending on $n + 1$.

We remark that $\boldsymbol{P}_{\triangleleft}$ can obviously be slightly simplified, which is relevant for implementing our approach. All weakly negated literals $not\,rej(r)$ in rules with heads L_n can be removed: Indeed, since $rej(r)$ cannot be derived, each such atom evaluates to false in any answer set of $\boldsymbol{P}_{\triangleleft}$. Thus, no rule from P_n is rejected in an answer set of $\boldsymbol{P}_{\triangleleft}$, i.e., all most recent rules are obeyed.

The intended answer sets of an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ are defined in terms of the answer sets of $\boldsymbol{P}_{\triangleleft}$.

Definition 3
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$. Then, $S \subseteq Lit_{\mathcal{A}}$ is an *update answer set* of $\boldsymbol{P}$ iff $S = S' \cap \mathcal{A}$ for some answer set S' of $\boldsymbol{P}_{\triangleleft}$. The collection of all update answer sets of $\boldsymbol{P}$ is denoted by $\mathcal{U}(\boldsymbol{P})$.

Following the case of single programs, an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ is regarded as the epistemic state of an agent, and the belief set $Bel(\boldsymbol{P})$ is given by $Bel(\mathcal{U}(\boldsymbol{P}))$. The update sequence $\boldsymbol{P}$ is said to be satisfiable iff $\mathcal{U}(\boldsymbol{P}) \neq \emptyset$, and $\boldsymbol{P} \equiv \boldsymbol{P}'$ iff $Bel(\boldsymbol{P}) = Bel(\boldsymbol{P}')$ ($\boldsymbol{P}'$ some update sequence). General properties of the belief operator $Bel(\cdot)$ in the context of update sequences will be discussed in Section 4.

For illustration of Definition 3, consider the following example, adapted from (Alferes *et al.*, 1998).

Example 1
Consider the update of P_1 by P_2, where

$$\begin{aligned} P_1 &= \{\, r_1 : \; sleep \leftarrow not\; tv\_on, \quad r_2 : \; night \leftarrow, \quad r_3 : \; tv\_on \leftarrow, \\ &\qquad r_4 : \; watch\_tv \leftarrow tv\_on \,\}; \\ P_2 &= \{\, r_5 : \; \neg tv\_on \leftarrow power\_failure, \quad r_6 : \; power\_failure \leftarrow \,\}. \end{aligned}$$

The single answer set of $\boldsymbol{P} = (P_1, P_2)$ is, as desired,

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

since the only answer set of $\boldsymbol{P}_{\triangleleft}$ is given by

$$\begin{aligned} S' = \{\, & power\_failure_2, power\_failure_1, power\_failure, \\ & \neg tv\_on_2, \neg tv\_on_1, \neg tv\_on, rej(r_3), sleep_1, sleep, night_1, night \,\}. \end{aligned}$$

If new information arrives in form of the program P_3:

$$P_3 = \{\, r_7 : \; \neg power\_failure \leftarrow \,\},$$

then the update sequence (P_1, P_2, P_3) has the answer set

$$T = \{\, \neg power\_failure, tv\_on, watch\_tv, night \,\},$$

generated by the following answer set T' of $P_1 \triangleleft P_2 \triangleleft P_3$:

$$\begin{aligned} T' = \{\, & \neg power\_failure_3, \neg power\_failure_2, \neg power\_failure_1, \neg power\_failure, \\ & rej(r_6), tv\_on_1, tv\_on, watch\_tv_1, watch\_tv, night_1, night \,\}. \end{aligned}$$

3.2 Properties and Characterizations

Next, we discuss some properties of our approach. The first result guarantees that answer sets of $\boldsymbol{P}$ are uniquely determined by the answer sets of $\boldsymbol{P}_{\triangleleft}$.

Theorem 2

Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$, and let $S, T \subseteq Lit_{\mathcal{A}^*}$ be answer sets of $\boldsymbol{P}_\triangleleft$. Then, $S \cap Lit_{\mathcal{A}} = T \cap Lit_{\mathcal{A}}$ only if $S = T$.

Proof

See Appendix A.1. □

In view of this result, the following notation is well-defined.

Definition 4

Let $\boldsymbol{P}$ be an update sequence over $\mathcal{A}$, and let S be an answer set of $\boldsymbol{P}$. Then, $\check{S}$ denotes the (uniquely determined) answer set of $\boldsymbol{P}_\triangleleft$ obeying $S = \check{S} \cap Lit_{\mathcal{A}}$.

If an update sequence $\boldsymbol{P}$ consists of a single program P_1, the update answer sets of $\boldsymbol{P}$ coincide with the regular answer sets of P_1.

Theorem 3

Let $\boldsymbol{P}$ be an update sequence consisting of a single program P_1, i.e., $\boldsymbol{P} = P_1$. Then, $\mathcal{U}(\boldsymbol{P}) = \mathcal{S}(P_1)$.

Proof

This follows at once from the observation that the only difference between P_1 and $\boldsymbol{P}_\triangleleft$ is that each rule $r = L \leftarrow B(r)$ occurring in P_1 is replaced by the two rules $L_1 \leftarrow B(r), \mathit{not}\ \mathit{rej}(r)$ and $L \leftarrow L_1$. Since there are no rules in $\boldsymbol{P}_\triangleleft$ having head literal $\mathit{rej}(r)$, it holds that, for each set S of literals, r is defeated by S exactly if $L_1 \leftarrow B(r), \mathit{not}\ \mathit{rej}(r)$ is defeated by S. □

Answer sets of update sequences can also be characterized in a purely declarative way. To this end, we introduce the concept of a *rejection set*. Let us call two rules r_1 and r_2 *conflicting* iff $H(r_1) = \neg H(r_2)$. For an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$, based on the principle of founded rule rejection, we define the rejection set of S by $\mathit{Rej}(S, \boldsymbol{P}) = \bigcup_{i=1}^{n} \mathit{Rej}_i(S, \boldsymbol{P})$, where $\mathit{Rej}_n(S, \boldsymbol{P}) = \emptyset$, and, for $n > i \geq 1$,

$$\begin{aligned} \mathit{Rej}_i(S, \boldsymbol{P}) \;=\; \big\{\, r \in P_i \mid \exists r' \in P_j \setminus \mathit{Rej}_j(S, \boldsymbol{P}),\ &\text{for some } j \in \{i+1, \ldots, n\}, \\ &\text{such that } r, r' \text{ are conflicting and } S \models B(r) \cup B(r') \,\big\}. \end{aligned}$$

That is, $\mathit{Rej}(S, \boldsymbol{P})$ contains those rules from $\boldsymbol{P}$ which are rejected on the basis of rules which are not rejected themselves.

The next lemma ensures that the rejection set $\mathit{Rej}(S, \boldsymbol{P})$ precisely matches the intended meaning of the control atoms $\mathit{rej}(\cdot)$.

Lemma 1

Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$, let S be an answer set of $\boldsymbol{P}$, and let $\check{S}$ be the corresponding answer set of $\boldsymbol{P}_\triangleleft$. Then, $r \in \mathit{Rej}(S, \boldsymbol{P})$ iff $\mathit{rej}(r) \in \check{S}$.

Proof

We show by induction on j $(0 \leq j < n)$ that $r \in Rej_{n-j}(S, \boldsymbol{P})$ iff $rej(r) \in \check{S}$, whenever $r \in P_{n-j}$.

INDUCTION BASE. Assume $j = 0$. Then the statement holds trivially because $Rej_n(S, \boldsymbol{P}) = \emptyset$ and $rej(r) \notin \check{S}$ for all $r \in P_n$.

INDUCTION STEP. Assume $n > j > 0$, and let the statement hold for all $k < j$. We show the assertion for $k = j$. Consider some $r \in P_{n-j}$ and suppose $r \in Rej_{n-j}(S, \boldsymbol{P})$. We show $rej(r) \in \check{S}$. According to the definition of $Rej_{n-j}(S, \boldsymbol{P})$, there is some $r' \in P_{n-k} \setminus Rej_{n-k}(S, \boldsymbol{P})$, $0 \leq k < j$, such that $H(r') = \neg H(r)$, $B^+(r) \cup B^+(r') \subseteq S$, and both r and r' are not defeated by S. The rule $r \in P_{n-j}$ induces the rule $rej(r) \leftarrow B(r), \neg L_{n-j+1} \in \boldsymbol{P}_{\lhd}$, where $L = H(r)$. From the properties above, we have $rej(r) \leftarrow B^+(r), \neg L_{n-j+1} \in (\boldsymbol{P}_{\lhd})^{\check{S}}$. Now, since $B^+(r) \subseteq S \subseteq \check{S}$, in order to show $rej(r) \in \check{S}$ it suffices to show that $\neg L_{n-j+1} \in \check{S}$. This can be seen as follows. First of all, the rule $r' \in P_{n-k}$ induces the rule $L'_{n-k} \leftarrow B(r), not\, rej(r') \in \boldsymbol{P}_{\lhd}$, where $L' = H(r')$. Since $H(r') = \neg H(r)$, we actually have $\neg L_{n-k} \leftarrow B(r), not\, rej(r') \in \boldsymbol{P}_{\lhd}$. Now, given that $r' \notin Rej_{n-k}(S, \boldsymbol{P})$, and since $k < j$, by induction hypothesis we have $rej(r) \notin \check{S}$. Furthermore, $B^-(r') \cap S = \emptyset$ implies $\neg L_{n-k} \leftarrow B^+(r') \in (\boldsymbol{P}_{\lhd})^{\check{S}}$. Given that $B^+(r') \subseteq S \subseteq \check{S}$, we obtain $\neg L_{n-k} \in \check{S}$. By observing that $n - j + 1 \leq n - k$ (since $k < j$), and given the inertia rules $\neg L_m \leftarrow \neg L_{m+1} \in (\boldsymbol{P}_{\lhd})^{\check{S}}$ $(1 \leq m < n)$, we eventually obtain $L_{n-j+1} \in \check{S}$. This proves $rej(r) \in \check{S}$.

Conversely, assume $rej(r) \in \check{S}$. We show $r \in Rej_{n-j}(S, \boldsymbol{P})$. By construction of the update program $\boldsymbol{P}_{\lhd}$, the atom $rej(r)$ can only be derived by means of the rule $rej(r) \leftarrow B(r), \neg L_{n-j+1} \in \boldsymbol{P}_{\lhd}$. So, it must hold that $B^+(r) \subseteq S$, $\neg L_{n-j+1} \in \check{S}$, and $B^-(r) \cap S = \emptyset$. Moreover, since $\neg L_{n-j+1} \in \check{S}$, there must be some $r' \in P_{n-k}$, $k < j$, such that $\neg L_{n-k} \leftarrow B(r'), not\, rej(r') \in \boldsymbol{P}_{\lhd}$, $B^+(r') \subseteq S$, $B^-(r') \cap S = \emptyset$, and $rej(r') \notin \check{S}$. By induction hypothesis, the latter fact implies $r' \notin Rej_{n-k}(S, \boldsymbol{P})$. So, we have that there is some $r' \in P_{n-k} \setminus Rej_{n-k}(S, \boldsymbol{P})$, $k < j$, such that $H(r') = \neg H(r)$, $B^+(r) \cup B^+(r') \subseteq S$, and both r and r' are not defeated by S. This means that $r \in Rej_{n-j}(S, \boldsymbol{P})$. □

It turns out that update answer sets can be characterized in terms of a modified Gelfond-Lifschitz reduction, by taking the elements of the respective rejection sets into account. In what follows, for a given update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$, we write $\cup \boldsymbol{P}$ to denote the set of all rules occurring in $\boldsymbol{P}$, i.e., $\cup \boldsymbol{P} = \bigcup_{i=1}^{n} P_i$.

Theorem 4

Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$ a set of literals. Then, S is an answer set of $\boldsymbol{P}$ iff S is the minimal model of $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$.

Proof

See Appendix A.2. □

Update answer sets can also be described using a weaker notion of rejection sets. For $\boldsymbol{P} = (P_1, \ldots, P_n)$ over $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$, let us define

$$Rej'(S, \boldsymbol{P}) \;=\; \bigcup_{i=1}^{n} \{\, r \in P_i \mid \exists r' \in P_j, \text{ for some } j \in \{i+1, \ldots, n\}, \text{ such that } r \text{ and } r' \text{ are conflicting and } S \models B(r) \cup B(r') \,\}.$$

Obviously, $Rej(S, \boldsymbol{P}) \subseteq Rej'(S, \boldsymbol{P})$ always holds. We get the following partial characterization of update answer sets:

Theorem 5
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$ a set of literals. Then, S is an answer set of $\boldsymbol{P}$ if S is the minimal model of $(\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))^S$.

Proof
Suppose that S is the minimal model of $(\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))^S$, but there is some $r \in (\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P})) \setminus (\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))$ such that $S \models B(r)$ and $H(r) \notin S$. It follows that $r \in Rej'(S, \boldsymbol{P}) \setminus Rej(S, \boldsymbol{P})$.

Define $Q_i = \{r' \in P_i \mid H(r') \in \{H(r), \neg H(r)\} \text{ and } S \models B(r')\}$ and let $r' \in Q_k$, where $k = \max\{i \mid r' \in Q_i\} \neq 0$. Then, $r' \notin Rej'(S, \boldsymbol{P})$. Since $H(r) \notin S$, it follows that $H(r') = \neg H(r)$. Furthermore, $r' \notin Rej'(S, \boldsymbol{P})$ implies $r' \notin Rej(S, \boldsymbol{P})$. Therefore, it follows that $r \in Rej(S, \boldsymbol{P})$, a contradiction. We obtain that S is a model of $(\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. Moreover, since $(\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))^S \subseteq (\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$, S must be a minimal model of $(\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. □

The converse of Theorem 5 is not true in general, but holds for restricted classes of programs. For example, consider the following property of an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ and an answer set S of $\boldsymbol{P}$ ("chain condition"):

(CH) For each pair r, r' of rules $r \in P_i$ and $r' \in P_j$ such that $1 \leq i < j < n$, $H(r) = \neg H(r')$, and $S \models B(r) \cup B(r')$, it holds that either $r' \notin Rej(S, \boldsymbol{P})$ or some $r'' \in P_{j+1} \cup \cdots \cup P_n$ exists such that $H(r'') = \neg H(r')$ and $B(r'') \subseteq B(r)$.

Then, we obtain the following result.

Theorem 6
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence over a set of atoms $\mathcal{A}$ and let S be an answer set of $\boldsymbol{P}$ such that Property (CH) holds. Then, S is the minimal model of $(\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))^S$.

Proof
Suppose S is an answer set of $\boldsymbol{P}$. Then, by Theorem 4, S is a minimal model of $(\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$, i.e., $S = Cn((\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S)$, where $Cn(\cdot)$ is the classical-literal consequence operator. We show that for each rule $r \in Rej'(S, \boldsymbol{P}) \setminus Rej(S, \boldsymbol{P})$, there exists some rule $r'' \in \cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P})$ such that $H(r) = H(r'')$ and $B(r'') \subseteq B(r)$; this implies that $Cn((\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S) = Cn((\cup\boldsymbol{P} \setminus Rej'(S, \boldsymbol{P}))^S)$, from which the result is easily obtained.

Let $r \in P_i$ for some $i \in \{1, \ldots, n\}$ such that $r \in Rej'(S, \boldsymbol{P}) \setminus Rej(S, \boldsymbol{P})$. By definition of $Rej'(S, \boldsymbol{P})$, there exists some $r' \in P_j$ for some $j \in \{i+1, \ldots, n\}$ such that $H(r) = \neg H(r')$ and $S \models B(r')$. Without loss of generality, let r' be such that

j is maximal. We consider two cases: Assume first $r' \notin Rej(S, \boldsymbol{P})$. The definition of $Rej(S, \boldsymbol{P})$ implies that $r \in Rej(S, \boldsymbol{P})$, which is a contradiction. Assume now that $r' \in Rej(S, \boldsymbol{P})$ holds. By Property (CH), there exists a rule r'' in some $P_{j'}$, $j' \in \{j+1, \ldots, n\}$ such that $H(r'') = \neg H(r')$ $(= H(r))$ and $S \models B(r'')$. Now if $r'' \notin Rej'(S, \boldsymbol{P})$, then r'' is as to prove. If, on the other hand, $r'' \in Rej'(S, \boldsymbol{P})$, then some rule $r''' \in P_{j''}$ with $j'' \in \{j'+1, \ldots, n\}$, $H(r''') = H(r')$, and $S \models B(r''')$ exists. However, r''' contradicts the maximality of r', and thus $r'' \in Rej'(S, \boldsymbol{P})$ is not possible. This proves the claim and the result. □

We can easily derive from this result syntactic classes of update sequences for which the converse of Theorem 5 holds; a natural class is given by the following condition on an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$:

(CH′) For each pair r, r' of rules $r \in P_i$ and $r' \in P_j$ such that $1 \leq i < j < n$, $H(r) = \neg H(r')$, and $B(r) \cup B(r')$ is satisfiable, there exists some $r'' \in P_{j+1} \cup \cdots \cup P_n$ such that $H(r'') = \neg H(r')$ and $B(r'') \subseteq B(r)$.

It is important to emphasize that in our approach, the update program $P_{\triangleleft}$ is not the *result* of the update intended to be the new knowledge state of the agent, but it *represents the semantic result* of the information that a sequence of updates $P_2, \ldots, P_n$ has occurred to a knowledge base P_1. Compiling the result of updates into a single logic program in the original language (having the desired answer sets) would mean losing history information about the update sequence. Instead, the formalism results in a program over an extended set of atoms, which expresses at the object level meta-concepts determining applicability of rules and computation of those intended answer sets. In some sense, the result is therefore a declarative specification of how rules of the original logic program and of subsequent updates should be applied, expressed in the language of logic programs themselves.

4 Principles of Program Updates

In this section, we discuss several kinds of postulates which have been advocated in the literature on belief change and examine to what extent update sequences satisfy these principles. This issue has not been addressed extensively in previous work. We first consider update programs from the perspective of *belief revision* and assess the relevant postulates from this area. Afterwards, we briefly analyze further properties, like viewing update programs as *nonmonotonic consequence operators* and other general principles. We remark that our analysis applies, in slightly adapted form, to dynamic logic programming as well (cf. Section 7.3).

4.1 Belief Revision

Following Gärdenfors and Rott (1995), two different approaches to belief revision can be distinguished: (i) *immediate revision*, where the new information is simply added to the current stock of beliefs and the belief change is accomplished by the semantics of the underlying (often, nonmonotonic) logic; and (ii) *logic-constrained* *revision*, where the new stock of beliefs is determined by a nontrivial operation which adds and retracts beliefs, respecting logical inference and some constraints.

In the latter approach, it is assumed that beliefs are sentences from a given logical language $\mathcal{L}_\mathcal{B}$, closed under the standard boolean connectives. A *belief set*, K, is a subset of $\mathcal{L}_\mathcal{B}$ which is closed under a consequence operator $Cn(\cdot)$ of the underlying logic. A *belief base* for K is a subset $B \subseteq K$ such that $K = Cn(B)$. A belief base is a special case of *epistemic state* (Darwiche & Pearl, 1997), which is a set of sentences E representing an associated belief set K in terms of a mapping $Bel(\cdot)$ such that $K = Bel(E)$, where E need not necessarily have the same language as K.

In what follows, we first introduce different classes of postulates, and then we examine them with respect to update sequences.

4.1.1 AGM Postulates

One of the main aims of logic-constrained revision is to characterize suitable revision operators through postulates. In the AGM approach (after Alchourrón, Gärdenfors, and Makinson (1985)), three basic operations on a belief set K are considered:

- *expansion* $K + \phi$, which is simply adding the new information $\phi \in \mathcal{L}_B$ to K;
- *revision* $K \star \phi$, which is sensibly revising K in the light of ϕ (in particular, when K contradicts ϕ); and
- *contraction* $K - \phi$, which is removing ϕ from K.

AGM proposes a set of postulates, K$\star$1–K$\star$8, that any revision operator $\star$ mapping a belief set $K \subseteq \mathcal{L}_B$ and a sentence $\phi \in \mathcal{L}_B$ into the revised belief set $K \star \phi$ should satisfy. If, following both Darwiche and Pearl (1997) and Brewka (2000), we assume that K is represented by an epistemic state E, then the postulates K$\star$1–K$\star$8 can be reformulated as follows:

(K1) $E \star \phi$ represents a belief set.
(K2) $\phi \in Bel(E \star \phi)$.
(K3) $Bel(E \star \phi) \subseteq Bel(E + \phi)$.
(K4) $\neg\phi \notin Bel(E)$ implies $Bel(E + \phi) \subseteq Bel(E \star \phi)$.
(K5) $\bot \in Bel(E \star \phi)$ only if ϕ is unsatisfiable.
(K6) $\phi_1 \equiv \phi_2$ implies $Bel(E \star \phi_1) = Bel(E \star \phi_2)$.
(K7) $Bel(E \star (\phi \wedge \psi)) \subseteq Bel((E \star \phi) + \psi)$.
(K8) $\neg\psi \notin Bel(E \star \phi)$ implies $Bel((E \star \phi) + \psi) \subseteq Bel(E \star (\phi \wedge \psi))$.

Here, $E \star \phi$ and $E + \phi$ is the revision and expansion operation, respectively, applied to E. Informally, these postulates express that the new information should be reflected after the revision, and that the belief set should change as little as possible. As has been pointed out, this set of postulates is appropriate for new information about an *unchanged world*, but not for incorporation of a change to the actual world. Such a mechanism is addressed by the next set of postulates, expressing *update* operations.

4.1.2 Update Postulates

For update operators $B \diamond \phi$ realizing a change ϕ to a belief base B, Katsuno and Mendelzon (1991) proposed a set of postulates, U◇1–U◇8, where both ϕ and B are propositional sentences over a finitary language. For epistemic states E, these postulates can be reformulated as follows.

(U1) $\phi \in Bel(E \diamond \phi)$.
(U2) $\phi \in Bel(E)$ implies $Bel(E \diamond \phi) = Bel(E)$.
(U3) If $Bel(E)$ is consistent and ϕ is satisfiable, then $Bel(E \diamond \phi)$ is consistent.
(U4) If $Bel(E) = Bel(E')$ and $\phi \equiv \psi$, then $Bel(E \diamond \phi) = Bel(E \diamond \psi)$.
(U5) $Bel(E \diamond (\phi \wedge \psi)) \subseteq Bel((E \diamond \phi) + \psi)$.
(U6) If $\phi \in Bel(E \diamond \psi)$ and $\psi \in Bel(E \diamond \phi)$, then $Bel(E \diamond \phi) = Bel(E \diamond \psi)$.
(U7) If $Bel(E)$ is complete, then $Bel(E \diamond (\psi \vee \psi')) \subseteq Bel(E \diamond \psi) \wedge Bel(E \diamond \psi'))$.[3]
(U8) $Bel((E \vee E') \diamond \psi) = Bel((E \diamond \psi) \vee (E' \diamond \psi)$.

Here, conjunction and disjunction of epistemic states are presumed to be definable in the given language (e.g., in terms of intersection and union of associated sets of models, respectively).

