TLP **7** (4): 451–479, 2007. © 2007 Cambridge University Press
doi:10.1017/S1471068406003000 Printed in the United Kingdom

Updates in answer set programming: An approach based on basic structural properties

MAURICIO OSORIO and VÍCTOR CUEVAS
Universidad de las Américas Puebla, Sta. Catarina Mártir, Cholula, Puebla, 72820, México
(*e-mail:* {osoriomauri, victorcuevasv}@gmail.com)

submitted 14 February 2005; revised 28 October 2005, 11 August 2006; accepted 20 September 2006

Abstract

We have studied the update operator $\oplus_1$ defined for update sequences by Eiter *et al.* without tautologies and we have observed that it satisfies an interesting property[1]. This property, which we call Weak Independence of Syntax (WIS), is similar to one of the postulates proposed by Alchourrón, Gärdenfors, and Makinson (AGM); only that in this case it applies to nonmonotonic logic. In addition, we consider other five additional basic properties about update programs and we show that $\oplus_1$ satisfies them. This work continues the analysis of the AGM postulates with respect to the $\oplus_1$ operator under a refined view that considers $\mathbf{N_2}$ as a monotonic logic which allows us to expand our understanding of answer sets. Moreover, $\mathbf{N_2}$ helped us to derive an alternative definition of $\oplus_1$ avoiding the use of unnecessary extra atoms.

KEYWORDS: answer set programming, $\mathbf{N_2}$ logic, update programs, strong negation, AGM postulates

1 Introduction

In recent years, a lot of theoretical work has been devoted to explore the relationships between intuitionistic logic and answer set programming (ASP) (Lifschitz *et al.* 2001; Osorio *et al.* 2001; Osorio *et al.* 2004; Osorio *et al.* 2005a). These results have provided a characterization of ASP by intuitionistic logic as follows: a literal is entailed by a program in the answer set semantics if and only if it belongs to every intuitionistically complete and consistent extension of the program formed by adding only negated literals (Osorio *et al.* 2004). The idea of these completions using intermediate logics in general is due to Pearce (Pearce 1999b). This logical approach provides the foundations to define the notion of nonmonotonic inference of any propositional theory (using the standard connectives) in terms of a monotonic logic (namely intuitionistic logic), see Lifschitz *et al.* (2001) Osorio *et al.* (2001) and Osorio

[1] This paper can be considered as an extended and revised version of the papers presented in (Osorio and Zacarías 2003; Osorio and Zacarías 2004).

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

et al. (2004). As Pearce *et al.* noticed, if we include strong negation we just have to move to Nelson logics (Lifschitz *et al.* 2001). We select here $\mathbf{N_2}$ logic, because it characterizes adequately strong equivalence. A formalization of $\mathbf{N_2}$ can be obtained from intuitionistic logic by adding the axiom schema $F \vee (F \rightarrow G) \vee \neg G$ and the axioms of Nelson logic.

When an intelligent agent obtains new knowledge consistent with its current information that knowledge is added to the agent's knowledge base. If however the new knowledge contradicts the agent's current information, a method for incorporating this new knowledge must be developed. In this paper we present our approach to update nonmonotonic knowledge bases represented as logic programs under the answer set semantics (Gelfond and Lifschitz 1988; Gelfond and Lifschitz 1991). Some well-known recent proposals are presented in Eiter *et al.* (2002) and Alferes and Pereira (2002) accompanied by very good reviews with many references.

In particular, with the purpose of obtaining a reliable update process for providing agents with advanced reasoning capabilities, the authors of Eiter *et al.* (2002) present an approach in which new information is incorporated into the current knowledge base subject to a causal rejection principle. Causal rejection enforces that in case of conflicts between rules, more recent rules are preferred and older rules are overridden (Eiter *et al.* 2002). As was noted in Eiter *et al.* (2002), causal rejection is not novel, and what they provided was a simple and rigorous realization of this principle, as well as the formalization of a semantics for updating logic programs based on it. We developed the present work based on this principle also, aiming to express program updates succinctly while preserving relevant properties.