The most important differences between (K1)–(K8) and (U1)–(U8) are that revision should yield the same result as expansion $E + \phi$, providing ϕ is compatible with E, which is not desirable for update in general, cf. (Winslett, 1988). On the other hand, (U8) says that if E can be decomposed into a disjunction of states (e.g., models), then each case can be updated separately and the overall result is formed by taking the disjunction of the emerging states.

4.1.3 Iterated Revision

Darwiche and Pearl (1997) have proposed postulates for iterated revision, which can be rephrased in our setting as follows (we omit parentheses in sequences $(E \star \phi_1) \star \phi_2$ of revisions):

(C1) If $\psi_2 \in Bel(\psi_1)$, then $Bel(E \star \psi_2 \star \psi_1) = Bel(E \star \psi_1)$.
(C2) If $\neg\psi_2 \in Bel(\psi_1)$, then $Bel(E \star \psi_1 \star \psi_2) = Bel(E \star \psi_2)$.
(C3) If $\psi_2 \in Bel(E \star \psi_1)$, then $\psi_2 \in Bel(E \star \psi_2 \star \psi_1)$.
(C4) If $\neg\psi_2 \notin Bel(E \star \psi_1)$, then $\neg\psi_2 \notin Bel(E \star \psi_2 \star \psi_1)$.
(C5) If $\neg\psi_2 \in Bel(E \star \psi_1)$ and $\psi_1 \notin Bel(E \star \psi_2)$, then $\psi_1 \notin Bel(E \star \psi_1 \star \psi_2)$.
(C6) If $\neg\psi_2 \in Bel(E \star \psi_1)$ and $\neg\psi_1 \in Bel(E \star \psi_2)$, then $\neg\psi_1 \in Bel(E \star \psi_1 \star \psi_2)$.

Another set of postulates for iterated revision, corresponding to a sequence E of observations, has been formulated by Lehmann (1995). Here, each observation is a sentence which is assumed to be consistent (i.e., falsity is not observed), and the epistemic state E has an associated belief set $Bel(E)$. Lehmann's postulates read as follows, where E, E' denote sequences of observations and "," stands for concatenation:

[3] A belief set K is *complete* iff, for each atom A, either $A \in K$ or $\neg A \in K$.

(I1) $Bel(E)$ is a consistent belief set.
(I2) $\phi \in Bel(E, \phi)$.
(I3) If $\psi \in Bel(E, \phi)$, then $\phi \Rightarrow \psi \in Bel(E)$.
(I4) If $\phi \in Bel(E)$, then $Bel(E, \phi, E') = Bel(E, E)$.
(I5) If $\psi \vdash \phi$ then $Bel(E, \phi, \psi, E') = Bel(E, \psi, E')$.
(I6) If $\neg\psi \notin Bel(E, \phi)$, then $Bel(E, \phi, \psi, E') = Bel(E, \phi, \psi, E')$.
(I7) $Bel(E, \neg\phi, \phi) \subseteq Cn(E + \phi)$.

4.1.4 Analysis of the Postulates

In order to evaluate the different postulates, we need to adapt them for the setting of update programs. Naturally, the epistemic state $\boldsymbol{P} = (P_1, \ldots, P_n)$ of an agent is subject to revision. However, the associated belief set $Bel(\boldsymbol{P})$ ($\subseteq \mathcal{L}_{\mathcal{A}}$) does not belong to a logical language closed under boolean connectives. Closing $\mathcal{L}_{\mathcal{A}}$ under conjunction does not cause much troubles, as the identification of finite logic programs with finite conjunctions of clauses permits that updates of a logic program P by a program P' can be viewed as the update of P with a single sentence from the underlying belief language. Ambiguities arise, however, with the interpretation of expansion, as well as with the meaning of negation and disjunction of rules and programs, respectively.

Depending on whether the particular structure of the epistemic state E should be respected, different definitions of expansion are imaginable in our framework. At the "extensional" level of sentences, represented by a program or sequence of programs $\boldsymbol{P}$, $Bel(\boldsymbol{P} + P')$ is defined as $Bel(Bel(\boldsymbol{P}) \cup P')$. At the "intensional" level of sequences $\boldsymbol{P} = (P_1, \ldots, P_n)$, $Bel(\boldsymbol{P}+P')$ could be defined as $Bel(P_1, \ldots, P_n \cup P')$. An intermediate approach would be defining $Bel(\boldsymbol{P} + P') = Bel_{\mathcal{A}}(\boldsymbol{P}_{\triangleleft} \cup P')$. We adopt the extensional view here. Note that, in general, adding P' to $Bel(\boldsymbol{P})$ does not amount to the semantical intersection of P' and $Bel(\boldsymbol{P})$ (nor of $\cup\boldsymbol{P}$ and P', respectively).

As for negation, we might interpret the condition $\neg\phi \notin Bel(E)$ (or $\neg\psi \notin Bel(E \star \phi)$ in (K4) and (K8)) as satisfiability requirement for $E + \phi$ (or $(E \star \phi) + \psi$, respectively).

Disjunction $\vee$ of rules or programs (as epistemic states) appears to be meaningful only at the semantical level. The union $\mathcal{S}(P_1) \cup \mathcal{S}(P_2)$ of the answer sets of programs P_1 and P_2 may be represented syntactically through a program P_3, which in general requests an extended set of atoms. We thus do not consider the postulates involving the operator $\vee$.

Given these considerations, Table 1 summarizes our interpretation of postulates (K1)–(K8) and (U1)–(U6), and includes references whether the respective property holds or fails. We assume that $\boldsymbol{P}, \boldsymbol{P'}$ are sequences of ELPs, and P, P' denote single ELPs. Moreover, the notation $(\boldsymbol{P}, P)$ is an abbreviation for the sequence $(P_1, \ldots, P_n, P)$ if $\boldsymbol{P} = (P_1, \ldots, P_n)$. Demonstrations and counterexamples concerning these properties are given in Appendix A.3, and can be easily adapted for dynamic logic programming too.

As can be seen from Table 1, apart from very simple postulates, the majority of

Table 1. *Interpretation of Postulates (K1)–(K8) and (U1)–(U6).*

Postulate	Interpretation	Postulate holds
(K1)	$(\boldsymbol{P}, P)$ represents a belief set	yes
(K2), (U1)	$P \subseteq Bel((\boldsymbol{P}, P))$	yes
(U2)	$Bel(P) \subseteq Bel(\boldsymbol{P})$ implies $Bel((\boldsymbol{P}, P)) = Bel(\boldsymbol{P})$	no
(K3)	$Bel((\boldsymbol{P}, P)) \subseteq Bel(Bel(\boldsymbol{P}) \cup P)$	yes[a]
(U3)	If $\boldsymbol{P}$ and P are satisfiable, then $(\boldsymbol{P}, P)$ is satisfiable	no
(K4)	If $Bel(\boldsymbol{P}) \cup P$ has an answer set, then $Bel(Bel(\boldsymbol{P}) \cup P) \subseteq Bel((\boldsymbol{P}, P))$	no
(K5)	$(\boldsymbol{P}, P)$ is unsatisfiable only if P is unsatisfiable	no
(K6), (U4)	$\boldsymbol{P} \equiv \boldsymbol{P}'$ and $P \equiv P'$ implies $(\boldsymbol{P}, P) \equiv (\boldsymbol{P}', P')$	no
(K7), (U5)	$Bel((\boldsymbol{P}, P \cup P')) \subseteq Bel(Bel((\boldsymbol{P}, P)) \cup P')$	yes[b]
(U6)	$Bel(P') \subseteq Bel((\boldsymbol{P}, P))$ and $Bel(P) \subseteq Bel((\boldsymbol{P}, P'))$ implies $Bel((\boldsymbol{P}, P)) = Bel((\boldsymbol{P}, P'))$	no
(K8)	If $Bel((\boldsymbol{P}, P)) \cup P'$ is satisfiable, then $Bel(Bel((\boldsymbol{P}, P)) \cup P') \subseteq Bel((\boldsymbol{P}, P \cup P'))$	no

[a] If either $\boldsymbol{P}$ or P has a finite alphabet.
[b] If either $(\cup\boldsymbol{P}) \cup P$ or P' has a finite alphabet.

the adapted AGM and update postulates are violated by update programs. This holds even for the case where $\boldsymbol{P}$ is a single program. In particular, $Bel((\boldsymbol{P}, P))$ violates discriminating postulates such as (U2) for update and (K4) for revision. In the light of this, update programs neither have update nor revision flavor.

We remark that the picture does not change if we abandon extensional expansion and consider the postulates under intensional expansion. Thus, also under this view, update programs do not satisfy minimality of change.

The postulates (C1)–(C6) and (I1)–(I7) for iterated revision are treated in Table 2; proofs of these properties can be found in Appendix A.4. Observe that Lehmann's postulate (I3) is considered as the pendant to AGM postulate K$\star$3. In a literal interpretation of (I3), since the belief language associated with logic programs does not have implication, we may consider the case where ψ is a default literal L_0 and $\phi = L_1 \wedge \cdots \wedge L_k$ is a conjunction of literals L_i, such that $\phi \Rightarrow \psi$ corresponds to the rule $L_0 \leftarrow L_1, \ldots, L_k$. Moreover, since the negation of logic programs is not defined, we do not interpret (I7).

Note that, although postulate (C3) fails in general, it holds if P' contains a single rule. Thus, all of the above postulates except (C4) fail, and, with the exception of (C3), each change is given by a single rule.

We can view the epistemic state $\boldsymbol{P} = (P_1, \ldots, P_n)$ of an agent as a prioritized belief base in the spirit of (Brewka, 1991b; Nebel, 1991; Benferhat *et al.*, 1993).

Table 2. *Interpretation of Postulates (C1)–(C6) and (I1)–(I6).*

Postulate	Interpretation	Postulate holds
(C1)	If $P' \subseteq Bel(P)$, then $Bel((\boldsymbol{P}, P', P)) = Bel((\boldsymbol{P}, P))$	no
(C2)	If $S \not\models P'$, for all $S \in \mathcal{S}(P)$, then $Bel((\boldsymbol{P}, P, P')) = Bel((\boldsymbol{P}, P'))$	no
(C3)	If $P' \subseteq Bel((\boldsymbol{P}, P))$, then $P' \subseteq Bel((\boldsymbol{P}, P', P))$	no
(C4)	If $S \models P'$ for some $S \in \mathcal{U}((\boldsymbol{P}, P))$, then $S \models P'$ for some $S \in \mathcal{U}((\boldsymbol{P}, P', P))$	yes
(C5)	If $S \not\models P'$ for all $S \in \mathcal{U}((\boldsymbol{P}, P))$ and $P \not\subseteq Bel((\boldsymbol{P}, P'))$, then $P \not\subseteq Bel((\boldsymbol{P}, P, P'))$	no
(C6)	If $S \not\models P'$ for all $S \in \mathcal{U}((\boldsymbol{P}, P))$ and $S \not\models P$ for all $S \in \mathcal{U}((\boldsymbol{P}, P'))$, then $S \not\models P$ for all $S \in \mathcal{U}((\boldsymbol{P}, P, P'))$	no
(I1)	$Bel(\boldsymbol{P})$ is a consistent belief set	no
(I2)	$P \subseteq Bel((\boldsymbol{P}, P))$	yes
(I3)	If $L_0 \leftarrow\ \in Bel((\boldsymbol{P}, \{L_1 \leftarrow, \ldots, L_k \leftarrow\}))$, then $L_0 \leftarrow L_1, \ldots, L_k \in Bel(\boldsymbol{P})$	yes
(I4)	If $Q_1 \subseteq Bel(\boldsymbol{P})$, then $Bel((\boldsymbol{P}, Q_1, Q_2, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_2, \ldots, Q_n))$	no
(I5)	If $Bel(Q_2) \subseteq Bel(Q_1)$, then $Bel((\boldsymbol{P}, Q_1, Q_2, Q_3, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_2, Q_3, \ldots, Q_n))$	no
(I6)	If $S \models Q_2$ for some $S \in \mathcal{U}((\boldsymbol{P}, Q_1))$, then $Bel((\boldsymbol{P}, Q_1, Q_2, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_1, Q_1 \cup Q_2, Q_3, \ldots, Q_n))$	no

Revision with a new piece of information Q is accomplished by simply changing the epistemic state to $\boldsymbol{P'} = (P_1, \ldots, P_n, Q)$. The change of the belief base is then automatically accomplished by the nonmonotonic semantics of a sequence of logic programs. Under this view, updating logic programs amounts to an instance of the immediate revision approach.

On the other hand, referring to the update program, we may view the belief set of the agent represented through a pair $\langle P, \mathcal{A} \rangle$ of a logic program P and a (fixed) set of atoms $\mathcal{A}$, such that its belief set is given by $Bel_{\mathcal{A}}(P)$. Under this view, a new piece of information Q is incorporated into the belief set by producing a representation, $\langle P', \mathcal{A} \rangle$, of the new belief set, where $P' = P \triangleleft Q$. Here, (a set of) sentences from an extended belief language is used to characterize the new belief set, which is constructed by a non-trivial operation employing the semantics of logic programs. Thus, update programs enjoy to some extent also a logic-constrained revision flavor. Nonetheless, as also the failure of postulates shows, they are more an instance of *immediate* than *logic-constrained* revision. What we naturally expect, though, is that the two views described above amount to the same *at a technical level*. However, as we shall demonstrate below, this is not true in general.

4.2 Update Programs as Nonmonotonic Consequence Relations

Following Gärdenfors and Makinson (1991; 1994), belief revision can be related to nonmonotonic reasoning by interpreting it as an abstract consequence relation on sentences, where the epistemic state is fixed. In the same way, we can interpret update programs as abstract consequence relation on programs as follows. For a fixed epistemic state $\boldsymbol{P}$ and logic programs P_1 and P_2, we define

$$P_1 \mathrel{|\!\sim_{\boldsymbol{P}}} P_2 \text{ if and only if } P_2 \subseteq Bel(\boldsymbol{P}, P_1),$$

i.e., if the rules P_2 are in the belief set of the agent after update of the epistemic state with P_1.

Various postulates for nonmonotonic inference operations have been identified in the literature. In what follows, we consider some sets of postulates and discuss their interpretations in terms of update programs. First of all, we review principles discussed by Makinson (1993), who considered a set of (desirable) properties for nonmonotonic reasoning, and analyzed the behavior of some reasoning formalisms with respect to these properties. Afterwards, we consider postulates proposed by Lehmann and Magidor (1992), which deal with properties of so-called *preferential consequence relations*. It is argued that such properties are necessary but not sufficient for a preferential consequence relation to be meaningful and useful in reasoning. As we will see, updates fail also in satisfying the essential properties.

Although our analysis is based on the specific semantics expressed by the transformation of Definition 2, arguably it holds for other update formalisms as well. In fact, quite the same pattern can be found for dynamic logic programs (Alferes *et al.*, 2000), because, with few exceptions, all proofs and counterexamples hold for this formalism too (dynamic logic programming will be discussed in detail in Section 7.3). Thus, intuitively, the failure of some basic principles of nonmonotonic reasoning in the context of updates stems from the same nature of update semantics based on rule rejection, and not on the particular transformation chosen.

4.2.1 General Patterns of Nonmonotonic Inference Relations

Gabbay (1985) was the first to propose the idea that the output of nonmonotonic systems should be considered as an abstract consequence relation, in order to get a clearer understanding of the diverse nonmonotonic reasoning formalisms. Ensuing research identified several important principles, based on both syntactic and model-theoretic considerations. Among the different properties analyzed by Makinson (1993), the following principles are amenable for logic programs under the standard Gelfond-Lifschitz approach, and can thus be formulated for update programs as well:

(N1) $P_1 \mathrel{|\!\sim_{\boldsymbol{P}}} P_1$.

(N2) If $P_1 \mathrel{|\!\sim}_{\boldsymbol{P}} Q_1 \wedge \ldots \wedge Q_m$ and $P_1 \wedge Q_1 \wedge \ldots \wedge Q_m \mathrel{|\!\sim}_{\boldsymbol{P}} P_2$, then $P_1 \mathrel{|\!\sim}_{\boldsymbol{P}} P_2$.

(N3) If $P_1 \mathrel{|\!\sim}_{\boldsymbol{P}} Q_1 \wedge \ldots \wedge Q_m$ and $P_1 \mathrel{|\!\sim}_{\boldsymbol{P}} P_2$, then $P_1 \wedge Q_1 \wedge \ldots \wedge Q_m \mathrel{|\!\sim}_{\boldsymbol{P}} P_2$.

(N4) If $P_1 \mathrel{|\!\sim}_{\boldsymbol{P}} P_2,\ P_2 \mathrel{|\!\sim}_{\boldsymbol{P}} P_3, \ldots,\ P_n \mathrel{|\!\sim}_{\boldsymbol{P}} P_1$ $(n \geq 2)$, then $\{P' \mid P_i \mathrel{|\!\sim}_{\boldsymbol{P}} P'\} = \{P' \mid P_j \mathrel{|\!\sim}_{\boldsymbol{P}} P'\}$, for all $i, j \leq n$.

Postulate (N1) is called *Inclusion* and coincides with (K2) and (U1). Properties (N2) and (N3) are important nonmonotonic inference principles and are respectively called *Cut* and *Cautious Monotony*. Inference relations which obey both of these properties are said to be *cumulative*. It is well-known that most nonmonotonic formalisms are not cumulative, and several variants of standard nonmonotonic approaches have been defined in order to satisfy cumulativity (e.g., (Brewka, 1991a; Schaub, 1991)). The last principle, (N4), is called *Loop* and was first formulated and studied by Kraus, Lehmann, and Magidor (1990) as a property of inference relations generated by preferential model structures. Roughly speaking, Loop expresses a syntactic counterpart of transitivity on the model structure.

Other properties, additionally studied by Makinson (1993), which cannot be interpreted for logic programs include *Supraclassicality*, *Absorption*, *Distribution*, and *Consistency Preservation*. We refer the reader to (Makinson, 1993) for more information on these principles.

4.2.2 Properties of Updates as a Preferential Relation

Kraus, Lehmann, and Magidor (1990) defined *preferential consequence relations* as binary relations $\mathrel{|\!\sim}$ over propositional formulas satisfying the following properties (here, "$\models$" denotes validity in classical propositional logic):

(P1) If $\models (\phi \Leftrightarrow \psi)$ and $\phi \mathrel{|\!\sim} \gamma$, then $\psi \mathrel{|\!\sim} \gamma$.
(P2) If $\models (\phi \Rightarrow \psi)$ and $\gamma \mathrel{|\!\sim} \phi$, then $\gamma \mathrel{|\!\sim} \psi$.
(P3) $\phi \mathrel{|\!\sim} \phi$.
(P4) If $\phi \mathrel{|\!\sim} \psi$ and $\phi \mathrel{|\!\sim} \gamma$, then $\phi \mathrel{|\!\sim} \psi \wedge \gamma$.
(P5) If $\phi \mathrel{|\!\sim} \gamma$ and $\psi \mathrel{|\!\sim} \gamma$, then $(\phi \vee \psi) \mathrel{|\!\sim} \gamma$.
(P6) If $\phi \mathrel{|\!\sim} \psi$ and $\phi \mathrel{|\!\sim} \gamma$, then $(\phi \wedge \psi) \mathrel{|\!\sim} \gamma$.

Rule (P1) is called *Left Logical Equivalence* and (P2) is the principle of *Right Weakening*. Property (P3) coincides with (N1) and is referred to by Kraus, Lehmann, and Magidor as *Reflexivity*. (P4) and (P5) are respectively called *And* and *Or*. The last rule, (P6), is identical with (N3), the principle of Cautious Monotony.

As noted by the above authors, any assertional relation satisfying (P1)–(P6) also satisfies Cut, expressed here as the following rule:

$$\text{If } \phi \wedge \psi \mathrel{|\!\sim} \gamma \text{ and } \phi \mathrel{|\!\sim} \psi, \text{ then } \phi \mathrel{|\!\sim} \gamma.$$

Since not all preferential relations can be considered as reasonable inference procedures, Lehmann and Magidor (1992) subsequently defined a more restricted class of preferential relations, called *rational consequence relations*. They show that such rational consequence relations give rise to logical closure operations which satisfy the principle of cumulativity. Since none of the postulates for rational relations can be formulated for logic programs, they are not discussed here.

Table 3. *Interpretation of Postulates (N1)–(N4), (P1), (P2), and (P4).*

Postulate	Interpretation	Postulate holds
(N1)	$P_1 \in Bel((\boldsymbol{P}, P_1))$	yes
(N2)	If $\bigcup_{i=1}^m Q_i \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup \bigcup_{i=1}^m Q_1))$, then $P_2 \subseteq Bel((\boldsymbol{P}, P_1))$	yes
(N3)	If $\bigcup_{i=1}^m Q_i \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_2 \subseteq Bel((\boldsymbol{P}, P_1))$, then $P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup \bigcup_{i=1}^m Q_1))$	no
(N4)	If $P_{i+1} \subseteq Bel((\boldsymbol{P}, P_i))$ $(1 \leq i < n)$ and $P_1 \subseteq Bel((\boldsymbol{P}, P_n))$ $(n \geq 2)$, then $\{P' \mid P' \subseteq Bel((\boldsymbol{P}, P_i))\} = \{P' \mid P' \subseteq Bel((\boldsymbol{P}, P_j))\}$, for all $i, j \leq n$	no
(P1)	If $P_1 \equiv P_2$ and $P_3 \subseteq Bel((\boldsymbol{P}, P_1))$, then $P_3 \subseteq Bel((\boldsymbol{P}, P_2))$	no
(P2)	If $P_1 \models P_2$ and $P_1 \subseteq Bel((\boldsymbol{P}, P_3))$, then $P_2 \subseteq Bel((\boldsymbol{P}, P_3))$	no
(P4)	If $P_2 \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_3 \subseteq Bel((\boldsymbol{P}, P_1))$, then $P_2 \cup P_3 \subseteq Bel((\boldsymbol{P}, P_1))$	yes

4.2.3 Analysis

The interpretation of postulates (N1)–(N4) in terms of update sequences is given in Table 3. The results show that (N1) and (N2) hold, whereas (N3) and (N4) fail. This corresponds to the situation of standard logic programs under the answer set semantics. Hence, in some sense, updates do not represent a loss in properties with respect to standard answer set semantics.

Table 3 contains also the interpretation of postulates (P1)–(P6). As a matter of fact, since (P3) and (P6) coincide with (N1) and (N3), respectively, and (P5) admits no interpretation in terms of logic programs, only postulates (P1), (P2), and (P4) are included in Table 3. Like the failure of (K6) and (U4), the failure of postulate (P1) showcases the syntax-dependency of update programs, as equivalent programs do not behave the same way under identical update information. Proofs and counterexamples for properties (N1)–(N4), (P1), (P2), and (P4) are given in Appendix A.5.

4.3 Further Properties

Rounding off our discussion on principles of update sequences, we describe some additional general properties which, as we believe, updates and sequences of updates should satisfy. The given properties are not developed in a systematic manner, though, and they are not meant to represent an exhaustive list. Unless stated otherwise, update programs enjoy these properties.

Addition of Tautologies: If the program Q contains only tautological rules (i.e., if Q contains only rules of the form $L \leftarrow L$), then $(\boldsymbol{P}, Q) \equiv \boldsymbol{P}$.

This property is violated, which is also the case e.g. for dynamic logic programs. Consider the programs $P_1 = \{a \leftarrow\ \}$, $P_2 = \{\neg a \leftarrow\ \}$, and $P_3 = \{a \leftarrow a\}$. Then (P_1, P_2) has the single answer set $\{\neg a\}$. By updating with P_3, the interaction between P_1 and P_3 generates another answer set for (P_1, P_2, P_3), namely $\{a\}$. Note, however, that tautological rules in updates are, as we believe, rare in practical applications and can be eliminated easily.

Initialization: $(\emptyset, P) \equiv P$.

This property states that the update of an initial empty knowledge base yields just the update itself.

Idempotence: $(P, P) \equiv P$.

Updating program P by itself has no effect. This property is in fact a special case of the following principle:

Absorption: $(\boldsymbol{P}, Q, Q) \equiv (\boldsymbol{P}, Q)$.

The next three properties express conditions involving programs over disjoint alphabets.

Update of Disjoint Programs: If $P = P_1 \cup P_2$ is the union of programs P_1, P_2 on disjoint alphabets $\mathcal{A}_1$ and $\mathcal{A}_2$, then $P \lhd Q \equiv_{\mathcal{A}_1 \cup \mathcal{A}_2} (P_1 \lhd Q) \cup (P_2 \lhd Q)$.

Parallel Updates: If $\boldsymbol{P} = (P_1, \ldots, P_n)$ is an update sequence over $\mathcal{A}$, and Q_1 and Q_2 are programs defined over disjoint alphabets $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, then $P_1 \lhd \cdots \lhd P_n \lhd (Q_1 \cup Q_2) \equiv_{\mathcal{A} \cup \mathcal{A}_1 \cup \mathcal{A}_2} (P_1 \lhd \cdots \lhd P_n \lhd Q_1) \cup (P_1 \lhd \cdots \lhd P_n \lhd Q_2)$.

In other words, the update by non-interfering programs can be done in parallel, by merging the respective results. This property is not satisfied: Consider the case $n = 1$, with $P_1 = P$ and $Q_2 = \emptyset$. Assuming that the property holds, we would have $Bel(P \lhd Q_1) = Bel((P \lhd Q_1) \cup P)$, i.e., P holds in (P, Q_1) no matter what. This is quite obviously not the case.

Noninterference: If P_1 and P_2 are programs defined over disjoint alphabets, then $(\boldsymbol{P}, P_1, P_2) \equiv (\boldsymbol{P}, P_2, P_1)$.

That is, the order of updates which do not interfere with each other is immaterial.

This property is an immediate consequence of the following stronger property: Suppose $Q \subseteq P_2$ is a program such that there are no rules $r \in Q$ and $r' \in (P_2 \backslash Q) \cup P_1$ with $H(r) = \neg H(r')$. Then, $(\boldsymbol{P}, P_1, P_2) \equiv (\boldsymbol{P}, P_1 \cup Q, P_2 \setminus Q)$.

Augmented Update: If $P_1 \subseteq P_2$, then $(\boldsymbol{P}, P_1, P_2) \equiv (\boldsymbol{P}, P_2)$.

Updating with additional rules makes the previous update obsolete. This property is a somewhat stronger, syntactic variant of the postulate (C1) from above, which fails. On the other hand, it includes Absorption as a special case.

Note that $(\boldsymbol{P}, P_2, P_1) \equiv (\boldsymbol{P}, P_2)$ does in general *not* hold, which may be desired in some cases: Omission of a rule r in P_2 with respect to P_1 leaves the possibility to violate r.

As mentioned before, a sequence of updates $\boldsymbol{P} = (P_1, \ldots, P_n)$ can be viewed either from the point of view of "immediate" revision, or as "logic-constrained" revision. The following property, which deserves particular attention, expresses equivalence of these views (the property is formulated for the case $n = 3$):

Iterativity: For any epistemic state P_1 and ELPs P_2 and P_3 over $\mathcal{A}$, it holds that $P_1 \lhd P_2 \lhd P_3 \equiv_{\mathcal{A}} (P_1 \lhd P_2) \lhd P_3$.

However, this property fails. Informally, soundness of this property would mean that a sequence of three updates is a shorthand for iterated update of a single program, i.e., the result of $P_1 \lhd P_2$ is viewed as a singleton sequence. Stated another way, this property would mean that the definition for $P_1 \lhd P_2 \lhd P_3$ can be viewed as a shorthand for the nested case. Vice versa, this property reads as possibility to forget an update once and for all, by incorporating it immediately into the current belief set.

For a concrete counterexample, consider $P_1 = \emptyset$, $P_2 = \{a \leftarrow, \ \neg a \leftarrow \}$, and $P_3 = \{a \leftarrow \}$. The program $\boldsymbol{P}_{\lhd} = P_1 \lhd P_2 \lhd P_3$ has a unique answer set, in which a is true. On the other hand, $(P_1 \lhd P_2) \lhd P_3$ has no consistent answer set. Informally, while the "local" inconsistency of P_2 is removed in $P_1 \lhd P_2 \lhd P_3$ by rejection of the rule $\neg a \leftarrow$ via P_3, a similar rejection in $(P_1 \lhd P_2) \lhd P_3$ is blocked because of a renaming of the predicates in $P_1 \lhd P_2$. The local inconsistency of P_2 is thus not eliminated.

However, under certain conditions, which exclude such possibilities for local inconsistencies, the iterativity property holds, given by the following result:

Theorem 7
Let $\boldsymbol{P} = (P_1, \ldots, P_m, P_{m+1}, \ldots, P_n)$, $n > m \geq 2$ be a sequence of programs over a set of atoms $\mathcal{A}$. Suppose that for any conflicting rules $r_1, r_2 \in P_i$, $i \leq m$, one of the following conditions holds:

(i) There is some rule $r \in P_j$, $i < j \leq m$, such that either $H(r) = H(r_1)$ and $B(r) \subseteq B(r_1)$, or $H(r) = H(r_2)$ and $B(r) \subseteq B(r_2)$;
(ii) there are rules $r'_1, r'_2 \in P_j$, $m < j \leq n$, such that $H(r_k) = H(r'_k)$ and $B(r'_k) \subseteq B(r_k)$, $k \in \{1, 2\}$, and no rule $r \in P_{j'}$ exists with $j < j' \leq n$ and $H(r) = H(r_1)$ or $H(r) = H(r_2)$; or
(iii) $B(r_1) \cup B(r_2)$ is unsatisfiable.

Then:

$$P_1 \lhd \cdots \lhd P_n \equiv_{\mathcal{A}} (P_1 \lhd \cdots \lhd P_m) \lhd P_{m+1} \lhd \cdots \lhd P_n.$$

The proof of this theorem is technically involving and is not presented here; it can be found in (Eiter *et al.*, 2000b). Observe that Conditions (i)–(iii) of Theorem 7 are simple syntactic criteria, which can be easily checked.

A weaker version of Theorem 7 may be applied if updates should be incorporated instantaneously, by only considering Condition (iii). This condition can be locally checked on each update, and is useful, e.g., if $P_1 \triangleleft P_2$ has already been constructed. Since, for any programs Q_1 and Q_2, the update program $Q_1 \triangleleft Q_2$ does not have rules with opposite heads, we can conclude from Theorem 7 that incorporating consecutive updates which obey assertion (iii) is equivalent to the update program for the sequence of updates.

Theorem 8
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$, $n \geq 2$, be an update sequence on a set of atoms $\mathcal{A}$. Suppose that, for any conflicting rules $r_1, r_2 \in P_i$, $i \leq n$, the union $B(r_1) \cup B(r_2)$ of their bodies is unsatisfiable. Then:

$$(\cdots(P_1 \triangleleft P_2) \triangleleft P_3) \cdots \triangleleft P_{n-1}) \triangleleft P_n \equiv_{\mathcal{A}} P_1 \triangleleft P_2 \triangleleft P_3 \triangleleft \cdots \triangleleft P_n.$$

In certain cases, the assertions in Theorem 7 can be dropped. One such case is a repeated update, i.e., (P_1, P_2, P_2); see (Eiter *et al.*, 2000b) for more details.

5 Refined Semantics: Minimal and Strictly Minimal Answer Sets

A property which update programs intuitively do not respect is *minimality of change*. In general, it is desirable to incorporate a new set of rules P_2 into an existing program P_1 with as little change as possible. This, of course, requests us to specify how similarity (or difference) between programs is understood and, furthermore, how proximity of programs is measured. In particular, the question is whether similarity should be model-based, or syntactically defined.

Since the semantics of update programs depends on syntax, a pure model-based notion of similarity between logic programs seems less appealing for defining minimality of change. A natural approach for measuring the change which P_1 undergoes by an update with P_2 is by considering those rules in P_1 which are abandoned. This leads us to prefer an answer set S_1 of $\boldsymbol{P} = (P_1, P_2)$ over another answer set S_2 if S_1 satisfies a larger set of rules from P_1 than S_2.

Definition 5
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be an update sequence. An answer set $S \in \mathcal{U}(\boldsymbol{P})$ is *minimal* iff there is no $S' \in \mathcal{U}(\boldsymbol{P})$ such that $Rej(S', \boldsymbol{P}) \subset Rej(S, \boldsymbol{P})$.

Example 2
Consider the sequence (P_1, P_2, P_3) from Example 1. Assume that the following additional update is received, describing that a TV can also be turned off:

$$\begin{aligned} P_4 = \{\, & r_8 : \ \mathit{switched\_off} \leftarrow \mathit{not}\ \mathit{tv\_on}, \mathit{not}\ \mathit{power\_failure};\\ & r_9 : \ \mathit{tv\_on} \leftarrow \mathit{not}\ \mathit{switched\_off}, \mathit{not}\ \mathit{power\_failure};\\ & r_{10} : \ \neg \mathit{tv\_on} \leftarrow \mathit{switched\_off};\\ & r_{11} : \ \neg \mathit{switched\_off} \leftarrow \mathit{tv\_on}\ \}. \end{aligned}$$

While (P_1, P_2, P_3) has the single answer set $S_1 = \{\mathit{night}, \neg \mathit{power\_failure}, \mathit{tv\_on}, \mathit{watch\_tv}\}$, the new sequence $\boldsymbol{P} = (P_1, P_2, P_3, P_4)$ has two answer sets: $S_1 \cup \{\neg \mathit{switched\_off}\}$ and, additionally, $S_2 = \{\mathit{night}, \neg \mathit{power\_failure}, \mathit{switched\_off}, \neg \mathit{tv\_on}, \mathit{sleep}\}$. Both answer sets reject rule r_6, but S_2 rejects r_3, too. Thus, S_1 is minimal and, corresponding to our intuition, should be preferred to S_2.

Minimal answer sets put no further emphasis on the temporal order of updates. Rules in more recent updates may be violated in order to satisfy rules from previous updates. Eliminating this possibility leads us to the following notion:

Definition 6
Let $S, S' \in \mathcal{U}(\boldsymbol{P})$, for some update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$. Then, S is *preferred* over S' iff some $i \in \{1, \ldots, n\}$ exists such that (i) $Rej_i(S, P) \subset Rej_i(S', P)$, and (ii) $Rej_j(S, P) = Rej_j(S', P)$, for all $j = i + 1, \ldots, n$. An answer set S of $\boldsymbol{P}$ is *strictly minimal*, if no $S' \in \mathcal{U}(\boldsymbol{P})$ exists which is preferred over S.

Example 3
Suppose in the previous example we had observed that the TV was off when the power returned, i.e., replace P_3 in (P_1, P_2, P_3, P_4) by:

$$P'_3 \quad = \quad \{\ r_7 : \ \neg \mathit{power\_failure} \leftarrow, \quad r'_7 : \ \neg \mathit{tv\_on} \leftarrow \ \}.$$

The modified update sequence $\boldsymbol{P'} = (P_1, P_2, P'_3, P_4)$ yields the same answer sets as before:

$$S_1 \quad = \quad \{\mathit{night}, \neg \mathit{power\_failure}, \neg \mathit{switched\_off}, \mathit{tv\_on}, \mathit{watch\_tv}\};$$
$$S_2 \quad = \quad \{\mathit{night}, \neg \mathit{power\_failure}, \mathit{switched\_off}, \neg \mathit{tv\_on}, \mathit{sleep}\}.$$

However, now both answer sets are minimal: We have $Rej(S_1, \boldsymbol{P'}) = \{r'_7, r_6\}$ and $Rej(S_2, \boldsymbol{P'}) = \{r_3, r_6\}$. Thus, $Rej(S_1, \boldsymbol{P'})$ and $Rej(S_2, \boldsymbol{P'})$ are incomparable, and hence both S_1 and S_2 are minimal answer sets. Since in S_1 the more recent rule of P'_3 is violated, S_2 is the unique strictly minimal answer set.

We denote by $Bel_{min}(\boldsymbol{P})$ the set of all rules which are true in any minimal answer set of an update sequence $\boldsymbol{P}$. Likewise, $Bel_{str}(\boldsymbol{P})$ denotes the set of all rules which are true in all strictly minimal answer sets of $\boldsymbol{P}$.

Let us consider some further example stressing the difference between regular update answer sets, minimal answer sets, and strictly minimal answer sets.

Example 4
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 by one of the sources that there is no concert on Friday:

$$P_1 \ = \ \{\ r_1 : \ \neg \mathit{concert\_friday} \leftarrow \ \}.$$

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

$$P_2 \ = \ \{\ r_2 : \ \neg \mathit{final\_rehearsal\_friday} \leftarrow, \quad r_3 : \ \neg \mathit{concert\_saturday} \leftarrow \ \}.$$

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 Friday, a concert on Saturday or Sunday follows:

$$P_3 \ = \ \{\ r_4 : \ \mathit{concert\_friday} \leftarrow \mathit{not}\ \mathit{final\_rehearsal\_friday};$$
$$r_5 : \ \mathit{final\_rehearsal\_friday} \leftarrow \mathit{not}\ \mathit{concert\_friday};$$

$$r_6: \ concert\_saturday \leftarrow final\_rehearsal\_friday, not\ concert\_sunday;$$
$$r_7: \ concert\_sunday \leftarrow final\_rehearsal\_friday, not\ concert\_saturday\ \}.$$

The update sequence $\boldsymbol{P} = (P_1, P_2, P_3)$ yields three answer sets:

$$S_1 = \{final\_rehearsal\_friday, \neg concert\_friday, concert\_saturday\};$$
$$S_2 = \{final\_rehearsal\_friday, \neg concert\_friday, \neg concert\_saturday, concert\_sunday\};$$
$$S_3 = \{\neg final\_rehearsal\_friday, concert\_friday, \neg concert\_saturday\},$$

The corresponding rejection sets are:

$$\begin{aligned} Rej(S_1, \boldsymbol{P}) &= \{r_2, r_3\}; \\ Rej(S_2, \boldsymbol{P}) &= \{r_2\}; \\ Rej(S_3, \boldsymbol{P}) &= \{r_1\}. \end{aligned}$$

Thus, S_2 and S_3 are minimal answer sets, with S_3 being the single strictly minimal answer set of $\boldsymbol{P}$.

Clearly, every strictly minimal answer set is minimal, but not vice versa. It is easily seen that for the case of update sequences involving only two update programs, i.e., for update sequences of the form $\boldsymbol{P} = (P_1, P_2)$, the notions of strictly minimal answer sets and minimal answer sets coincide. As for the AGM postulates, inspection shows that minimal and strictly minimal answer sets satisfy the same postulates as regular update answer sets, with the exception that (K3) and (K4) hold for the former ones.

Concerning the implementation of minimal and strictly minimal answer sets, in Section 6.2 we will show how they can be characterized in terms of ELPs.

6 Computational Issues

6.1 Complexity

In this section, we address the computational complexity of update programs. We assume that the reader is familiar with the basic concepts of complexity theory; e.g., (Johnson, 1990) and (Papadimitriou, 1994) are good sources (for complexity results in logic programming, cf. (Schlipf, 1995; Eiter & Gottlob, 1995; Dantsin *et al.*, 1997)). In our analysis, we focus on the case of finite, propositional update sequences.

We briefly recall the definitions of the complexity classes relevant in the following analysis. The class NP consists of all decision problems which are solvable in polynomial time using a nondeterministic Turing machine, and Σ_2^P is the class of all decision problems solvable by a nondeterministic Turing machine in polynomial time with access to an oracle for problems in NP (Σ_2^P is also written as $\mathrm{NP}^{\mathrm{NP}}$). Furthermore, coNP refers to the class of problems whose complementary problems are in NP, and Π_2^P contains the complements of the problems in Σ_2^P.[4] All the men-

[4] Two decision problems, D_1 and D_2, are complementary (or, D_1 and D_2 are complements of each other) if it holds that I is a yes-instance of D_1 exactly if I is a no-instance of D_2.

tioned classes belong to the *polynomial hierarchy*: NP and coNP are at the first level of the hierarchy, and Σ_2^P and Π_2^P are the second level. As well, NP $\subseteq \Sigma_2^P$ and coNP $\subseteq \Pi_2^P$. It is widely held that these inclusions are proper.

It is clear that the complexity of normal logic programs, which resides at the first level of the polynomial hierarchy (Marek & Truszczyński, 1991), is a lower bound for the complexity of update programs. For arbitrary updates, the complexity does not increase, even if we consider a sequence of updates.

Theorem 9
Given an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over a set of atoms $\mathcal{A}$, then:

(i) determining whether $\boldsymbol{P}$ has an answer set is NP-complete;
(ii) determining whether $L \in Bel(\boldsymbol{P})$ for some literal L is coNP-complete.

Hardness holds in both cases for $n = 1$.

Proof
The program $\boldsymbol{P}_{\triangleleft} = P_1 \triangleleft P_2 \triangleleft \cdots \triangleleft P_n$ can obviously be generated in polynomial time from $\boldsymbol{P} = (P_1, \ldots, P_n)$. Furthermore, deciding consistency of $\boldsymbol{P}_{\triangleleft}$ is in NP, and checking whether $L \in Bel(\boldsymbol{P}_{\triangleleft})$ is in coNP. This proves membership. NP-hardness and coNP-hardness of the respective tasks is inherited from the complexity of normal logic programs (Marek & Truszczyński, 1991). □

Under minimal updates, the complexity of updates increases by one level in the polynomial hierarchy. This is no surprise, though, and parallels analogous results on update logic programs by Sakama and Inoue (1999) as well as previous results on updating logical theories and iterated circumscription (Eiter & Gottlob, 1992; Eiter & Gottlob, 1995).

Theorem 10
Given an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over a set of atoms $\mathcal{A}$ and some rule r, the following two problems are Π_2^P-complete:

(i) determining whether $r \in Bel_{min}(\boldsymbol{P})$; and
(ii) determining whether $r \in Bel_{str}(\boldsymbol{P})$.

Hardness holds even if $n = 2$.

Proof
We first show that the two tasks are in Π_2^P. We treat only task (i); the case of task (ii) is analogous. In order to show membership of (i) in Π_2^P, we show that the complementary problem is in Σ_2^P. To disprove $r \in Bel_{min}(\boldsymbol{P})$, we can construct the update program $\boldsymbol{P}_{\triangleleft} = P_1 \triangleleft P_2 \triangleleft \cdots \triangleleft P_n$ in polynomial time from $\boldsymbol{P}$ and guess an answer set $\check{S} \subseteq \mathcal{A}^*$ of $\boldsymbol{P}_{\triangleleft}$ such that $\check{S} \not\models r$ and where $S = \check{S} \cap \mathcal{A}$ is a minimal answer set of $\boldsymbol{P}$ (recall that $S \in \mathcal{U}(\boldsymbol{P})$ is minimal iff there is no $T \in \mathcal{U}(\boldsymbol{P})$ such that $Rej(T, \boldsymbol{P}) \subset Rej(S, \boldsymbol{P})$). With the help of an NP-oracle, the guess for $\check{S}$ can be verified in polynomial time. This concludes the proof that checking whether $r \notin Bel_{min}(P)$ is in Σ_2^P.

Hardness of both tasks is shown by a simple reduction from the Π_2^P-hard irrelevance test in abduction from normal logic programs (Eiter *et al.*, 1997b), which is the following problem: Given a normal logic program P, a set of atoms H, a set of literals M, and an atom $h_0 \in H$, decide whether h_0 is not contained in any minimal brave explanation of M, i.e., decide whether $h_0 \notin E$ holds for each minimal $E \subseteq H$ (with respect to inclusion) such that $P \cup E$ has some stable model S with $S \models L$, for all $L \in M$.

The reduction is defined as follows: For each $h \in H$, let h' and h'' be fresh atoms, and consider the update sequence $\boldsymbol{P} = (P_1, P_2)$, where

$$\begin{aligned} P_1 &= \{\neg h' \leftarrow \mid h \in H\}, \\ P_2 &= P \cup \{ \leftarrow \mathit{not}\, L \mid L \in M\} \cup \{h \leftarrow h',\, h' \leftarrow \mathit{not}\, h'',\, h'' \leftarrow \mathit{not}\, h' \mid h \in H\}. \end{aligned}$$

It can be shown that there is a one-to-one correspondence between the rejection sets $\mathit{Rej}(S, \boldsymbol{P})$ of the minimal answer sets S of $\boldsymbol{P}$ and the minimal brave explanations E for M. In particular, the rule $\neg h_0' \leftarrow$ is in $\mathit{Rej}(S)$ iff h_0 belongs to the corresponding minimal explanation E. It follows that $h_0 \leftarrow \in \mathit{Bel}_{min}(\boldsymbol{P})$ iff h_0 is not contained in any brave explanation for M, which establishes Π_2^P-hardness of (i). Since the notions of minimal and strictly minimal answer sets coincide for update sequences of length 2, we have that $\mathit{Bel}_{min}(\boldsymbol{P}) = \mathit{Bel}_{str}(\boldsymbol{P})$. Thus, Π_2^P-hardness of (ii) holds as well. □

Similar results hold in the approach of Inoue and Sakama (1999). Furthermore, they imply that minimal and strictly minimal answer sets can be polynomially translated into disjunctive logic programming, which can serve as a basis for implementation purposes. The next section deals with some algorithmic issues.

6.2 Implementation

The notion of update sequence can be easily extended to the case where rules may contain variables. As usual, the semantics of a program P containing variables is defined in terms of the semantics of its ground instances P^* over the Herbrand base. Rules r with variables $\mathbf{X} = X_1, \ldots, X_n$ are denoted $r(\mathbf{X})$, and rejection of r is represented by a predicate $\mathit{rej}_r(\mathbf{X})$; further details, can be found in (Eiter *et al.*, 2000b). In the rest of this section, we consider function-free update sequences $\boldsymbol{P}$.

Since answer sets of (first-order) update sequences are defined by answer sets of (first-order) ELPs, it is relative straightforward to implement the current update approach. In fact, an implementation can be built on top of existing solvers for answer set semantics. In the present case, we implemented updates as a front-end for the logic programming tool `DLV` (Eiter *et al.*, 1997a; Eiter *et al.*, 1998). The latter system is a state-of-the-art solver for *disjunctive logic programs* (DLPs) under the answer set semantics. It allows for non-ground rules and calculates answer sets by performing a reduction to the stable model semantics. (Another highly efficient logic programming implementation, realizing stable model semantics, is the system `smodels` (Niemelä & Simons, 1996), which would similarly fit as underlying reasoning engine. We chose `DLV` because of familiarity and its optimization techniques for grounding, as well as its expressiveness which would allow an integral solution to compute strict and strictly minimal answer sets, respectively.) Formally, DLPs are defined as ELPs where disjunctions may appear in the head of rules; the answer set semantics for DLPs is due to Gelfond and Lifschitz (1991).

The implemented tasks agree with the decision problems discussed in the previous section, i.e., they comprise the following problems: (i) checking the existence of an answer set for a given update sequence, (ii) brave reasoning, and (iii) skeptical reasoning; as well as the corresponding problems for minimal and strictly minimal answer sets. All of these tasks have been realized for first-order update sequences, employing the advanced grounding mechanism of `DLV`.

As regards the implementation for minimal and strictly minimal update answer sets, although in principle it is possible to express the corresponding reasoning tasks in terms of DLPs (which is a consequence of Theorem 10 and well-known expressibility results for the disjunctive answer set semantics (Eiter *et al.*, 1997c)), we chose instead to pursue a two-step evaluation approach for our purposes, remaining within the present non-disjunctive framework, and, at the same time, adhering more closer to the underlying intuitions. Roughly speaking, this two-step approach can be described as follows: First, all candidates for minimal (strictly minimal) answer sets are calculated, i.e., all answer sets of the update program $\boldsymbol{P}_{\triangleleft}$. Afterwards, every candidate is tested for being minimal (strictly minimal).

Testing a candidate, S, for minimality (strict minimality) is performed by evaluating a test program, $\boldsymbol{P}_S^{min}$ ($\boldsymbol{P}_S^{strict}$), consisting of the rules of $\boldsymbol{P}_{\triangleleft}$ and a set of additional rules. Intuitively, the additional rules constrain the answer sets of $\boldsymbol{P}_S^{min}$ ($\boldsymbol{P}_S^{strict}$) to those answer sets of $\boldsymbol{P}_{\triangleleft}$ having a smaller set of rejected rules compared to the rules rejected by S (or to those answer sets of $\boldsymbol{P}_{\triangleleft}$ preferred over S, respectively). Hence, the candidate S is minimal (strictly minimal) if the corresponding test program $\boldsymbol{P}_S^{min}$ ($\boldsymbol{P}_S^{strict}$) has no answer set. In the following subsections, the implementation approach is described more formally.

6.2.1 Minimal Answer Sets

Definition 7
Let $\boldsymbol{P}_{\triangleleft} = P_1 \triangleleft \ldots \triangleleft P_n$ be a (first-order) update program and S an answer set of $\boldsymbol{P}_{\triangleleft}$. Let *ok* be a new nullary predicate symbol (i.e., propositional atom) and, for each predicate symbol rej_r occurring in $\boldsymbol{P}_{\triangleleft}$, let s_r be a new predicate symbol of the same arity as rej_r. Then, the minimality-test program with respect to S, $\boldsymbol{P}_S^{min}$, consists of all rules and constraints of $\boldsymbol{P}_{\triangleleft}$, together with the following items:

(i) for each predicate symbol rej_r occurring in $\boldsymbol{P}_{\triangleleft}$:

$$\leftarrow \quad rej_r(\mathbf{X}), not\, s_r(\mathbf{X});$$

(ii) for each ground formula $rej_r(\mathbf{t}) \in S$:

$$\begin{aligned} ok &\leftarrow not\, rej_r(\mathbf{t}); \\ s_r(\mathbf{t}) &\leftarrow \; ; \end{aligned}$$

(iii) the constraint

$$\leftarrow \quad not\, ok.$$

Algorithm Compute_Minimal_Models($\boldsymbol{P}$)

Input: A sequence of ELPs $\boldsymbol{P} = (P_1, \ldots, P_n)$.
Output: All minimal answer sets of $\boldsymbol{P}$.

```
var Cands : SetOfAnswerSets;
var MinModels : SetOfAnswerSets;
Cands := Compute_Answer_Sets(P_◁);
for all S ∈ Cands do
        var Counter : SetOfAnswerSets;
        Counter := Compute_Answer_Sets(P_S^min);
        if (Counter = ∅) then
                MinModels := MinModels ∪ {S};
        fi
rof
return MinModels;
```

Fig. 1. Algorithm to calculate minimal answer sets.

Note that in the above definition, only the rules and facts of (ii) manifest the dependence of $\boldsymbol{P}_S^{min}$ from S. Informally, the constraints (i) eliminate all answer sets with rejection sets which cannot be subsets of $Rej(S, \boldsymbol{P})$, i.e., which reject at least one rule not rejected in S. In the remaining answer sets, if any, either ok is true, i.e., at least one rule which is rejected in S is not rejected in such a set, or ok is false, in which case their rejection sets equal $Rej(S, \boldsymbol{P})$, and thus these sets are eliminated by Constraint (iii). Actually, the following proposition holds:

Theorem 11
Let S be an answer set of $\boldsymbol{P}_\lhd$. Then, S is a minimal answer set of $\boldsymbol{P}_\lhd$ iff $\boldsymbol{P}_S^{min}$ has no answer set.

Proof
Only-if part. Suppose $\boldsymbol{P}_S^{min}$ has an answer set S'. Then ok must be true in S' due to the constraint (iii) of Definition 7. Since rules (ii) of Definition 7 are the only ones in $\boldsymbol{P}_S^{min}$ with head ok, there exists a ground term $rej_r(\mathbf{t}) \in S$ such that $rej_r(\mathbf{t}) \notin S'$. Moreover, no ground term $rej_{r'}(\mathbf{t}) \in S' \setminus S$ can exist due to the constraints (i) of Definition 7. (Observe that $rej_{r'}(\mathbf{t}) \notin S$ implies $s_{r'}(\mathbf{t}) \notin S'$; hence, if $rej_{r'}(\mathbf{t}) \in S'$, then the body of one of the constraints is true in S'.) This proves $Rej(S', \boldsymbol{P}_S^{min}) = Rej(S', \boldsymbol{P}) \subset Rej(S, \boldsymbol{P})$.

Since the predicate symbols ok and s_r do not occur in $\boldsymbol{P}_\lhd$, and $\boldsymbol{P}_S^{min}$ contains all rules and constraints of $\boldsymbol{P}_\lhd$, results on the splitting of logic programs (Lifschitz & Turner, 1994) imply that $\tilde{S} = S' \setminus (\{ok\} \cup \{s_r(\mathbf{t}) \mid s_r(\mathbf{t}) \in S'\})$ is an answer set of $\boldsymbol{P}_\lhd$. Given that $Rej(\tilde{S}, \boldsymbol{P}) = Rej(S', \boldsymbol{P}) \subset Rej(S, \boldsymbol{P})$, we obtain that S is not minimal.

If part. Suppose S is not minimal. Then there exists an answer set $\tilde{S}$ of $\boldsymbol{P}_\lhd$ with $Rej(\tilde{S}, \boldsymbol{P}) \subset Rej(S, \boldsymbol{P})$. Consider $S' = \tilde{S} \cup \{ok\} \cup \{s_r(\mathbf{t}) \mid rej_r(\mathbf{t}) \in S\}$. It is easily verified that $\tilde{S}$ is an answer set of $\boldsymbol{P}_S^{min}$. □

This result allows us to calculate all minimal answer sets of $\boldsymbol{P}_\lhd$ using the straightforward algorithm depicted in Figure 1, which proceeds as follows: compute all answer sets of $\boldsymbol{P}_\lhd$ and check for every answer set S if the corresponding minimality-test program $\boldsymbol{P}_S^{min}$ has at least one answer set. If not, then add S to the set of minimal answer sets of $\boldsymbol{P}_\lhd$.

6.2.2 Strictly Minimal Answer Sets

Definition 8

Let $\boldsymbol{P}_\lhd = P_1 \lhd \ldots \lhd P_n$ be a (first-order) update program and S an answer set of $\boldsymbol{P}_\lhd$. Let ok, ok_i $(1 \leq i \leq n)$, and eq_i $(1 \leq i \leq n+1)$ be new nullary predicate symbols, and, for each predicate symbol rej_r occurring in $\boldsymbol{P}_\lhd$, let s_r be a new predicate symbol of the same arity as rej_r. Then, the program $\boldsymbol{P}_S^{strict}$ consists of all rules and constraints of $\boldsymbol{P}_\lhd$, together with the following items:

(i) for each predicate rej_r occurring in $\boldsymbol{P}_\lhd$, corresponding to $r \in P_i$:

$$\leftarrow \quad rej_r(\mathbf{X}), not\, s_r(\mathbf{X}), eq_{i+1};$$

(ii) for each ground term $rej_r(\mathbf{t}) \in S$, corresponding to $r \in P_i$:

$$\begin{aligned} ok_i &\leftarrow not\, rej_r(\mathbf{t}), eq_{i+1}; \\ s_r(\mathbf{t}) &\leftarrow \ ; \end{aligned}$$

(iii) for $1 \leq i \leq n$:

$$\begin{aligned} eq_i &\leftarrow eq_{i+1}, not\, ok_i; \\ ok &\leftarrow ok_i; \end{aligned}$$

(iv) the constraint

$$\leftarrow \quad not\, ok$$

and the fact

$$eq_{n+1} \quad \leftarrow \quad .$$

Again, program $\boldsymbol{P}_S^{strict}$ depends on S only in virtue of Item (ii). The constraints of Item (i) eliminate all answer sets S' which cannot be preferred over S because at some level i they reject a rule not rejected in S, and $Rej_j(S, \boldsymbol{P}) = Rej_j(S', \boldsymbol{P})$ holds for $j = i+1, \ldots, n$ (expressed by eq_{i+1}). In the remaining answer sets, if there is any, ok is either true, or false. If ok is true in S', then ok_i is true in S' for some level i, i.e., S' does not reject a rule of P_i which is rejected in S, and $Rej_j(S, \boldsymbol{P}) = Rej_j(S', \boldsymbol{P})$ for $j = i+1, \ldots, n$. In this case, S' is preferred over S. If, however, ok is false in S', then $Rej_i(S, \boldsymbol{P}) = Rej_i(S', \boldsymbol{P})$, for $i = 1, \ldots, n$, and S' is killed by the constraint of Item (iv).

An equivalent result as for minimality-test programs holds for the above test programs as well. Hence, the same algorithm using $\boldsymbol{P}_S^{strict}$ instead of $\boldsymbol{P}_S^{min}$ can be used to compute all strictly minimal answer sets of $\boldsymbol{P}_\lhd$.

Theorem 12
Let S be an answer set of $\boldsymbol{P}_{\lhd}$. Then, S is a strictly minimal answer set of $\boldsymbol{P}_{\lhd}$ iff $\boldsymbol{P}_S^{strict}$ has no answer set.

Proof
Only-if part. Suppose $\boldsymbol{P}_S^{strict}$ has an answer set S'. Then ok must be true in S', due to Constraint (iv) of Definition 8. Since the rules of Item (iii) of Definition 8 are the only ones in $\boldsymbol{P}_S^{strict}$ with head ok, there exists some i, $1 \leq i \leq n$, such that $ok_i \in S'$. This implies that the body of the corresponding rule of (ii) must be true in S'. Hence, there exists a ground term $rej_r(\mathbf{t}) \in S \setminus S'$, where $r \in P_i$ and such that $eq_{i+1} \in S'$. Moreover, except for the fact $eq_{n+1} \leftarrow$, the rules of (iii) are the only ones in $\boldsymbol{P}_S^{strict}$ with eq predicate symbols in their heads, so that eq_{i+1} implies $eq_j \in S'$ and $ok_j \notin S'$ for $j = i+1, \ldots, n$ if $i < n$. From this, and the constraints of (i), it follows that no ground term $rej_{r'}(\mathbf{t})S' \setminus S$, $r' \in P_j$, $j = i, \ldots, n$, can exist ($rej_{r'}(\mathbf{t}) \notin S$ implies $s_{r'}(\mathbf{t}) \notin S'$; hence, having $rej_{r'}(\mathbf{t}) \in S'$ and $eq_{j+1} \in S'$, the body of one of the constraints is true in S'). It also follows that for every $r' \in P_j$, $j = i+1, \ldots, n$, if $rej_{r'}(\mathbf{t}) \in S$, then $rej_{r'}(\mathbf{t}) \in S'$ (otherwise the body of one of the rules of (ii) would be true in S', implying $ok_j \in S'$, a contradiction).

Summarizing, we have shown $Rej_i(S', \boldsymbol{P}_S^{strict}) = Rej_i(S', \boldsymbol{P}_{\lhd}) \subset Rej_i(S, \boldsymbol{P}_{\lhd})$ and $Rej_j(S', \boldsymbol{P}_S^{strict}) = Rej_j(S', \boldsymbol{P}_{\lhd}) = Rej_j(S, \boldsymbol{P}_{\lhd})$, for $j = i+1, \ldots, n$. So, S' is preferred over S.

Since none of the predicate symbols ok, eq, and s_r occurs in $\boldsymbol{P}_{\lhd}$, and $\boldsymbol{P}_S^{strict}$ contains all rules and constraints of $\boldsymbol{P}_{\lhd}$, by a similar argument as in the proof of Theorem 11 (i.e., invoking splitting results from (Lifschitz & Turner, 1994)) it follows that

$$\tilde{S} = S' \setminus (\{ok, ok_i\} \cup \{eq_j \mid j = i+1, \ldots, n+1\} \cup \{s_r(\mathbf{t}) \mid s_r(\mathbf{t}) \in S'\})$$

is an answer set of $\boldsymbol{P}_{\lhd}$. Given that $Rej(\tilde{S}, \boldsymbol{P}_{\lhd}) = Rej(S', \boldsymbol{P}_{\lhd})$, we obtain that $\tilde{S}$ is preferred over S. Consequently, S is not a strictly minimal answer set.

If part. Suppose S is not a strictly minimal answer set. Then there exists an answer set $\tilde{S}$ of $\boldsymbol{P}_{\lhd}$ which is preferred over S. In particular, there exists some i, $1 \leq i \leq n$, such that $Rej_j(\tilde{S}, \boldsymbol{P}_{\lhd}) = Rej_j(S, \boldsymbol{P}_{\lhd})$ for $j = i+1, \ldots, n$ and $Rej_i(\tilde{S}, \boldsymbol{P}_{\lhd}) \subset Rej_i(S, \boldsymbol{P}_{\lhd})$. Consider

$$S' = \tilde{S} \cup \{ok, ok_i\} \cup \{eq_j \mid j = i+1, \ldots, n+1\} \cup \{s_r(\mathbf{t}) \mid rej_r(\mathbf{t}) \in S\}.$$

It is easily verified that $\tilde{S}$ is an answer set of $\boldsymbol{P}_S^{strict}$. □

7 Relations to Other Approaches

In this section, we analyze the relations between the current update framework and other formalisms. First of all, we discuss the connection with inheritance programs by Buccafurri *et al.* (1999a), which has not been introduced as a formalism for updates but can be successfully interpreted as such, coming to an equivalence result with our update sequences over the common fragment.

In a second step, we study the relation on the one hand to the approach of

Leite and Pereira (1997), also modeling Revision Programming by Marek and Truszczyński (1994), on the other hand to dynamic logic programming (Alferes *et al.*, 1998; Alferes *et al.*, 2000), which is close in spirit to the present update method, in the sense that update sequences are translated to standard logic programs. In particular, we describe the semantical differences between our update programs and dynamic programs, and show that both proposals have problems on certain examples.

Then, we briefly discuss update approaches for logic programs based on preference handling (Zhang & Foo, 1998) and abduction (Inoue & Sakama, 1999). Finally, we mention a method due to Delgrande, Schaub, and Tompits (2000) for handling preference information in the context of logic programs, which is also based on an encoding to ELPs.

7.1 Relation to Inheritance Programs

The update semantics we suggest resolves conflicts by assigning "preference" to the more recent information. As already pointed out earlier, this can also be interpreted as some form of inheritance mechanism, where the more recent information is considered as being more specific. In this section, we discuss this aspect in more detail. To wit, we consider the inheritance approach introduced by Buccafurri *et al.* (1999a) and we show that update sequences can equivalently be described in terms of inheritance programs.

In what follows, we briefly describe the basic layout of the inheritance approach by Buccafurri *et al.* (1999a). Since that method has originally been specified for non-ground DLPs, and we deal here only with non-disjunctive ELPs, we adapt some of the original definitions accordingly.

A DLP$^<$*-program*, $P^<$, is a finite set $\{\langle o_1, P_1\rangle, \ldots, \langle o_n, P_n\rangle\}$ of object identifiers o_i ($1 \leq i \leq n$) and associated ELPs P_i, together with a strict partial order "$<$" between object identifiers (pairs $\langle o_i, P_i\rangle$ are called *objects*).[5] As well, we say that $P^<$ is a DLP$^<$-program *over* a set of atoms $\mathcal{A}$ iff $\mathcal{A}$ denotes the set of all atoms appearing in $P^<$. Informally, possible conflicts in determining properties of the objects are resolved in favor of rules which are *more specific* according to the hierarchy, in the sense that rule $r \in P_k$ is considered to represent more specific information than rule $r' \in P_l$ whenever $o_k < o_l$ holds ($1 \leq k, l \leq n$ and $k \neq l$). In the following, $\rho(P^<)$ denotes the multiset of all rules appearing in the programs of $P^<$.

Consider some DLP$^<$-program $P^<$ over a set of atoms $\mathcal{A}$. Let $I \subseteq Lit_{\mathcal{A}}$ be an interpretation and let $r_1 \in P$ and $r_2 \in P'$ be two conflicting rules, where $\langle o, P\rangle, \langle o', P'\rangle \in P^<$. Then, r_1 *overrides* r_2 *in* I iff (i) $o < o'$, (ii) $H(r_1)$ is true in I, and (iii) $B(r_2)$ is true in I. A rule $r \in \rho(P^<)$ is *overridden in* I iff there exists some $r' \in \rho(P^<)$ which overrides r in I.

An interpretation $I \subseteq Lit_{\mathcal{A}}$ is a *model* of $P^<$ iff every rule in $\rho(P^<)$ is either

[5] Strictly speaking, in the current context, the term "DLP$^<$-program" (as introduced by Buccafurri *et al.* (1999a)) is a bit of a misnomer, because "DLP" points to *disjunctive* logic programs; however, we retained the original name for reference's sake.

overridden or true in I; moreover, I is *minimal* iff it is the least model of all these rules. The *reduct*, $G_I(P^<)$, of the DLP$^<$-program $P^<$ *relative* to I results from $\rho(P^<)$ by (i) deleting any rule $r \in \rho(P^<)$ which is either overridden in I or defeated by I; and (ii) deleting all weakly negated literals in the bodies of the remaining rules of $\rho(P^<)$. Then, I is an *answer set* of $P^<$ iff it is a minimal model of $G_I(P^<)$.

This concludes our brief review of the inheritance framework of (Buccafurri *et al.*, 1999a); we continue with our correspondence result.

Theorem 13
$S \subseteq Lit_{\mathcal{A}}$ is an answer set of the update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over $\mathcal{A}$ iff S is an answer set of the DLP$^<$-program $P^< = \{\langle o_1, P_1\rangle, \ldots, \langle o_n, P_n\rangle\}$ having inheritance order $o_n < o_{n-1} < \cdots < o_1$.

Proof
We first note the following two properties, which can be verified in a straightforward way. Let $I \subseteq Lit_{\mathcal{A}}$ be some interpretation and $r \in \rho(P^<)$. Then:

(i) If $r \in Rej(I, \boldsymbol{P})$, then r is overridden in I.
(ii) Assume I satisfies for any $r' \in \rho(P^<)$ the condition that $B(r')$ is true in I whenever $H(r')$ is true in I. Then, r is overridden in I only if $r \in Rej(I, \boldsymbol{P})$.

We proceed with the proof of the main result. Suppose S is an answer set of $\boldsymbol{P} = (P_1, \ldots, P_n)$. We show S is an answer set of $P^< = \{\langle o_1, P_1\rangle, \ldots, \langle o_n, P_n\rangle\}$ with inheritance order $o_n < o_{n-1} < \cdots < o_1$.

First, we show that S is a model of $G_S(P^<)$. Consider some $r \in G_S(P^<)$. Then, there is some rule $\hat{r} \in \rho(P^<)$ such that $r = \hat{r}^+$ and $\hat{r}$ is neither overridden in S nor defeated by S. Applying Property (i), we get that $\hat{r} \notin Rej(S, \boldsymbol{P})$. Hence, $\hat{r} \in (\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$ since $\hat{r}$ is not defeated by S. Thus, given that S is an answer set of $\boldsymbol{P}$, Theorem 4 implies that $r = \hat{r}^+$ is true in S. It follows that S is a model of $G_S(P^<)$.

Next, we show that there is no proper subset of S which is also a model of $G_S(P^<)$. Suppose there is such a set $S_0 \subset S$. Since S is an answer set of $\boldsymbol{P}$, Property (ii) can be applied, and we obtain $r \in G_S(P^<)$ whenever $r \in (\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$ holds. As a consequence, S_0 is a model of $r \in (\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$, which contradicts the fact that S is an answer set of $\boldsymbol{P}$. This concludes the proof that S is an answer set of $P^<$ if S is an answer set of $\boldsymbol{P}$.

For the converse direction, assume S is an answer set of $P^<$. Similar to the argumentation given above, Property (ii) implies that S is a model of $(\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. As well, S is a minimal model of $(\cup\boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$, because otherwise Property (i) would yield a proper subset S_0 of S being a model of $G_S(P^<)$, contradicting the fact that such a subset S_0 cannot exist, because S is an answer of $P^<$. □

Inheritance programs are also related to *ordered logic programs*, due to Laenens *et al.* (1990) and further analyzed by Buccafurri *et al.* (1996). The difference between inheritance programs and ordered logic programs is that the latter ones have a built-in *contradiction removal* feature, which eliminates local inconsistencies in a given hierarchy of programs. Thus, for linearly ordered programs $P_1 < P_2 < \cdots < P_n$ where such inconsistencies do not occur, e.g., if for any two conflicting rules in P_i ($1 \leq i \leq n$) their bodies cannot be simultaneously satisfied, the above equivalence result holds for ordered logic programs as well.

7.2 Revision Programs and the Approach of Leite and Pereira

In the framework of Marek and Truszczyński (1994), a knowledge base is a collection of positive facts, and revision programs specify conditional insertions or removals of facts under a semantics very similar to the stable semantics. In discussing this approach, Leite and Pereira (1997) argued that the approach of revision programs is not adequate if more complex knowledge is represented in the form of logic programs, because revision programs compute only "model-by-model updates", which do not capture the additional information encoded by logic programs. Accordingly, they proposed an extended framework in which a suitable inertia principle for rules realizes the update independently of any specific model of the original program. In the following, we briefly sketch their approach.

In a first step, Leite and Pereira (1997) define their approach for normal logic programs, and afterwards they extend it to handle programs with strong negation as well. Furthermore, (Leite & Pereira, 1997) deals only with the situation where a given program is updated by a single program; the general case involving an arbitrary number of updates is described in (Leite, 1997). We describe here the latter approach, but, for the sake of simplicity, only the case of normal logic programs.

Following the method of revision programs (Marek & Truszczyński, 1994), an update program in the sense of (Leite & Pereira, 1997) is a finite collection of rules of the form

$$in(A) \leftarrow in(B_1), \ldots, in(B_m), out(C_1), \ldots, out(C_n), \quad \text{and} \tag{1}$$

$$out(A') \leftarrow in(B'_1), \ldots, in(B'_m), out(C'_1), \ldots, out(C'_n), \tag{2}$$

where $A, A', B_i, B'_i, C_j, C'_j$ are atoms ($1 \leq i \leq m,\ 1 \leq j \leq n$). Intuitively, Rule (1) states that A should be true given that $B_1, \ldots, B_m$ are true and $C_1, \ldots, C_n$ are false, and a similar meaning holds for Rule (2). Rule (1) is called an *in-rule*, and Rule (2) is an *out-rule*. Semantically, in-rule (1) is interpreted as the logic program rule

$$A \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n,$$

whilst out-rule (2) is interpreted as

$$\neg A \leftarrow B_1, \ldots, B_m, not\, C_1, \ldots, not\, C_n.$$

When speaking about update programs, in the following they are always identified with finite sets of rules of the above form. Let us call a sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ of such programs an *IO-sequence* (for "sequence of in- and out-rules"). Consider an IO-sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over $\mathcal{A}$, and let $S \subseteq Lit_{\mathcal{A}}$ be a set of literals.

Leite (1997) introduces the following notion of a rejection set (for $1 \leq i, j \leq n$):

$$Rejected(S,i,j) \;=\; \bigcup_{i<k\leq j}\{\; r \in P_i \mid \exists r' \in P_k \text{ such that } r \text{ and } r' \text{ are conflicting and } S \models B(r) \cup B(r') \;\}.$$

Then, $S \cap \mathcal{A}$ is a $\mathbf{P}$-*justified update at state* j $(1 \leq j \leq n)$ iff S is an answer set of

$$\bigcup_{i\leq j}(P_i \setminus Rejected(S,i,j)),$$

provided that each program $\bigcup_{i\leq l}(P_i \setminus Rejected(S,i,j))$, for $l < j$, possesses an answer set.[6]

It is easily seen that for $\boldsymbol{P} = (P_1, \dots, P_n)$ and S as above, S is an answer set of $\bigcup_{i\leq n}(P_i \setminus Rejected(S,i,j))$ iff it is an answer set of $\cup\boldsymbol{P}\setminus Rej'(S,\boldsymbol{P})$, where $Rej'(S,\boldsymbol{P})$ is the weak form of a rejection set, as defined in Section 3.2. Hence, we can state the following proposition:

Theorem 14
Let $\boldsymbol{P} = (P_1, \dots, P_n)$ be an IO-sequence over $\mathcal{A}$ and $S \subseteq Lit_{\mathcal{A}}$ a set of literals. Assume that each subsequence $(P_1, \dots, P_j)$ has an answer set, for $1 \leq j \leq n$. Then, S is an answer set of $\boldsymbol{P}$ if it is a $\boldsymbol{P}$-justified update at state n.

The converse of the above result, viz. that $\boldsymbol{P}$-justified updates at state n are answer sets of $\boldsymbol{P}$, holds under certain restrictions (cf. Section 3.2).

Concerning the extended framework of Leite and Pereira in which rules of the form (1) and (2) may contain literals instead of plain atoms, only weaker correspondences can be found, assuming that update sequences contain only in-rules.

7.3 Dynamic Logic Programming

Alferes *et al.* (1998; 2000) introduced the concept of *dynamic logic programs* as a generalization of both the idea of updating interpretations through revision programs (Marek & Truszczyński, 1994) and of updating programs as defined by Alferes and Pereira (1997) and by Leite and Pereira (1997). Syntactically, dynamic logic programs are based on *generalized logic programs* (GLPs), which allow default negation in the head of rules, but no strong negation whatsoever.

In dynamic logic programming (DynLP in the following), the models of a sequence of updates are defined as the stable models of the program resulting from a syntactic rewriting, similar to the transformation used in our approach. This is called a *dynamic update*. Elements of the sequence are GLPs.

Regarding the formalisms discussed in the previous subsection, in (Alferes *et al.*, 2000) it is demonstrated that revision programs and dynamic updates are equivalent, provided that the original knowledge is *extensional*, i.e., the initial program contains only rules of the form $A \leftarrow$ or $not\,A \leftarrow$.

[6] Strictly speaking, Leite (1997) requires that, for each $l \leq j$, $\bigcup_{i\leq l}(P_i \setminus Rejected(S,i,j))$ has a consistent answer set. However, in our setting, answer sets are always consistent.

Our analysis of dynamic updates can be summarized as follows. First, basic definitions and semantical characterizations of dynamic update programs are given. Afterwards, the relation between dynamic updates and updates according to Definition 2 is investigated. Since the two approaches are defined over different languages, the comparison must include suitable translations to take this distinction into account. As a matter of fact, Alferes *et al.* (2000) already discussed how ELPs can be handled within their framework; likewise, we define a similar translation schema such that GLPs can be treated by our update method.

As it turns out, there is a semantic difference between dynamic updates and updates according to Definition 2. Although any dynamic update is an update answer set in the sense of Definition 2, the converse does not hold in general, even if Property (CH) from Section 3.2 holds. Intuitively, this can be explained by the fact that dynamic updates are more restrictive as regards to certain circularities in the given update information. On the other hand, there are conditions under which both approaches yield equivalent results (cf. (Eiter *et al.*, 2000b)). We briefly discuss that dynamic logic programs do not eliminate all kinds of circularities.

In view of the equivalent semantics for the example programs, except in case of Addition of Tautologies, the failure of update principles we presented in Section 4 applies to dynamic logic programs as well. Satisfaction of properties may be established using similar arguments. Furthermore, similar complexity results for dynamic logic programs can be concluded, based on the constructions in Section 6.1.

7.3.1 Semantics of Dynamic Logic Programs

Given an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ of GLPs over $\mathcal{A}$, let $\mathcal{A}_{dyn}$ be $\mathcal{A}$ extended by new, pairwise distinct atoms A^-, A_i, A_i^-, A_{P_i}, $A^-_{P_i}$, and $\mathit{reject}(A_i)$, for each $A \in \mathcal{A}$ and each $i \in \{1, \ldots, n\}$. The dynamic update program $\boldsymbol{P}_{\oplus} = P_1 \oplus \cdots \oplus P_n$ over $\mathcal{A}_{dyn}$ is defined as the GLP consisting of the following items:

(i) for each $r \in P_i$, $1 \leq i \leq n$, with $B^-(r) = \{C_1, \ldots, C_n\}$:

$$\begin{array}{rcll} A_{P_i} & \leftarrow & B^+(r), C_1^-, \ldots, C_n^- & \text{if } H(r) = A; \\ A^-_{P_i} & \leftarrow & B^+(r), C_1^-, \ldots, C_n^- & \text{if } H(r) = \mathit{not}\, A; \end{array}$$

(ii) for each atom A occurring in $\boldsymbol{P}$ and each $i \in \{1, \ldots, n\}$:

$$\begin{array}{rclcrcl} A_i & \leftarrow & A_{P_i}; & \qquad & \mathit{reject}(A^-_{i-1}) & \leftarrow & A_{P_i}; \\ A^-_i & \leftarrow & A^-_{P_i}; & \qquad & \mathit{reject}(A_{i-1}) & \leftarrow & A^-_{P_i}; \end{array}$$

$$\begin{array}{rcl} A^-_i & \leftarrow & A^-_{i-1}, \mathit{not}\, \mathit{reject}(A^-_{i-1}); \\ A_i & \leftarrow & A_{i-1}, \mathit{not}\, \mathit{reject}(A_{i-1}); \end{array}$$

(iii) for each atom A occurring in $\boldsymbol{P}$:

$$A^-_0 \leftarrow\, ; \qquad A \leftarrow A_n; \qquad A^- \leftarrow A^-_n; \qquad \mathit{not}\, A \leftarrow A^-_n.$$

One major difference can immediately be identified between our update programs and dynamic updates: In dynamic updates, the value of each atom is determined from the bottom level P_1 *upwards* towards P_n (in virtue of rules $A_i \leftarrow$

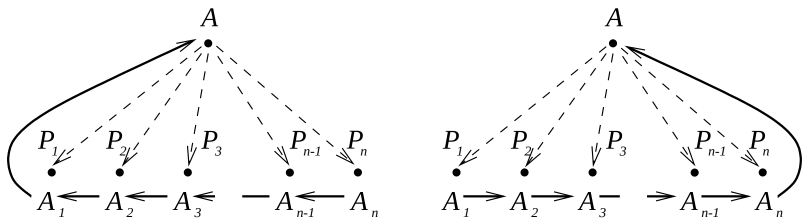

Fig. 2. “Top-down” evaluation of update sequences (left diagram) vs. “bottom-up” evaluation of dynamic logic programs (right diagram).

$A_{i-1}, \mathit{not}\, \mathit{reject}(A_{i-1})$ and $A \leftarrow A_n$ for positive atoms, and the corresponding ones for dashed atoms), whilst update programs determine such values in a *downward* fashion (cf. rules $L_i \leftarrow L_{i+1}$ and $L \leftarrow L_1$ in Definition 2). This difference is visually depicted in Figure 2. More importantly, the different evaluation strategy leads in effect to different semantics, which will be shown later on.

Before we can properly define the semantics of dynamic updates, based on the transformation $\boldsymbol{P}_{\oplus}$ introduced above, we must emphasize that Alferes *et al.* (1998; 2000) use a slightly non-standard concept of stable models. To wit, weakly negated literals $\mathit{not}\, A$ (A some atom) are treated like ordinary propositional atoms, so that rules $A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\, A_{m+1}, \ldots, \mathit{not}\, A_n$ are effectively be viewed as *propositional Horn formulas.* Accordingly, an interpretation I is in this context understood as a set consisting of atoms and weakly negated atoms such that for each atom A it holds that $A \in I$ iff $\mathit{not}\, A \notin I$. To distinguish such interpretations from interpretations in the usual sense, we call them *generalized interpretations.* As usual, a set B, comprised of atoms and weakly negated atoms, is true in a generalized interpretation I, symbolically $I \models B$, iff $B \subseteq I$. Towards defining stable models, the following notation is required:

Let $\mathcal{A}$ be a set of atoms. Then, $\mathit{not}\, \mathcal{A}$ denotes the set $\{\mathit{not}\, A \mid A \in \mathcal{A}\}$. Furthermore, for $M \subseteq \mathcal{A} \cup \mathit{not}\, \mathcal{A}$, we set $M^- = \{\mathit{not}\, A \mid \mathit{not}\, A \in M\}$, and, for $Z \in \mathcal{A} \cup \mathit{not}\, \mathcal{A}$, we define $\mathit{not}\, Z = \mathit{not}\, A$ if $Z = A$ and $\mathit{not}\, Z = A$ if $Z = \mathit{not}\, A$. For a program P over $\mathcal{A}$, the deductive closure, $\mathit{Cn}_{\mathcal{A}}(P)$, is given by the set

$$\{L \mid L \in \mathcal{A} \cup \mathit{not}\, \mathcal{A} \text{ and } P \vdash L\},$$

where P is interpreted as a propositional Horn theory and “$\vdash$” denotes classical derivability. Usually, the subscript “$\mathcal{A}$” will be omitted from $\mathit{Cn}_{\mathcal{A}}(P)$. A generalized interpretation S is a stable model of a program P iff $S = \mathit{Cn}(P \cup S^-)$.

Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be a sequence of GLPs over $\mathcal{A}$, and let I be a generalized interpretation. Alferes *et al.* (1998; 2000) introduce the following concepts:

$$\begin{aligned} \mathit{Rejected}(I, \boldsymbol{P}) &= \textstyle\bigcup_{i=1}^{n} \{r \in P_i \mid \exists r' \in P_j, \text{ for some } j \in \{i+1, \ldots, n\}, \text{ such} \\ &\qquad \text{that } H(r') = \mathit{not}\, H(r) \text{ and } I \models B(r) \cup B(r')\}; \\ \mathit{Defaults}(I, \boldsymbol{P}) &= \{\mathit{not}\, A \mid \nexists r \text{ in } \boldsymbol{P} \text{ such that } H(r) = A \text{ and } I \models B(r)\}. \end{aligned}$$

The *dynamic stable models* of $\boldsymbol{P}$ are defined as the projections $S = S' \cap (\mathcal{A} \cup \mathit{not}\, \mathcal{A})$ of the stable models of $\boldsymbol{P}_{\oplus}$. As shown by Alferes *et al.*, $S \subseteq \mathcal{A} \cup \mathit{not}\, \mathcal{A}$ is a stable model of $\boldsymbol{P}$ if and only if S satisfies the following condition:

$$S = Cn((\cup \boldsymbol{P} \setminus Rejected(S, \boldsymbol{P})) \cup Defaults(S, \boldsymbol{P})).$$

Alferes *et al.* (2000) defined also an extension of their semantics to the three-valued case: Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be a sequence of ELPs over $\mathcal{A}$. Then, a consistent set $S \subseteq Lit_{\mathcal{A}}$ is a *dynamic answer set* of $\boldsymbol{P}$ iff $S \cup \{not\, L \mid L \in Lit_{\mathcal{A}} \setminus S\}$ is a dynamic stable model of the sequence $\boldsymbol{P} = (P_1, \ldots, P_n \cup \{not\, A \leftarrow \neg A,\ not\, \neg A \leftarrow A \mid A \in \mathcal{A}\})$ of GLPs. Here, the rules in $\{not\, A \leftarrow \neg A,\ not\, \neg A \leftarrow A \mid A \in \mathcal{A}\}$ serve for emulating classical negation through weak negation.

7.3.2 Relating Dynamic Answer Sets and Update Answer Sets

Let us now define how GLPs are to be rewritten in order to constitute a valid input for update programs according to Definition 2. For any rule r, by r° we denote the rule which results from r by replacing weak negation occurring in the head of r by strong negation, i.e.,

$$r^\circ = \begin{cases} \neg A \leftarrow B(r) & \text{if } H(r) = not\, A; \\ r & \text{otherwise.} \end{cases}$$

Furthermore, for any GLP P, we define $P^\circ = \{r^\circ \mid r \in P\}$.

Definition 9
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be a sequence of GLPs over $\mathcal{A}$. Then, the update sequence $\mathcal{Q}(\boldsymbol{P})$ is given by the sequence $(P_1^\circ, \ldots, P_n^\circ \cup \{\neg A \leftarrow not\, A \mid A \in \mathcal{A})$.

Notice that the part $\{\neg A \leftarrow not\, A \mid A \in \mathcal{A}\}$ serves for making all answer sets complete. Moreover, no strong negation occurs in rule bodies of $\mathcal{Q}(\boldsymbol{P})$. Thus, application of a rule with $\neg A$ in the head can never lead to the application of further rules; it can only enable that rules with A in their heads are overridden. As well, the rules in $\{\neg A \leftarrow not\, A \mid A \in \mathcal{A}\}$ are not able to override any rule in $\mathcal{Q}(\boldsymbol{P})$.

Theorem 15
Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be any sequence of GLPs over a set of atoms $\mathcal{A}$, and let $S \subseteq \mathcal{A} \cup not\, \mathcal{A}$ be a dynamic stable model of $\boldsymbol{P}$. Then, $(S \cap \mathcal{A}) \cup \neg(\mathcal{A} \setminus S)$ is an answer set of $\mathcal{Q}(\boldsymbol{P})$.

Proof
Let $S \subseteq \mathcal{A} \cup not\, \mathcal{A}$ be a dynamic stable model of $\boldsymbol{P}$, i.e.,

$$S \;=\; Cn(\tilde{P}),$$

where $\tilde{P} = (\cup \boldsymbol{P} \setminus Rejected(S, \boldsymbol{P})) \cup Defaults(S, \boldsymbol{P})$.

Let, for each set of atoms $X \subseteq \mathcal{A} \cup not\, \mathcal{A}$, denote $X^\circ = (X \cap \mathcal{A}) \cup \{\neg A \mid not\, A \in X\}$, and let $\mathcal{Q}(\boldsymbol{P}) = (P_1', \ldots, P_n')$. Note that, for any rule r from $\boldsymbol{P}$, it holds that $S \models r$ iff $S^\circ \models r^\circ$.

We have to show that S° is an answer set of $\mathcal{Q}(\boldsymbol{P})$, i.e., in view of Theorem 4, that S° is a minimal model of $\tilde{Q} = \big(\cup\, \mathcal{Q}(\boldsymbol{P}) \setminus Rej(S^\circ, \mathcal{Q}(\boldsymbol{P}))\big)^{S^\circ}$.

We first show that for each $r \in \boldsymbol{P}$, it holds that $r \in \mathit{Rejected}(S, \boldsymbol{P})$ iff $r^\circ \in \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$.

Indeed, if $r \in \mathit{Rejected}(S, \boldsymbol{P})$, then we immediately obtain from the definitions of $\mathit{Rejected}(S, \boldsymbol{P}))$, S°, and $\mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$, that $r^\circ \in \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$. Conversely, suppose that $r^\circ \in \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$. Then, $r \in P_i$ where $i \in \{1, \ldots, n-1\}$, and some $\hat{r} \in P'_j$, where $j \in \{i+1, \ldots, n\}$, exists such that $H(r^\circ) = \neg H(\hat{r})$ and $S^\circ \models B(r^\circ) \cup B(\hat{r})$. If $\hat{r}$ stems from $\boldsymbol{P}$, i.e., $\hat{r} = r'^\circ$ for some $r' \in P_j$, then $r \in \mathit{Rejected}(S, \boldsymbol{P})$ follows. Otherwise, if $\hat{r}$ is a rule of the form $\neg A \leftarrow \mathit{not}\, A$ which has been added in $\mathcal{Q}(\boldsymbol{P})$ to P'_n, then $A \notin S^\circ$ and $H(r^\circ) = A$, which by definition means $A \notin S$ and $H(r) = A$. Since $S^\circ \models B(r^\circ)$ implies that $S \models B(r)$ and S is closed under the rules in $\tilde{P}$, it follows that $r \in \mathit{Rejected}(S, \boldsymbol{P})$ must hold. This proves the claim.

Thus, for every $r \in \tilde{P}$, we conclude that

$$S \models B(r) \cap \mathit{not}\, \mathcal{A} \text{ iff } \begin{cases} A \leftarrow B^+(r) \in \tilde{Q}, & \text{if } H(r) = A, \\ \neg A \leftarrow B^+(r) \in \tilde{Q}, & \text{if } H(r) = \mathit{not}\, A. \end{cases}$$

Consider, for any program P, the standard T_P operator and its powers, where $\mathit{not}\, A$ and $\neg A$ are viewed as propositional atoms. Since $\mathit{Rej}(S^\circ, \mathcal{Q}(\boldsymbol{P})) \subseteq \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$ and for each $\mathit{not}\, A \in \mathit{Defaults}(S, \boldsymbol{P})$ we have $\neg A \leftarrow \; \in \tilde{Q}$, it clearly holds that

$$T^k_{\tilde{P}}(\emptyset)^\circ \quad \subseteq \quad T^k_{\tilde{Q}}(\emptyset), \qquad \forall k \geq 0. \qquad (*)$$

On the other hand,

$$T^k_{\tilde{Q}}(\emptyset) \quad \subseteq \quad T_{\tilde{P}}(S)^\circ \; = \; S^\circ, \qquad \forall k \geq 0 \qquad (**)$$

holds. Indeed, note that $T_{\tilde{P}}(S)^\circ = S^\circ$ since S is a fixed point of $T_{\tilde{P}}$. For the left-hand-side inclusion, consider any rule $\hat{r} \in \tilde{Q}$ which fires in $T^k_{\tilde{Q}}(\emptyset)$. If $\hat{r}$ stems from a default rule $\neg A \leftarrow \mathit{not}\, A$ which has been added in $\mathcal{Q}(\boldsymbol{P})$ to P'_n, then $A \notin S$; hence, $\mathit{not}\, A \in S$, which means $\neg A \in S^\circ$. If $\hat{r}$ stems from a rule $\hat{r}' \in (\cup \boldsymbol{P})^\circ \setminus \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$, then $\hat{r}' = r^\circ$ for some rule $r \in \cup \boldsymbol{P} \setminus \mathit{Rejected}(S, \boldsymbol{P})$ which fires in S. Thus, $H(\hat{r})\, (= H(r)^\circ) \in S^\circ$. Finally, if $\hat{r}$ stems from a rule $\hat{r}' \in \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$, then $\hat{r}' \notin \mathit{Rej}(S^\circ, \mathcal{Q}(\boldsymbol{P}))$ and hence some $\hat{r}'' \in (\cup \boldsymbol{P})^\circ \setminus \mathit{Rej}'(S^\circ, \mathcal{Q}(\boldsymbol{P}))$ must exist such that $H(\hat{r}'') = H(r')$ and $S^\circ \models B(\hat{r}'')$. Similar as above, we conclude that $H(\hat{r})\, (= H(\hat{r}'')) \in S^\circ$.

From $(*)$ and $(**)$, we conclude

$$\mathit{lfp}(T_{\tilde{P}})^\circ \; \subseteq \; \mathit{lfp}(T_{\tilde{Q}}) \; \subseteq \; S^\circ,$$

where $\mathit{lfp}(T_P)$ denotes the least fixed point of the T_P operator. Since $\mathit{lfp}(T_{\tilde{P}})^\circ = S$, it follows $\mathit{lfp}(T_{\tilde{Q}}) = S^\circ$. This means that S° is a minimal model of $\tilde{Q}$, which completes the proof. □

Theorem 16

Let $\boldsymbol{P} = (P_1, \ldots, P_n)$ be a sequence of ELPs over $\mathcal{A}$. Suppose $S \subseteq \mathit{Lit}_{\mathcal{A}}$ is a dynamic answer set of $\boldsymbol{P}$. Then, $S \in \mathcal{U}(\boldsymbol{P})$.

Proof
Let us denote classical negation in $\boldsymbol{P}$ by $\sim A$, and rewrite S accordingly, i.e., $S' = (S \cap \mathcal{A}) \cup (\{\sim A \mid \neg A \in S\})$. Then, by combining the emulation of classical negation in $\boldsymbol{P}$ through rules $not\, A \leftarrow \sim A$ and $not \sim A \leftarrow A$ $(A \in \mathcal{A})$, and the transformation $\mathcal{Q}(\cdot)$, we obtain by Theorem 15 that the set

$$S'' \;=\; S' \cup \neg(\{A, \sim A \mid A \in \mathcal{A}\} \setminus S')$$

is an answer set of $\boldsymbol{P'} = (P_1, \ldots, P_n \cup Q)$, where each $\sim A$ is viewed as a propositional atom and Q contains for each atom $A \in \mathcal{A}$ the following rules:

$$\begin{aligned} \neg \sim A &\leftarrow A; \\ \neg A &\leftarrow \sim A; \\ \neg \sim A &\leftarrow not \ \sim A; \\ \neg A &\leftarrow not\, A. \end{aligned}$$

Observe that $\sim A \in S''$ implies $A \notin S'$, and $A \in S''$ implies $\neg A \notin S''$. Furthermore, atoms $\sim A$ or A are included in S'' due to applications of rules $r \in P_i$, $1, \leq i \leq n$, which are not rejected.

By induction on i $(0 \leq i < n)$ one can show that $Rej_{n-i}(S'', \boldsymbol{P'}) = Rej_{n-i}(S, \boldsymbol{P})$ holds. It follows that S is a minimal model of $((\cup \boldsymbol{P}) \setminus Rej(S, \boldsymbol{P}))^S$, i.e., S is an answer set of $\boldsymbol{P}$. $\square$

Theorems 15 and 16 do not hold in the converse direction, even if Property (CH) from Section 3.2 (suitably adapted for default negation in rule heads) is satisfied. This can be seen by the following example:

Example 5
Consider programs P_1, P_2, and P_2', where

$$\begin{aligned} P_1 &= \{\ \mathit{it\_is\_raining} \leftarrow\ \}; \\ P_2 &= \{\ not\ \mathit{it\_is\_raining} \leftarrow not\ \mathit{it\_is\_raining}\ \}; \\ P_2' &= \{\ \neg \mathit{it\_is\_raining} \leftarrow not\ \mathit{it\_is\_raining}\ \}. \end{aligned}$$

The sequence $\boldsymbol{P} = (P_1, P_2)$ of GLPs has one dynamic stable model, $\{\mathit{it\_is\_raining}\}$, but $\mathcal{Q}(\boldsymbol{P})$ has two answer sets, $\{\mathit{it\_is\_raining}\}$ and $\{\neg \mathit{it\_is\_raining}\}$. Likewise, the sequence $\boldsymbol{P'} = (P_1, P_2')$ of ELPs has also $\{\mathit{it\_is\_raining}\}$ as single dynamic stable model, but $\{\mathit{it\_is\_raining}\}$ and $\{\neg \mathit{it\_is\_raining}\}$ are answer sets of $\boldsymbol{P'}$.

Intuitively, the syntactic mechanism responsible for the elimination of some stable models in dynamic updates as above is the renaming of weakly negated literals in the body of rules. This renaming ensures that weakly negated literals are not derived in a cyclic way, i.e., the truth value of a weakly negated literal has to be supported by other information besides the literal itself. This, however, is in general not the case with the transformation for update programs based on Definition 2. One can find syntactic criteria, employing graph-theoretical concepts, under which both approaches yield equivalent results.

Recalling the update sequence $\boldsymbol{P} = (P_1, P_2)$ and $\boldsymbol{P'} = (P_1, P_2')$ from Example 5, the single dynamic stable model $\{\mathit{it\_is\_raining}\}$ of $\boldsymbol{P}$ seems, in the sense of inertia, more intuitive than the answer set $\{\neg it\_is\_raining\}$ of $\boldsymbol{P}' = \mathcal{Q}(\boldsymbol{P})$ because the tautological update information

$$P_3 = \{not\ it\_is\_raining \leftarrow not\ it\_is\_raining\}$$

is quite irrelevant to the fact that it is raining, as given by

$$P_1 = \{it\_is\_raining \leftarrow\}.$$

So, in some sense, the semantics of Alferes *et al.* (1998; 2000) eliminates unintended stable models, as it does not allow for cyclic derivations of negative information. However, the rewritten rule of

$$P_2' = \{\neg it\_is\_raining \leftarrow not\ it\_is\_raining\}$$

differs in that it allows to conclude that it is not raining given that there is no information whether it is raining. In this sense, both answer sets $\{it\_is\_raining\}$ and $\{\neg it\_is\_raining\}$ are, in principle, reasonable. Observing that the more intuitive answer set of $\boldsymbol{P}'$ is minimal while the other is not, one can use the notion of minimality to filter out the unintended answer set. But, in general, there exist dynamic stable models such that the corresponding answer sets are not minimal (or strictly minimal) and vice versa. Also, acyclic derivations of negative information do not always capture the intuition of inertia as shown by the following example:

Example 6
Let us consider a slight modification of Example 5, where the knowledge base

$$P = \{\ it\_is\_raining \leftarrow\ ,\ it\_is\_cloudy \leftarrow it\_is\_raining\ \}$$

is updated by the information

$$U = \{not\ it\_is\_raining \leftarrow not\ it\_is\_cloudy\},$$

which, by the same intuition of inertia, is also irrelevant to the fact that it is now actually raining and thus cloudy. However, this yields two dynamic stable models

$$S_1' = \{it\_is\_raining, it\_is\_cloudy\};$$
$$S_2' = \{not\ it\_is\_raining, not\ it\_is\_cloudy\},$$

corresponding to the answer sets

$$S_1 = S_1';$$
$$S_2 = \{\neg it\_is\_raining, \neg it\_is\_cloudy\}$$

of the rewritten ELP[7], showing that also the mechanisms enacted in DynLP do not completely avoid cyclic derivations.

Despite their differences, the general properties of program updates, as investigated in Section 4, hold for dynamic logic programs also. One can easily verify that

[7] Similar to Example 5, here, the intuitively preferred answer set is also a minimal answer set, while the other is not.

every counterexample for an invalid property belongs to a class where update answer sets and dynamic stable models coincide. As well, arguments similar to those used for the demonstrations of the valid properties of Section 4 can be found in order to show that these properties also hold for dynamic logic programs.

7.4 Program Updates Through Abduction

The use of abduction for solving update problems in logic programming and databases goes back to (Kakas & Mancarella, 1990). Taking advantage of their framework of *extended abduction* (Inoue & Sakama, 1995), Inoue and Sakama (1999) integrated three different types of updates into a single framework, namely *view update*, *theory update*, and *inconsistency removal*. In particular, view update deals with the problem of changing *extensional facts* (which do not occur in the heads of rules), whilst theory update covers the general case in which (a set of) rules should be incorporated into a knowledge base. We discuss the latter problem here.

Informally, for ELPs P_1 and P_2, an update of P_1 by P_2 is a largest program P' such that $P_1 \subseteq P' \subseteq P_1 \cup P_2$ holds and where P' is consistent (i.e., P' has a consistent answer set). This intuition is formally captured by reducing the problem of updating P_1 with P_2 to computing a minimal set of abducible rules $Q \subseteq P_1 \setminus P_2$ such that $(P_1 \cup P_2) \setminus Q$ is consistent. In technical terms of (Inoue & Sakama, 1995), the program $P_1 \cup P_2$ is considered for abduction where the rules in $P_1 \setminus P_2$ are abducible, and the intended update is realized via a minimal *anti-explanation* for falsity, which removes abducible rules to restore consistency.

While this looks similar to our minimal updates, there is, however, a salient difference: abductive update does not respect *causal rejection*. That is, a rule r from $P_1 \setminus P_2$ may be rejected even if no rule r' in P_2 fires whose head contradicts the application of r. For example, consider $P_1 = \{q \leftarrow, \neg q \leftarrow a\}$ and $P_2 = \{a \leftarrow\}$. Both P_1 and P_2 have consistent answer sets, but (P_1, P_2) has no (consistent) answer set because no rule in P_1 is rejected and thus both rules must fire. On the other hand, in Inoue and Sakama's approach, one of the two rules in P_1 will be removed. Furthermore, inconsistency removal in a program P occurs in this framework as special case of updating (take, e.g., $P_1 = P$ and $P_2 = \emptyset$).

From a computational point of view, abductive updates are—due to inherent minimality criteria—harder than update programs; in particular, some abductive reasoning problems are shown to be Σ_2^P-complete (Inoue & Sakama, 1999).

7.5 Updates Through Priorities

Zhang and Foo (1998) described an approach for updating logic programs based on their preference-handling framework for logic programs introduced in (Zhang & Foo, 1997). The general approach is rather involved and proceeds in two stages, roughly described as follows. For updating P_1 with P_2, in the first stage, each answer set S of P_1 is updated to a "closest" answer set S' of P_2, where distance is measured in terms of the set of atoms for which S and S' have different truth values, and closeness is set inclusion. Then, a maximal set of rules $Q \subseteq P_1$ is chosen in such a way that $P_3 = P_2 \cup Q$ has an answer set containing S'. In the second stage, P_3 is viewed as a prioritized logic program in which rules from P_2 have higher priority than rules from Q, and its answer sets are computed. The resulting answer sets are identified as the answer sets of the update of P_1 with P_2.

This approach is apparently different from our update framework. In fact, it is in the spirit of Winslett's (1988) *possible models approach*, where the models of a propositional theory are updated separately and which satisfies update postulate (U8). More specifically, the two stages in Zhang and Foo's approach respectively aim at removing contradictory rules from P_1 and resolving conflicts between the remaining rules of P_2. However, like in Inoue and Sakama's approach, rules are not removed on the basis of causal rejection. In particular, on the example considered in (Zhang & Foo, 1998), both approaches yield the same result. The second stage of the procedure indicates a strong update flavor of the approach, since rules are unnecessarily abandoned. For example, the update of $P_1 = \{p \leftarrow not\, q\}$ with $P_2 = \{q \leftarrow not\, p\}$ results in P_2, even though $P_1 \cup P_2$ is consistent. Since, in general, the result of an update is given by a set of programs, naive handling of sequences of updates consumes exponential space in general.

7.6 Compiled Preferences

Since the underlying conflict-resolution strategy of many update formalisms, including the current one, is to associate, in some sense, "higher preference" to new pieces of information, as final installment of our discussion on related work, we briefly review the approach of Delgrande *et al.* (2000) to preference handling in logic programming, which is also based on a transformational principle.

To begin with, Delgrande *et al.* (2000) define an *ordered logic program* as an ELP in which rules are named by unique terms and in which preferences among rules are given by a new set of atoms of the form $s \prec t$, where s and t are names. Thus, preferences among rules are encoded at the *object-level.* An ordered logic program is transformed into a second, regular ELP wherein the preferences are respected, in the sense that the answer sets obtained in the transformed theory correspond to the preferred answer sets of the original theory. The approach is sufficiently general to allow the specification of preferences among preferences, preferences holding in a particular context, and preferences holding by default.

The encoding of ordered logic programs into standard ELPs is realized by means of dedicated atoms, which control the applicability of rules with respect to the intended order. More specifically, if rule r has preference over rule r', the control elements ensure that r is considered before r', in the sense that, for a given answer set S, rule r is known to be applied or defeated *ahead of* r'.

This control mechanism is more strict than the rejection principle realized in Definition 2. For instance, in the preference approach, it may happen that no answer set exists because the applicability of a higher-ranked rule depends on the applicability of a lower-ranked rule, effectively resulting in a circular situation which cannot be resolved in a consistent manner. On the other hand, this is not necessarily the case in the current update framework, where newer rules may only be applicable given older pieces of information. So, in order to simulate updates within the framework of (Delgrande *et al.*, 2000), under the proviso that newer information has preference over older information, it is necessary to relax the conditions which enable successive rule applications.

8 Conclusion

In this paper, we considered a formalization of an approach to sequences of logic program updates based on a causal rejection principle for rules, which is inherent to other approaches as well. We provided, in the spirit of dynamic logic programming (Alferes *et al.*, 1998; Alferes *et al.*, 2000), a definition of the semantics of sequences $\boldsymbol{P}$ of ELPs in terms of a simple transformation to update programs, $\boldsymbol{P}_{\triangleleft}$, which are ordinary ELPs, and described a declarative semantical characterization as well. Then, as a main novel contribution, we investigated the properties of this approach and of similar ones from the perspective of belief revision and nonmonotonic reasoning, based on the given characterization. For this purpose, we considered different possibilities of interpreting update programs as theory change operators and abstract nonmonotonic consequence operators, respectively. Our main findings on this aspect were that many of the postulates and principles from these areas are not satisfied by update programs. We then have introduced further properties, including an iterativity property, and evaluated them on update programs.

Motivated by an apparent lack of minimality of change, we then considered refinements of the semantics in terms of minimal and strictly minimal answer sets, and discussed their complexity and implementation. Furthermore, we compared the current proposal to other related approaches, and found that it is semantically equivalent to a fragment of inheritance logic programs as defined by Buccafurri *et al.* (1999a). Moreover, our approach is more liberal than dynamic logic programming, which has been introduced by Alferes *et al.* (1998; 2000), and can be seen to coincide with it on certain classes of programs (cf. (Eiter *et al.*, 2000b)). Our discussion on general principles of update sequences based on causal rejection applies for these formalisms as well.

Several issues remain for further work. An interesting point concerns the formulation of postulates for update operators on logic programs and, more generally, on nonmonotonic theories. As we have seen, several postulates from the area of logical theory change fail for update programs (cf. (Brewka, 2000) for related observations on this topic). This may partly be explained by the nonmonotonicity of answer sets semantics, and by the dominant role of syntax for update embodied by causal rejection of rules. However, similar features are not exceptional in the context of logic programming. Therefore, it would be interesting to consider further postulates and desiderata for updating logic programs besides the ones we analyzed here, as well as an AGM style characterization of update operators compliant with them. This issue seems to be rather demanding, though, and we might speculate—without further evidence—that it will be difficult to find a general acceptable set of postulates which go beyond "obvious" properties.

A natural issue for update logic programs is the inverse of addition, i.e., retraction of rules from a logic program. Dynamic logic programming evolved into LUPS (Alferes *et al.*, 2000), which is a language for specifying update behavior in terms of conditional addition and retraction of sets of rules to a logic program. LUPS is generic, however, as in principle different approaches to updating logic programs could provide the underlying semantical basis for the single update steps. Exploring properties of the general framework, as well as of particular instantiations, and reasoning about update programs describing the behavior of agents programmed in LUPS or in other similar languages is topic of ongoing research.

Finally, building real-life applications, like intelligent information agents whose rational component is modeled by a knowledge base, which is in turn maintained using update logic programs, is an interesting issue for further research. The integration of reasoning components into agent architectures amenable to logic programming methods, such as the one of the IMPACT agent platform (Subrahmanian *et al.*, 2000), is an important next step in order to make the techniques available to agent developers. This is also part of our current research.

Acknowledgments.

The authors would like to thank the anonymous referees and Michael Gelfond for their comments, which helped improving the paper, as well as Josè Julio Alferes, João Alexandre Leite, and Luís Moniz Pereira for their useful remarks and corrections. This work was supported by the Austrian Science Fund (FWF) under grants P13871-INF and N Z29-INF.

A Proofs

A.1 Proof of Theorem 2

For any set $U \subseteq Lit_{\mathcal{A}^*}$, define $U_0 = U \cap Lit_{\mathcal{A}}$, and, for $1 \leq i \leq n$, let $U_i = \{L_i \mid L_i \in U\}$ and $U_i^{rej} = \{rej(r) \mid rej(r) \in U, r \in P_i\}$. Clearly, it holds that $U = U_0 \cup \bigcup_{i=1}^{n}(U_i \cup U_i^{rej})$.

Consider the answer sets S, T of $\boldsymbol{P}_{\triangleleft}$ and assume that $S \cap Lit_{\mathcal{A}} = T \cap Lit_{\mathcal{A}}$. We show by induction on j $(0 \leq j \leq n-1)$ that $S_{n-j} = T_{n-j}$ and $S_{n-j}^{rej} = T_{n-j}^{rej}$. From this, and given the relation $S_0 = T_0$ (by the assumption $S \cap Lit_{\mathcal{A}} = T \cap Lit_{\mathcal{A}}$), it follows that $S = T$.

Induction Base. Assume $j = 0$. First of all, it is quite obvious that $S_n^{rej} = T_n^{rej} = \emptyset$. Consider now some $L_n \in Lit_{\mathcal{A}^*}$. According to the construction of the transformation $\boldsymbol{P}_{\triangleleft}$, the literal L_n can only be derived by some rule $L_n \leftarrow B(r), not\, rej(r) \in \boldsymbol{P}_{\triangleleft}$, where $r \in P_n$. Since $S_n^{rej} = T_n^{rej} = \emptyset$, it follows that $L_n \leftarrow B^+(r)$ must be a member of both $\boldsymbol{P}_{\triangleleft}^S$ and $\boldsymbol{P}_{\triangleleft}^T$. Since $B^+(r) \subseteq Lit_{\mathcal{A}}$ and $S_0 = T_0$, we have $B^+(r) \subseteq S$ iff $B^+(r) \subseteq T$. Thus, $L_n \in S_n$ iff $L_n \in T_n$. This implies $S_n = T_n$.

Induction Step. Assume $n - 1 \geq j > 0$, and let the assertions $S_{n-k} = T_{n-k}$ and $S_{n-k}^{rej} = T_{n-k}^{rej}$ hold for all $k < j$. We show that they hold for $k = j$ as well. Consider some atom $rej(r)$ where $r \in P_{n-j}$. Given the transformation $\boldsymbol{P}_{\triangleleft}$, the atom $rej(r)$ can only be derived by means of rule $rej(r) \leftarrow B(r), \neg L_{n-j+1} \in \boldsymbol{P}_{\triangleleft}$. Since

$B^-(r) \subseteq \mathit{Lit}_{\mathcal{A}}$ and $S_0 = T_0$, it holds that $B^-(r) \cap S = B^-(r) \cap T$. Hence, $\mathit{rej}(r) \leftarrow B^+(r), \neg L_{n-j+1}$ is in $\boldsymbol{P}^S_{\triangleleft}$ iff it is in $\boldsymbol{P}^T_{\triangleleft}$. By induction hypothesis, $\neg L_{n-j+1} \in S$ iff $\neg L_{n-j+1} \in T$. Since we also have that $B^+(r) \subseteq S$ iff $B^+(r) \subseteq T$, it follows that $\mathit{rej}(r) \in S$ iff $\mathit{rej}(r) \in T$, and so $S^{\mathit{rej}}_{n-j} = T^{\mathit{rej}}_{n-j}$.

Consider now some literal $L_{n-j} \in \mathit{Lit}_{\mathcal{A}^*}$. This literal can only be derived by means of rule $L_{n-j} \leftarrow L_{n-j+1}$, or by a rule of the form $L_{n-j} \leftarrow B(r), \mathit{not\,rej}(r)$, for some $r \in P_{n-j}$. If L_{n-j} is derived by $L_{n-j} \leftarrow L_{n-j+1}$, it follows immediately from the induction hypothesis that $L_{n-j} \in S$ iff $L_{n-j} \in T$. So assume now that the second case applies. Since we already know that $S^{\mathit{rej}}_{n-j} = T^{\mathit{rej}}_{n-j}$, and since $B^-(r) \cap S = B^-(r) \cap T$, we have that $L_{n-j} \leftarrow B^+(r)$ lies in $\boldsymbol{P}^S_{\triangleleft}$ iff it lies in $\boldsymbol{P}^T_{\triangleleft}$. Again using the property that $B^+(r) \subseteq S$ iff $B^+(r) \subseteq T$, we obtain that $L_{n-j} \in S$ iff $L_{n-j} \in T$. Combining the two cases, and since the literal L_n was arbitrarily chosen, it follows that $S_{n-j} = T_{n-j}$.

A.2 Proof of Theorem 4

Only-if part. Suppose S is an answer set of $\boldsymbol{P} = (P_1, \ldots, P_n)$. We show that S is a minimal model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. First, we show that S is a model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$.

Let $\check{S}$ be the uniquely determined answer set of $\boldsymbol{P}_{\triangleleft}$ such that $S = \check{S} \cap \mathit{Lit}_{\mathcal{A}}$. Consider some $r^+ \in (\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. We first assume that r is a constraint. So, $r \in \boldsymbol{P}_{\triangleleft}$. Since $B^-(r) \cap S = \emptyset$, $B^-(r) \subseteq \mathit{Lit}_{\mathcal{A}}$, and $S \subseteq \check{S}$, we have $B^-(r) \cap \check{S} = \emptyset$. Hence, $B^+(r) \not\subseteq \check{S}$, since $\check{S}$ is an answer set of $\boldsymbol{P}_{\triangleleft}$ and $r^+ \in (\boldsymbol{P}_{\triangleleft})^{\check{S}}$. It follows that r^+ is true in S. Let us now consider the case when r is not a constraint. Then, there is some i, $1 \leq i \leq n$, such that $r \in P_i$ and $r \notin \mathit{Rej}(S, \boldsymbol{P})$. We must show that $H(r) \in S$ whenever $B^+(r) \subseteq S$. By construction of the update program $\boldsymbol{P}_{\triangleleft}$, r induces some rule $L_i \leftarrow B(r), \mathit{not\,rej}(r) \in \boldsymbol{P}_{\triangleleft}$, where $L = H(r)$. We claim that $L_i \leftarrow B(r), \mathit{not\,rej}(r)$ is not defeated by $\check{S}$. First of all, since $B^-(r) \cap S = \emptyset$, it follows that $B^-(r) \cap \check{S} = \emptyset$, as argued above. Furthermore, since $r \notin \mathit{Rej}(S, \boldsymbol{P})$, Lemma 1 implies $\mathit{rej}(r) \notin \check{S}$. This proves the claim. Thus, $L_i \leftarrow B^+(r) \in (\boldsymbol{P}_{\triangleleft})^{\check{S}}$. Consequently, assuming $B^+(r) \subseteq S$, it holds that $L_i \in \check{S}$, since $\check{S}$ is an answer set of $\boldsymbol{P}_{\triangleleft}$ and $S \subseteq \check{S}$. Moreover, since $(\boldsymbol{P}_{\triangleleft})^{\check{S}}$ contains the inertia rules $L_i \leftarrow L_{i+1}$ $(1 \leq i < n)$ and $L \leftarrow L_1$, it follows that $L \in \check{S}$. By observing that $L \in \mathit{Lit}_{\mathcal{A}}$, $L \in S$ follows, which implies that r^+ is true in S. This concludes the proof that S is a model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. It remains to show that S is a minimal model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$.

Assume that $S_0 \subset S$ is a model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. Consider the set

$$\tilde{S}_0 \quad = \quad \check{S} \setminus (\{L \mid L \in S \setminus S_0\} \cup \{L_i \mid L \in S \setminus S_0, 1 \leq i \leq n\}).$$

It is easy to show that $\tilde{S}_0$ is a model of $(\boldsymbol{P}_{\triangleleft})^{\check{S}}$. Moreover, $\tilde{S}_0 \subset \check{S}$. We arrive at a contradiction, because $\check{S}$ is assumed to be an answer set of $\boldsymbol{P}_{\triangleleft}$. As a consequence, S must be a minimal model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. This concludes the proof that S is a minimal model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$ whenever S is an answer set of $\boldsymbol{P}$.

If part. Assume that S is a minimal model of $(\cup \boldsymbol{P} \setminus \mathit{Rej}(S, \boldsymbol{P}))^S$. Define $\tilde{S} \subseteq \mathit{Lit}_{\mathcal{A}^*}$

as follows:

$$\tilde{S} = S \cup \{rej(r) \mid r \in Rej(S, \boldsymbol{P})\} \cup \bigcup_{i=1}^{n} \{L_j \mid 1 \le j \le i, \exists r \in P_i \setminus Rej(S, \boldsymbol{P}) \text{ such that } H(r) = L \text{ and } S \models B(r)\}.$$

We show that $\tilde{S}$ is an answer set of $\boldsymbol{P}_{\lhd}$. Since $\tilde{S} \cap Lit_{\mathcal{A}} = S$, this will imply that S is an answer set of $\boldsymbol{P}$.

We first show that $\tilde{S}$ is a model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$. Consider some $r^+ \in (\boldsymbol{P}_{\lhd})^{\tilde{S}}$. Depending on the construction of $\boldsymbol{P}_{\lhd}$, there are several cases to distinguish.

(i) r is a constraint. Then, $B^+(r) \not\subseteq \tilde{S}$. Otherwise, we would have $B^+(r) \subseteq S$ and $r^+ \in (\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$ (since $S \subseteq \tilde{S}$, $B^+(r) \subseteq Lit_{\mathcal{A}}$, and $B^-(r) \cap \tilde{S} = \emptyset$), violating the condition that S is a model of $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. Thus, r^+ is true in $\tilde{S}$.

(ii) r is a rule of form $L_i \leftarrow B(r'), \mathit{not}\, rej(r')$, where $L = H(r')$. Since $r^+ \in (\boldsymbol{P}_{\lhd})^{\tilde{S}}$, r^+ is not defeated by $\tilde{S}$ and $rej(r') \notin \tilde{S}$. According to the definition of $\tilde{S}$, the latter condition implies that $r' \notin Rej(S, \boldsymbol{P})$. Since $H(r) = L_i$, it holds that $r' \in P_i$, so $r' \in P_i \setminus Rej(S, \boldsymbol{P})$. Assume $B^+(r') \subseteq \tilde{S}$. Since $S \subseteq \tilde{S}$ and $B^+(r') \subseteq Lit_{\mathcal{A}}$, we get $B^+(r') \subseteq S$. Moreover, since r is not defeated by $\tilde{S}$, the definition of $\tilde{S}$ implies that $L_i \in \tilde{S}$. This shows that r^+ is true in $\tilde{S}$.

(iii) r is a rule of form $rej(r') \leftarrow B(r'), \neg L_{i+1}$, where $r' \in P_i$ and $L = H(r')$. Assume $B^+(r) \subseteq \tilde{S}$. Hence, $\neg L_{i+1} \in \tilde{S}$. By definition of $\tilde{S}$, this implies that there is some rule $r'' \in P_j \setminus Rej(S, \boldsymbol{P})$, $i+1 \le j \le n$, such that $H(r'') = \neg L$, $B^+(r'') \subseteq S$, and r'' is not defeated by S. Since $Rej_j(S, \boldsymbol{P}) \subseteq Rej(S, \boldsymbol{P})$, it follows immediately that $r' \in Rej_i(S, \boldsymbol{P}) \subseteq Rej(S, \boldsymbol{P})$, which in turn implies $rej(r') \in \tilde{S}$, by definition of $\tilde{S}$, proving that r^+ is true in $\tilde{S}$.

(iv) r is a rule of form $L_i \leftarrow L_{i+1}$ $(1 \le i < n)$. Then r is trivially true in $\tilde{S}$, by construction of $\tilde{S}$.

(v) r is a rule of form $L \leftarrow L_1$. If $L_1 \in \tilde{S}$, then there is some $r' \in P_1 \setminus Rej(S, \boldsymbol{P})$ such that $H(r') = L$, $B^+(r') \subseteq S$, and r' is not defeated by S. Since S is a model of $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$, it follows that $L \in S \subseteq \tilde{S}$. Thus, r is true in $\tilde{S}$.

This concludes the proof that $\tilde{S}$ is a model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$. We proceed by showing that $\tilde{S}$ is a minimal model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$. Suppose $\tilde{S}_0$ is a model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$ such that $\tilde{S}_0 \subset \tilde{S}$. We show that this implies $\tilde{S} \subseteq \tilde{S}_0$, a contradiction. Hence, $\tilde{S}$ must be minimal.

Let us first assume that $\tilde{S}_0 \cap Lit_{\mathcal{A}} \subset \tilde{S} \cap Lit_{\mathcal{A}}$, i.e., $\tilde{S}_0$ is smaller on the literals in $Lit_{\mathcal{A}}$. Then, for some $L \in Lit_{\mathcal{A}}$, no rule $r^+ \in P_{\lhd}^{\tilde{S}}$ with $H(r) = L_i$ fires in $\tilde{S}_0$, i.e., $B^+(r) \not\subseteq \tilde{S}_0$. Hence, by definition of $\tilde{S}$ and $P_{\lhd}^{\tilde{S}}$, there is no $r' \in (\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$ such that $H(r') = L$ and $B^+(r') \subseteq S$. Consequently, $S \setminus \{L\}$ satisfies all rules in $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. This, however, contradicts the fact that S is a minimal model of $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$. It follows that $\tilde{S}_0 \cap Lit_{\mathcal{A}} = \tilde{S} \cap Lit_{\mathcal{A}}$ holds.

Now consider any $L_i \in \tilde{S}$. Then, there is a rule $r \in P_j \setminus Rej(S, \boldsymbol{P})$, $i \le j \le n$, such that $B^+(r) \subseteq S$ and r is not defeated by S. According to the definition of $\boldsymbol{P}_{\lhd}$, and by Lemma 1, rule r introduces a rule $A_j \leftarrow B^+(r) \in (\boldsymbol{P}_{\lhd})^{\tilde{S}}$. Since $B^+(r) \subseteq S = \tilde{S} \cap Lit_{\mathcal{A}}$ and $\tilde{S} \cap Lit_{\mathcal{A}} = \tilde{S}_0 \cap Lit_{\mathcal{A}}$, it follows that $A_j \in \tilde{S}_0$, by the assumption that $\tilde{S}_0$ is a model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$. Moreover, since $i \le j$, the inertia rules $L_k \leftarrow L_{k+1} \in (\boldsymbol{P}_{\lhd})^{\tilde{S}}$ $(1 \le k < n)$ imply $L_i \in \tilde{S}_0$.

Finally, consider $rej(r) \in \tilde{S}$, where $r \in P_i$. By the definitions of $Rej(S, \boldsymbol{P})$ and $\tilde{S}$, it follows that $B^+(r) \subseteq S$, $B^-(r) \cap S = \emptyset$, and $\neg L_{i+1} \in \tilde{S}$. From the above considerations, $\neg L_{i+1} \in \tilde{S}$ implies $\neg L_{i+1} \in \tilde{S}_0$. Moreover, $B^-(r) \cap \tilde{S} = \emptyset$. So, $rej(r) \leftarrow B^+(r), \neg L_{i+1} \in \boldsymbol{P}_{\lhd}^{\tilde{S}}$. Since $\tilde{S}_0$ is a model of $(\boldsymbol{P}_{\lhd})^{\tilde{S}}$, and given the fact that $\tilde{S}_0 \cap Lit_{\mathcal{A}} = S$, we obtain $rej(r) \in \tilde{S}_0$. This concludes the proof that $\tilde{S} \subseteq \tilde{S}_0$.

A.3 Proofs of the Revision and Update Postulates

In what follows, we assume that $\boldsymbol{P}$ is a nonempty sequence $(P_1, \ldots, P_n)$ of ELPs.

(K1) $(\boldsymbol{P}, P)$ represents a belief set.

This holds by convention.

(K2) & (U1) $P \subseteq Bel((\boldsymbol{P}, P))$.

This is clearly satisfied, as the rules of P cannot be rejected in the updated program.

(U2) $Bel(P) \subseteq Bel(\boldsymbol{P})$ implies $Bel((\boldsymbol{P}, P)) = Bel(\boldsymbol{P})$.

This postulate states that no change occurs if the update is already entailed. This means that inconsistency is preserved under updates and contradictions cannot be removed by updates. This is clearly not the case : Updating $\boldsymbol{P} = \{a \leftarrow \ , \neg a \leftarrow\}$ with $P = \{a \leftarrow\}$ removes inconsistency. Also for a consistent $\boldsymbol{P}$, update by a logically implied rule may lead to a change in semantics. Consider, e.g., $\boldsymbol{P} = \{a \leftarrow not\, b\}$ and $P = \{b \leftarrow not\, a\}$. Then $\boldsymbol{P}$ has the unique answer set $S = \{a\}$, and $S \models b \leftarrow not\, a$. However, $(\boldsymbol{P}, P)$ has, besides S, another answer set $S' = \{b\}$.

(K3) $Bel((\boldsymbol{P}, P)) \subseteq Bel(Bel(\boldsymbol{P}) \cup P)$.

This property fails in general, if programs have infinite alphabets. This can be seen by the following example. Let $\boldsymbol{P} = P_1$ and $P = P_2$, where

$$\begin{aligned} P_1 &= \{a_i \leftarrow not\, b_i,\, b_i \leftarrow not\, a_i,\, c \leftarrow a_i \mid i \geq 1\} \cup \{\ \leftarrow not\, c\}; \\ P_2 &= \{\ \leftarrow b_i \mid i \geq 1\}. \end{aligned}$$

It is easy to see that every answer set S of P_1 must contain c, and that either a_i or b_i (but not both) are contained in S. Therefore, $c \in Bel(\boldsymbol{P})$ holds. Furthermore, $S' = \{a_i \mid i \geq 1\} \cup \{c\}$ is an answer set of $Bel(\boldsymbol{P})$. Since $S' \models P_2$, it follows that S' is an answer set of $Bel(\boldsymbol{P}) \cup P$. This implies $Bel(Bel(\boldsymbol{P}) \cup P) \subset Bel(\{\leftarrow\})$, i.e., $Bel(Bel(\boldsymbol{P}) \cup P)$ does not contain all possible rules.

On the other hand, $Bel((\boldsymbol{P}, P)) = Bel(\{\leftarrow\})$: Since negation does not occur in rule heads of P_1 and P_2, we have $Rej(S, (\boldsymbol{P}, P)) = \emptyset$, and thus $\mathcal{U}((\boldsymbol{P}, P)) = \mathcal{S}(P_1 \cup P_2)$ holds. However, $P_1 \cup P_2$ clearly has no answer set, which implies $Bel((\boldsymbol{P}, P)) = Bel(P_1 \cup P_2) = Bel(\{\leftarrow\})$. It follows that $Bel((\boldsymbol{P}, P)) \not\subseteq Bel(Bel(\boldsymbol{P}) \cup P)$, which proves our claim.

That property (K3) holds if either $\boldsymbol{P}$ or P has a finite alphabet follows from (K7), which subsumes (K3) by choosing $P = \emptyset$ in (K7), and by virtue of $Bel((\boldsymbol{P}, \emptyset)) = Bel(\boldsymbol{P})$.

(U3) If both $\boldsymbol{P}$ and P are satisfiable, then $(\boldsymbol{P}, P)$ is satisfiable.

This is clearly violated. Consider, e.g., $\boldsymbol{P} = P_1$ and $P = P_2$, where

$$P_1 = \{a \leftarrow b, not\, a\};$$
$$P_2 = \{b \leftarrow \ \}.$$

(K4) If $Bel(\boldsymbol{P}) \cup P$ has an answer set, then $Bel(Bel(\boldsymbol{P}) \cup P) \subseteq Bel((\boldsymbol{P}, P))$.

The property is violated. Consider $P_1 = \{a \leftarrow\ ,\ b \leftarrow not\, c,\ c \leftarrow not\, b\}$ and $P_2 = \{\neg a \leftarrow b\}$. As easily seen, the sequence (P_1, P_2) has two answer sets, $S = \{b, \neg a\}$ and $S' = \{a, c\}$. On the other hand, since $P_1 \subseteq Bel(P_1)$, S cannot be an answer set of $Bel(P_1) \cup P_2$; in fact, S' is its unique answer set. Since, e.g., $S' \models c \leftarrow not\, a, b$ whilst $S \not\models c \leftarrow not\, a, b$, it follows that $Bel(Bel(P_1) \cup P_2) \not\subseteq Bel((P_1, P_2))$.

(K5) $(\boldsymbol{P}, P)$ is unsatisfiable iff P is unsatisfiable.

This is violated, since contradictory rules in $\boldsymbol{P}$ are not affected unless they are rejected by rules in P. For instance, if $\boldsymbol{P}$ consists of the single program $\{a \leftarrow\ ,\ \neg a \leftarrow\ \}$, then the update of $\boldsymbol{P}$ by $P = \{b \leftarrow\ \}$ does not have an answer set.

(K6) & (U4) $\boldsymbol{P} \equiv \boldsymbol{P'}$ and $P \equiv P'$ implies $(\boldsymbol{P}, P) \equiv (\boldsymbol{P'}, P')$.

This expresses *irrelevance of syntax* which is clearly not satisfied, since rejection of rules depends on their syntactical form. For instances, take $\boldsymbol{P} = \boldsymbol{P'} = P_1$, $P = P_2$, and $P' = P_2'$, where

$$P_1 = \{a \leftarrow\ , b \leftarrow\ \};$$
$$P_2 = \{\neg a \leftarrow b\};$$
$$P_2' = \{\neg b \leftarrow a\}.$$

Then clearly $\boldsymbol{P} \equiv \boldsymbol{P'}$ and $P_2 \equiv P_2'$, but the resulting updates have different answer sets: $\{\neg a, b\}$ is an answer set of (P_1, P_2) but not of (P_1, P_2').

(K7) & (U5) $Bel((\boldsymbol{P}, P \cup P')) \subseteq Bel(Bel((\boldsymbol{P}, P)) \cup P')$.

The property does not hold if both $\boldsymbol{P}$ and P' (or P and P') have infinite alphabets, which follows from the example showing the failure of (K3) (set $P = \emptyset$, and exploit the relation $Bel((\boldsymbol{P}, \emptyset)) = Bel(\boldsymbol{P})$).

Property (K7) holds if $(\cup\boldsymbol{P}) \cup P$ or P' has a finite alphabet. Towards a contradiction, suppose it fails. Then, there exists $r \in Bel((\boldsymbol{P}, P \cup P')) \setminus Bel(Bel((\boldsymbol{P}, P)) \cup P')$, and hence an answer set $S \in \mathcal{S}(Bel((\boldsymbol{P}, P)) \cup P')$ such that $S \not\models r$.

Consider $\boldsymbol{P'} = (\boldsymbol{P}, P)$, and let $\mathcal{A}'$ denote the atoms in $\boldsymbol{P'}$. Then, for every finite set of atoms $\mathcal{A}_0 \subseteq \mathcal{A}'$, there must exist some answer set $S_{\mathcal{A}_0}$ of $\boldsymbol{P'}$ such that S and $S_{\mathcal{A}_0}$ coincide with respect to $\mathcal{A}_0$. Indeed, $Bel(\boldsymbol{P'})$ must contain, for each interpretation M which does not coincide with any answer set of $\boldsymbol{P'}$ with respect to $\mathcal{A}_0$, the constraint $\leftarrow L_1, \ldots, L_m, not\, L_{m+1}, \ldots, not\, L_n$, where $\{L_1, \ldots, L_m\} = Lit_{\mathcal{A}_0} \cap M$ and $\{L_{m+1}, \ldots, L_n\} = Lit_{\mathcal{A}_0} \setminus M$, respectively. Furthermore, all answer sets of $Bel(\boldsymbol{P'}) \cup P'$ must coincide on the atoms in $\mathcal{A} \setminus \mathcal{A}'$. Thus, assuming that either $\boldsymbol{P'}$ or P' has a finite alphabet, it follows that S is an answer set of $\boldsymbol{P'}$ or P'. Without loss of generality, we assume that S is an answer set of $\boldsymbol{P'}$.

Hence, Theorem 4 implies that S is a minimal model of $((\cup\boldsymbol{P'}) \setminus Rej(S,\boldsymbol{P'}))^S$. Since $S \models P'$, we conclude that S is also a minimal model of $(((\cup\boldsymbol{P'}) \cup P') \setminus Rej(S,\boldsymbol{P}))^S$. Furthermore, for the update sequence $\boldsymbol{P''} = (\boldsymbol{P}, P \cup P')$, it holds that $Rej(S,\boldsymbol{P''}) = Rej(S,\boldsymbol{P'})$. Indeed, $S \models r'$ for all $r' \in P \cup P'$, thus $r' \notin Rej(S,\boldsymbol{P'})$ and $r \notin Rej(S,\boldsymbol{P''})$. Equivalence for the rules in $\boldsymbol{P}$ can be shown by induction on the length of $\boldsymbol{P}$. Hence, we obtain that S is a minimal model of $((\cup\boldsymbol{P''}) \setminus Rej(S,\boldsymbol{P''}))^S$. From Theorem 4, we obtain that S is an answer set of $\boldsymbol{P''}$. Since $S \not\models r$, it follows $r \notin Bel(\boldsymbol{P}, P \cup P')$, a contradiction.

(U6) Given $Bel(P') \subseteq Bel((\boldsymbol{P}, P))$ and $Bel(P) \subseteq Bel((\boldsymbol{P}, P'))$, then $Bel((\boldsymbol{P}, P)) = Bel((\boldsymbol{P}, P'))$.

This postulate fails. Consider, e.g., $\boldsymbol{P} = P_1$, $P = P_2$, and $P' = P_3$, where

$$\begin{aligned} P_1 &= \{\, b \leftarrow\ , d \leftarrow \,\}; \\ P_2 &= \{\, \neg a \leftarrow\ , \neg e \leftarrow d, \neg d \leftarrow e \,\}; \\ P_3 &= \{\, \neg a \leftarrow\ , \neg c \leftarrow b, \neg b \leftarrow b \,\}. \end{aligned}$$

Then, $\{\neg a, b, d, \neg e\}$ is the unique answer set of (P_1, P_2), and $\{\neg a, b, d, \neg c\}$ is the unique answer set of (P_1, P_3). Moreover, it is easily verified that $\neg a \in S \cap S'$, for any answer set S of P_2 and any answer set S' of P_3. Hence, $Bel(P_3) \subseteq Bel((P_1, P_2))$ and $Bel(P_2) \subseteq Bel((P_1, P_3))$. However, $Bel((P_1, P_2)) \neq Bel((P_1, P_3))$.

(K8) If $Bel((\boldsymbol{P}, P)) \cup P'$ is satisfiable, then $Bel(Bel(\boldsymbol{P}, P) \cup P') \subseteq Bel((\boldsymbol{P}, P \cup P'))$.

This postulate fails. Setting $P = \emptyset$, the property reduces to (K4) since $Bel(\boldsymbol{P}, \emptyset) = Bel(\boldsymbol{P})$. The failure follows from the failure of (K4).

A.4 Proofs of the Postulates for Iterated Revision

(C1) If $P' \subseteq Bel(P)$, then $Bel((\boldsymbol{P}, P', P)) = Bel((\boldsymbol{P}, P))$.

Adding rules which are implied after the previous update does not change the epistemic state. This is not satisfied: take, e.g., $\boldsymbol{P} = \emptyset$, $P = \{b \leftarrow \mathit{not}\, a\}$, and $P' = \{a \leftarrow \mathit{not}\, b\}$. Then $(\boldsymbol{P}, P', P)$ has two answer sets, while $(\boldsymbol{P}, P)$ has a single answer set. The associated belief sets are thus different.

(C2) If $S \not\models P'$, for all $S \in \mathcal{S}(P)$, then $Bel((\boldsymbol{P}, P, P')) = Bel((\boldsymbol{P}, P'))$.

This property is not satisfied. For a counterexample, consider $P_1 = \{a \leftarrow b\}$, $P_2 = \{b \leftarrow\ \}$, and $P_3 = \{\neg b \leftarrow \mathit{not}\, a\}$. Then (P_1, P_2, P_3) has two answer sets, $\{a, b\}$ and $\{\neg b\}$, whilst (P_1, P_3) possesses the single answer set $\{\neg b\}$.

(C3) If $P' \subseteq Bel((\boldsymbol{P}, P))$, then $P' \subseteq Bel((\boldsymbol{P}, P', P))$.

Implied rules can be added before the update. This property fails in general. For example, let $\boldsymbol{P} = P_1$, $P = P_2$, and $P' = P_3$, where

$$\begin{aligned} P_1 &= \emptyset; \\ P_2 &= \{\, a \leftarrow \mathit{not}\, b,\ b \leftarrow \mathit{not}\, a,\ g \leftarrow a,\ g \leftarrow \mathit{not}\, g,\ c \leftarrow \,\}; \\ P_3 &= \{\, g \leftarrow,\ \neg c \leftarrow \mathit{not}\, a \,\}. \end{aligned}$$

Note that P_2 has a single answer set, $S = \{a, g, c\}$, and clearly $S \models P_3$. However, (P_1, P_3, P_2) has among its answer sets $S' = \{b, g, c\}$, and $S' \not\models \neg c \leftarrow not\, a$.

The property holds, however, providing P' contains a single rule. Suppose $P' \subseteq Bel((\boldsymbol{P}, P))$ but $r \notin Bel((\boldsymbol{P}, P', P))$, for $P' = \{r\}$. Then, $r \in Rej(S, (\boldsymbol{P}, P', P))$ for some answer set S of $(\boldsymbol{P}, P', P)$. This means, however, that S is an answer set of $(\boldsymbol{P}, P)$ (as r cannot reject any rule in $\boldsymbol{P}$). Thus, $r \notin Bel((\boldsymbol{P}, P))$.

(C4) If $S \models P'$ for some $S \in \mathcal{U}((\boldsymbol{P}, P))$, then $S \models P'$ for some $S \in \mathcal{U}((\boldsymbol{P}, P', P))$.

This property holds. By hypothesis, there exists some $S \in \mathcal{U}((\boldsymbol{P}, P))$ such that $S \models P'$. By Theorem 4, S is a minimal model of

$$\big(((\cup\boldsymbol{P}) \cup P) \setminus Rej(S, (\boldsymbol{P}, P))\big)^S.$$

Since $S \models P'$ and $S \models P$ (due to $S \in \mathcal{U}((\boldsymbol{P}, P))$), no rule $r' \in P'$ can be rejected by a rule r of P. Also, r' can reject a rule r'' in P only if r'' is rejected within P. Thus, $Rej(S, (\boldsymbol{P}, P)) = Rej(S, (\boldsymbol{P}, P', P))$, and S is a minimal model of

$$\big(((\cup\boldsymbol{P}) \cup P' \cup P) \setminus Rej(S, (\boldsymbol{P}, P', P))\big)^S.$$

This means, by Theorem 4, that S is an answer set of $(\boldsymbol{P}, P', P)$.

(C5) If $S \not\models P'$ for all $S \in \mathcal{U}((\boldsymbol{P}, P))$ and $P \not\subseteq Bel((\boldsymbol{P}, P'))$, then $P \not\subseteq Bel((\boldsymbol{P}, P, P'))$.

This property fails: just consider $\boldsymbol{P} = \emptyset$, $P = \{a \leftarrow\ \}$, and $P' = \{b \leftarrow\ \}$.

(C6) If $S \not\models P'$ for all $S \in \mathcal{U}((\boldsymbol{P}, P))$ and $S \not\models P$ for all $S \in \mathcal{U}((\boldsymbol{P}, P'))$, then $S \not\models P$ for all $S \in \mathcal{U}((\boldsymbol{P}, P, P'))$.

This property fails as well, which can be seen by the counterexample for (C5), setting $\boldsymbol{P} = \emptyset$. Another counterexample for (C6)—which does not exploit minimization of answer sets—is $\boldsymbol{P} = \{\neg b \leftarrow\ ,\ \neg a \leftarrow b\}$, $P = \{a \leftarrow\ \}$, and $P' = \{b \leftarrow\ \}$.

(I1) $Bel(\boldsymbol{P})$ is a consistent belief set.

This is clearly violated in general.

(I2) $P \subseteq Bel((\boldsymbol{P}, P))$.

The postulate is easily seen to be satisfied (cf. (K2) and (U1)).

(I3) If $L_0 \leftarrow\ \in Bel((\boldsymbol{P}, \{L_1 \leftarrow\ , \ldots, L_k \leftarrow\ \}))$, then $L_0 \leftarrow L_1, \ldots, L_k \in Bel(\boldsymbol{P})$.

This property holds. Suppose there is some $S \in \mathcal{U}(\boldsymbol{P})$ such that $\{L_1, \ldots, L_k\} \subseteq S$ but $L_0 \notin S$. Let $\boldsymbol{P'} = (\boldsymbol{P}, \{L_1 \leftarrow\ , \ldots, L_k \leftarrow\ \})$. Then, the following holds: For every rule r in $\boldsymbol{P'}$, $r \in Rej(S, \boldsymbol{P'})$ iff $r \in Rej(S, \boldsymbol{P})$. Indeed, each L_i $(1 \leq i \leq n)$ is neither in $Rej(S, \boldsymbol{P'})$ nor in $Rej(S, \boldsymbol{P})$. By Theorem 4, S is a minimal model of $((\cup\boldsymbol{P}) \setminus Rej(S, \boldsymbol{P}))^S$. It follows that S is a minimal model of $((\cup\boldsymbol{P'}) \setminus Rej(S, \boldsymbol{P'}))^S$, which in turn implies, by using Theorem 4 again, that $S \in \mathcal{U}(\boldsymbol{P})$. Since $L_0 \notin S$, we obtain $L_0 \leftarrow\ \notin Bel(\boldsymbol{P'})$.

(I4) If $Q_1 \subseteq Bel(\boldsymbol{P})$, then $Bel((\boldsymbol{P}, Q_1, Q_2, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_2, \ldots, Q_n))$.

This property fails. Consider $\boldsymbol{P} = \{a \leftarrow not\, b\}$ and $Q_1 = \{b \leftarrow not\, a\}$ for $n = 1$.

(I5) If $Bel(Q_2) \subseteq Bel(Q_1)$, then
$Bel((\boldsymbol{P}, Q_1, Q_2, Q_3, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_2, Q_3, \ldots, Q_n))$.

This property fails, because it generalizes (C1), which fails.

(I6) If $S \models Q_2$ for some $S \in \mathcal{U}((\boldsymbol{P}, Q_1))$, then
$Bel((\boldsymbol{P}, Q_1, Q_2, Q_3, \ldots, Q_n)) = Bel((\boldsymbol{P}, Q_1, Q_1 \cup Q_2, Q_3, \ldots, Q_n))$.

The property fails: Let $\boldsymbol{P} = \{a \leftarrow not\, b,\ b \leftarrow not\, a\}$, $Q_1 = \{c \leftarrow\ \}$, and $Q_2 = \{\neg c \leftarrow a\}$. Then, $S = \{c, b\}$ is an answer set of $(\boldsymbol{P}, Q_1)$ such that $S \models Q_2$. However, $(\boldsymbol{P}, Q_1, Q_2)$ has two answer sets, $S_1 = \{a, \neg c\}$ and $S_2 = \{c, b\}$, whilst $(\boldsymbol{P}, Q_1, Q_1 \cup Q_2)$ has the single answer set $\{c, b\}$.

A.5 Proofs of the Postulates of Updates as Nonmonotonic Consequence Relations

(N1) $P_1 \in Bel((\boldsymbol{P}, P_1))$.

This is clearly satisfied (cf. (K2), (U1), and (I2)).

(N2) If $\bigcup_{i=1}^{m} Q_i \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup \bigcup_{i=1}^{m} Q_1))$, then
$P_2 \subseteq Bel((\boldsymbol{P}, P_1))$.

The property holds. Let $Q = \bigcup_{i=1}^{m} Q_i$ and $\boldsymbol{P'} = (\boldsymbol{P}, P_1)$. Assume $Q \subseteq Bel(\boldsymbol{P'})$ and $P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup Q))$, and consider some answer set S of $\boldsymbol{P'}$. Then, $S \models Q$. Moreover, $\check{S}$ is an answer set of $\boldsymbol{P'}_{\triangleleft} \cup Q$. Since $Q \subseteq Bel(\boldsymbol{P'})$, it follows that for each rule $s \in Q$ rejecting a rule r from $\boldsymbol{P}$, there exists a rule $r_1 \in P_1$ also rejecting r. Hence, no further rule in $\boldsymbol{P}$ can be rejected using Q. Let $\boldsymbol{P''} = (\boldsymbol{P}, P_1 \cup Q)$. Then, $\check{S}$ is an answer set of $\boldsymbol{P''}_{\triangleleft}$, so S is an answer set of $(\boldsymbol{P}, P_1 \cup Q)$. Since $P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup Q))$, we obtain $S \models P_2$. This proves the property.

(N3) If $\bigcup_{i=1}^{m} Q_i \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_2 \subseteq Bel((\boldsymbol{P}, P_1))$, then
$P_2 \subseteq Bel((\boldsymbol{P}, P_1 \cup \bigcup_{i=1}^{m} Q_1))$.

The property fails: Consider the counterexample $\boldsymbol{P} = \emptyset$, $P_1 = \{a \leftarrow not\, b\}$, $P_2 = \{a \leftarrow\ \}$, and, for $m = 1$, $Q_1 = \{b \leftarrow not\, a\}$.

(N4) If $P_{i+1} \subseteq Bel((\boldsymbol{P}, P_i))$ $(1 \leq i < n)$ and $P_1 \subseteq Bel((\boldsymbol{P}, P_n))$ $(n \geq 2)$, then
$\{P' \mid P' \subseteq Bel((\boldsymbol{P}, P_i))\} = \{P' \mid P' \subseteq Bel((\boldsymbol{P}, P_j))\}$, for all $i, j \leq n$.

The property does not hold, because it includes (U6) as a special case, which fails.

(P1) If $P_1 \equiv P_2$ and $P_3 \subseteq Bel((\boldsymbol{P}, P_1))$, then $P_3 \subseteq Bel((\boldsymbol{P}, P_2))$.

The property fails, due to the following counterexample: $\boldsymbol{P} = \{a \leftarrow\ , b \leftarrow\ \}$, $P_1 = \{\neg a \leftarrow b\}$, $P_2 = \{\neg b \leftarrow a\}$, and $P_3 = \{b \leftarrow\ \}$.

(P2) If $P_1 \models P_2$ and $P_1 \subseteq Bel((\boldsymbol{P}, P_3))$, then $P_2 \subseteq Bel((\boldsymbol{P}, P_3))$.

This property does not hold. For a counterexample, consider $\boldsymbol{P} = \emptyset$, $P_1 = \{a \leftarrow not\, b\}$, $P_2 = \{a \leftarrow\ \}$, and $P_3 = \{b \leftarrow\ , \neg a \leftarrow\ \}$.

(P4) If $P_2 \subseteq Bel((\boldsymbol{P}, P_1))$ and $P_3 \subseteq Bel((\boldsymbol{P}, P_1))$, then $P_2 \cup P_3 \subseteq Bel((\boldsymbol{P}, P_1))$.

The property is trivially satisfied.

References

Alchourrón, C.E., Gärdenfors, P., & Makinson, D. (1985). On the Logic of Theory Change: Partial Meet Functions for Contraction and Revision. *Journal of Symbolic Logic*, **50**, 510–530.

Alferes, J.J., & Pereira, L.M. (1997). Update-programs can Update Programs. *Pages 110–131 of:* Dix, J., Pereira, L.M., & Przymusinski, T.C. (eds), *Selected Papers from the ICLP'96 Workshop on Nonmonotonic Extensions of Logic Programming* (*NMELP'96*). LNAI, vol. 1216. Springer.

Alferes, J.J., & Pereira, L.M. (2000). Updates plus Preferences. *Pages 345–360 of:* Aciego, M.O., de Guzmán, I.P., Brewka, G., & Pereira, L.M. (eds), *Proc. Seventh Europ. Workshop on Logic in Artificial Intelligence* (*JELIA 2000*). LNAI, vol. 1919. Springer.

Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., & Przymusinski, T.C. (1998). Dynamic Logic Programming. *Pages 98–111 of:* Cohn, A., Schubert, L., & Shapiro, C. (eds), *Proc. Sixth Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'98*). Morgan Kaufmann.

Alferes, J.J., Pereira, L.M., Przymusinska, H., & Przymusinski, T.C. (1999). LUPS - A Language for Updating Logic Programs. *Pages 162–176 of:* Gelfond, M., Leone, N., & Pfeifer, G. (eds), *Proc. Fifth Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*). LNAI, vol. 1730. Springer.

Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., & Przymusinski, T. C. (2000). Dynamic Updates of Non-Monotonic Knowledge Bases. *Journal of Logic Programming*, **45**(1–3), 43–70.

Baral, C., & Gelfond, M. (1994). Logic Programming and Knowledge Representation. *Journal of Logic Programming*, **19/20**, 73–148.

Benferhat, S., Cayrol, C., Dubois, D., Lang, J., & Prade, H. (1993). Inconsistency Management and Prioritized Syntax-Based Entailment. *Pages 640–645 of:* Bajcsy, R. (ed), *Proc. Thirteenth Int. Joint Conf. on Artificial Intelligence* (*IJCAI'93*). Morgan Kaufmann.

Brewka, G. (1991a). Cumulative Default Logic: In Defense of Nonmonotonic Inference Rules. *Artificial Intelligence*, **50**(2), 183–205.

Brewka, G. (1991b). *Nonmonotonic Reasoning: Logical Foundations of Commonsense.* Cambridge Tracts in Theoretical Computer Science, vol. 12. Cambridge Univ. Press.

Brewka, G. (2000). Declarative Representation of Revision Strategies. *Pages 18–22 of:* Horn, W. (ed), *Proc. Fourteenth Europ. Conf. on Artificial Intelligence* (*ECAI 2000*). IOS Press. Preliminary version presented at Intl. Workshop on Nonmonotonic Reasoning, Breckenridge, 2000. URL: http://arXiv.org/abs/cs.AI/0003036.

Buccafurri, F., Leone, N., & Rullo, P. (1996). Stable Models and their Computation for Logic Programming with Inheritance and True Negation. *Journal of Logic Programming*, **27**(1), 5–43.

Buccafurri, F., Faber, W., & Leone, N. (1999a). Disjunctive Logic Programs with Inheritance. *Pages 79–93 of:* De Schreye, D. (ed), *Proc. Sixteenth Int. Conf. on Logic Programming* (*ICLP'99*). MIT Press.

Cholewiński, P., Marek, V.W., & Truszczyński, M. (1996). Default Reasoning System DeReS. *Pages 518–528 of:* Carlucci Aiello, L., Doyle, J., & Shapiro, S. (eds) *Proc. Fifth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR '96).* Morgan Kaufmann.

Dantsin, E., Eiter, T., Gottlob, G., & Voronkov, A. (1997). Complexity and Expressive Power of Logic Programming. *Pages 82–101 of: Proc. Twelfth IEEE Int. Conf. on Computational Complexity* (*CCC'97*). IEEE Computer Society Press.

Darwiche, A., & Pearl, J. (1997). On the Logic of Iterated Belief Revision. *Artificial Intelligence*, **89**(1–2), 1–29.

Delgrande, J., Schaub, T., & Tompits, H. (2000). Logic Programs with Compiled Preference. *Pages 392–398 of:* Horn, W. (ed), *Proc. Fourteenth Europ. Conf. on Artificial Intelligence* (*ECAI 2000*). IOS Press.

Eiter, T., & Gottlob, G. (1992). On the Complexity of Propositional Knowledge Base Revision, Updates and Counterfactuals. *Artificial Intelligence*, **57**(2–3), 227–270.

Eiter, T., & Gottlob, G. (1995). On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence*, **15**(3–4), 289–323.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., & Scarecello, F. (1997a). A Deductive System for Nonmonotonic Reasoning. *Pages 363–374 of:* Dix, J., Furbach, U., & Nerode, A. (eds), *Proc. Fourth Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'97*). LNAI, vol. 1265. Springer.

Eiter, T., Gottlob, G., & Leone, N. (1997b). Abduction From Logic Programs: Semantics and Complexity. *Theoretical Computer Science*, **189**(1–2), 129–177.

Eiter, T., Gottlob, G., & Mannila, H. (1997c). Disjunctive Datalog. *ACM Transactions on Database Systems*, **22**(3), 364–418.

Eiter, T., Fink, M., Sabbatini, G., & Tompits, H. (2000). Considerations on Updates of Logic Programs. *Pages 2–20 of:* Aciego, M.O., de Guzmàn, I.P., Brewka, G., & Pereira, L.M. (eds), *Proc. Seventh Europ. Workshop on Logic in Artificial Intelligence* (*JELIA 2000*). LNAI, vol. 1919. Springer.

Eiter, T., Fink, M., Sabbatini, G., & Tompits, H. (2000). On Updates of Logic Programs: Semantics and Properties. Technical Report INFSYS RR-1843-00-08, Technical Univ. of Vienna, Knowledge-Based Systems Group.

Eiter, T., Leone, N., Mateis, C., Pfeifer, G., & Scarcello, F. (1998). The KR System `dlv`: Progress Report, Comparisons and Benchmarks. *Pages 406–417 of:* Cohn, A., Schubert, L., & Shapiro, S. (eds), *Proc. Sixth Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98).* Morgan Kaufmann.

Gabbay, D.M. (1985). Theoretical Foundations for Non-Monotonic Reasoning in Expert Systems. *Pages 439–457 of:* Apt, K. R. (ed), *Logics and Models of Concurrent Systems.* Springer.

Gärdenfors, P., & Makinson, D. (1994). Nonmonotonic Inferences Based on Expectations. *Artificial Intelligence*, **65**(2), 197–245.

Gärdenfors, P., & Rott, H. (1995). Belief Revision. *Pages 35–132 of:* Gabbay, D.M., Hogger, C.J., & Robinson, J.A. (eds), *Handbook of Logic in AI and Logic Programming, Volume IV.* Oxford Science Publications.

Gelfond, M., & Lifschitz, V. (1988). The Stable Model Semantics for Logic Programming. *Pages 1070–1080 of:* Kowalski, R., & Bowen, K.A. (eds), *Proc. Fifth Int. Conf. and Symposium on Logic Programming.* MIT Press.

Gelfond, M., & Lifschitz, V. (1991). Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, **9**(3–4), 365–385.

Inoue, K. (2000). A Simple Characterization of Extended Abduction. *Pages 718–732 of:* Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.K., Palmidessi, C., Pereira, L.M., Sagiv, Y., & Stuckey, P.J. (eds), *Proc. First Int. Conf. on Computational Logic* (*CL 2000*). LNAI, vol. 1861. Springer.

Inoue, K., & Sakama, C. (1995). Abductive Framework for Nonomonotonic Theory Change. *Pages 204–210 of: Proc. Fourteenth Int. Joint Conf. on Artificial Intelligence* (*IJCAI'95*). Morgan Kaufmann.

Inoue, K., & Sakama, C. (1999). Updating Extended Logic Programs through Abduction. *Pages 147–161 of:* Gelfond, M., Leone, N., & Pfeifer, G. (eds), *Proc. Fifth Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'99*). LNAI, vol. 1730. Springer.

Johnson, D.S. (1990). A Catalog of Complexity Classes. *Pages 67–161 of:* Van Leeuwen, J. (ed), *Handbook of Theoretical Computer Science*, vol. A. Elsevier.

Kakas, A.C., & Mancarella, P. (1990). Database Updates Through Abduction. *Pages 650–661 of:* McLeod, D., Sacks-Davis, R., & Schek, H.J. (eds), *Proc. Sixteenth Conf. on Very Large Databases* (*VLDB'90*). Morgan Kaufmann.

Katsuno, H., & Mendelzon, A.O. (1991). On the Difference between Updating a Knowledge Database and Revising It. *Pages 387–394 of:* Allen, J.F., Fikes, R., & Sandewall, E. (eds), *Proc. Second Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'91*). Morgan Kaufmann.

Kowalski, R.A., & Toni, F. (1996). Abstract Argumentation. *Artificial Intelligence and Law*, **4**(3-4), 275–296.

Kraus, S., Lehmann, D., & Magidor, M. (1990). Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, **44**(1-2), 167–207.

Laenens, E., Saccà, D., & Vermeir, D. (1990). Extending Logic Programming. *Pages 184–193 of:* Garcia-Molina, H., & Jagadish, H.V. (eds) *Proc. Nineteenth ACM SIGMOD Conf. on the Management of Data*.

Lehmann, D. (1995). Belief Revision, Revised. *Pages 1534–1540 of: Proc. Fourteenth Int. Joint Conf. on Artificial Intelligence* (*IJCAI'95*). Morgan Kaufmann.

Lehmann, D., & Magidor, M. (1992). What Does a Conditional Knowledge Base Entail? *Artificial Intelligence*, **55**, 1–60.

Leite, J.A. (1997). Logic Program Updates. M.Sc. Thesis, Universidade Nova de Lisboa.

Leite, J.A., Alferes, J.J., & Pereira, L.M. (2000). Multi-Dimensional Dynamic Logic Programming. *Pages 17–26 of:* Satoh, K., & Sadri, F. (ed) *Proc. CL-2000 Workshop on Computational Logic in Multi-Agent Systems* (*CLIMA 2000*).

Leite, J.A., & Pereira, L.M. (1997). Generalizing Updates: From Models to Programs. *Pages 224–246 of:* Dix, J., Pereira, L.M., & Przymusinski, T.C. (eds), *Logic Programming and Knowledge Representation, Selected Papers of the Third Int. Workshop* (*LPKR'97*). LNAI, vol. 1471. Springer.

Lifschitz, V., & Turner, H. (1994). Splitting a Logic Program. *Pages 23–37 of:* Van Hentenryck, P. (ed) *Proc. Eleventh Int. Conf. on Logic Programming* (*ICLP'94*). MIT Press.

Makinson, D. (1993). General Patterns in Nonmonotonic Reasoning. *Pages 35–110 of:* Gabbay, D.M., Hogger, C.J., & Robinson, J.A. (eds), *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume III.* Oxford Science Publications.

Makinson, D., & Gärdenfors, P. (1991). Relations Between the Logic of Theory Change and Nonmonotonic Logic. *Pages 185–205 of:* Fuhrmann, A., & Morreau, M. (eds), *The Logic of Theory Change*. Springer.

Marek, V.W., & Truszczyński, M. (1994). Revision Specifications by Means of Programs. *Pages 122–136 of:* MacNish, C., Pearce, D., & Pereira, L.M. (eds), *Proc. Europ. Workshop on Logics in Artificial Intelligence* (*JELIA'94*). LNAI, vol. 838. Springer.

Marek, V.W., & Truszczyński, M. (1991). Autoepistemic Logic. *Journal of the ACM*, **38**(3), 588–619.

Nebel, B. (1991). Belief Revision and Default Reasoning: Syntax-Based Approaches. *Pages 417–428 of:* Allen, J.F., Fikes, R., & Sandewall, E. (eds) *Proc. Second Int. Conf. on Principles of Knowledge Representation and Reasoning* (*KR'91*). Morgan Kaufmann.

Niemelä, I., & Simons, P. (1996). Efficient Implementation of the Well-founded and Stable Model Semantics. *Pages 289–303 of:* Maher, M.J. (ed) *Proc. Thirteenth Joint Int. Conf. and Symposium on Logic Programming.* MIT Press.

Papadimitriou, C.H. (1994). *Computational Complexity.* Addison Wesley.

Rao, P., Sagonas, K.F., Swift, T., Warren, D.S., & Freire, J. (1997). XSB: A System for Efficiently Computing Well-Founded Semantics. *Pages 431–441 of:* Dix, J., Furbach, U., & Nerode, A. (eds) *Proc. Fourth Int. Conf. on Logic Programming and Non-monotonic Reasoning (LPNMR'97).* LNAI, vol. 1265. Springer.

Sadri, F., & Toni, F. (2000). Computational Logic and Multi-Agent Systems: a Roadmap. *Computational Logic, Special Issue on the Future Technological Roadmap of Compulog-Net.* Pages 1–31. URL: http://www.compulog.org/net/Forum/Supportdocs.html.

Schaub, T. (1991). On Commitment and Cumulativity in Default Logics. *Pages 305–309 of:* Kruse, R., & Siegel, P. (eds), *Symbolic and Quantitative Approaches for Uncertainty: Proc. of the Europ. Conf. ECSQAU.* Springer.

Schlipf, J.S. (1995). Complexity and Undecidability Results in Logic Programming. *Annals of Mathematics and Artificial Intelligence*, **15**(3–4), 257–288.

Subrahmanian, V.S., Bonatti, P., Dix, J., Eiter, T., Kraus, S., Ozcan, F., & Ross, R. (2000). *Heterogeneous Agent Systems: Theory and Implementation.* MIT Press.

Van Gelder, A., Ross, K.A., & Schlipf, J.S. (1991). The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, **38**(3), 620–650.

Winslett, M. (1988). Reasoning about Action Using a Possible Models Approach. *Pages 89–93 of: Proc. Seventh Nat. Conf. on Artificial Intelligence* (*AAAI'88*). AAAI Press/MIT Press.

Zhang, Y., & Foo, N.Y. (1997). Towards Generalized Rule-based Updates. *Pages 82–88 of: Proc. Fifteenth Int. Joint Conf. on Artificial Intelligence* (*IJCAI'97*), vol. 1. Morgan Kaufmann.

Zhang, Y., & Foo, N.Y. (1998). Updating Logic Programs. *Pages 403–407 of:* Prade, H. (ed), *Proc. Thirteenth Europ. Conf. on Artificial Intelligence* (*ECAI'98*). Wiley.