In our work we analyze the approach presented in Eiter *et al.* (2002) because under this approach a program is explicitly constructed. We believe that this is important from a computational perspective and also that such an approach is more suitable to study the properties of updates based on the causal rejection principle, since it provides insight into the operation of the principle itself. We go a step further by presenting an alternative but equivalent definition of update sequences as presented in (Eiter *et al.* 2002), for the case when considering two programs and that avoids the need of new atoms. We consider our definition to be more concise and accessible.

In this paper we consider many of the properties presented in (Eiter *et al.* 2002) because there the authors present a comprehensive analysis about the properties that an update operator should have. Moreover, we present new properties for update operators through an extensive analysis and making use of the new characterization of ASP by intermediate logics (Lifschitz *et al.* 2001; Osorio *et al.* 2001; Osorio *et al.* 2004). In this paper, we consider properties similar to the well-known AGM postulates. We think that it is necessary to reinterpret them in the context of nonmonotonic reasoning via ASP. As a starting point, we study Dalal's Principle (Katsuno and Mendelzon 1991) of Independence of Syntax, according to which the meaning of the knowledge that results from an update must be independent of the syntax of the original knowledge, as well as independent of the syntax of the update itself. In (Eiter *et al.* 2002) the authors analyze and interpret the AGM postulate

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

corresponding to Dalal's principle as follows:

$$T_1 \equiv T_2 \text{ implies } S(K \oplus_f T_1) = S(K \oplus_f T_2)$$

where T_1 and T_2 are any theories, $T_1 \equiv T_2$ denotes that T_1 and T_2 are equivalent, i.e. they have the same answer sets. By $S(P)$ we denote the set of all answer sets of P, "$\oplus_f$" is a general update operator. This interpretation expresses a very demanding principle of independence of syntax, due to the fact that the AGM postulates were introduced for monotonic logics. We propose to reconsider the AGM postulates (Alchourrón *et al.* 1985) under our new interpretation. To this aim we have introduced in (Osorio and Zacarías 2003) a new property, which we call Weak Independence of Syntax, as follows:

$$\text{(WIS): } T_1 \equiv_{N_2} T_2 \text{ implies } S(K \oplus_f T_1) = S(K \oplus_f T_2).$$

where T_1 and T_2 are any theories, $T_1 \equiv_{N_2} T_2$ denotes that T_1 and T_2 are equivalent in $\mathbf{N_2}$, in which case we say that they are equivalent knowledge. We consider this property to be desirable for any update operator based on ASP. We show that the proposal introduced in Eiter *et al.* (2002) for updates satisfies this principle to a great extent. In fact we show that for programs without tautologies, this principle holds. We should point out that this property is accepted as much in belief revision as in updates, as it is shown in Eiter *et al.* (2002).

The main contributions of our paper can be summarized as follows. First, we present a reinterpretation of the AGM postulates for ASP based on Nelson logics. Second, we present an analysis of how the update semantics presented in Eiter *et al.* (2002) can be expressed in an alternative form without relying on new atoms. Finally, we analyze the semantics presented in Eiter *et al.* (2002) and our equivalent form in terms of our adaptation of the AGM postulates.

Our paper is structured as follows: in Section 2 we present the general syntax of clauses, we also provide the definition of answer sets for logic programs as well as some background on logic, in particular on $\mathbf{N_2}$ logic. In addition, we review the AGM postulates and present the definition of update sequences given in (Eiter *et al.* 2002). Next, in Section 3 we introduce our $\mathbf{N_2}$ properties motivated by the AGM postulates. Section 4 presents an approach for update programs characterized by its simplicity. In Section 5 we introduce an alternative but equivalent definition for update programs as presented in Eiter *et al.* (2002) and analyze its properties. Finally, the conclusions are drawn in Section 6.

2 Background

In this section we present a brief background on intuitionistic and intermediate logics and introduce the notation and key concepts to be used in this paper. In addition we discuss the AGM postulates and present the definition of update sequences given in Eiter *et al.* (2002).

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

2.1 Intuitionistic and Intermediate Logics

Intuitionistic Logic was introduced at the beginning of the twentieth century and it was developed as a formal system around the 1930s. It is often referred to as Heyting's Constructive Logic and here we will represent it by the symbol **I**. It was proposed as an alternative to standard propositional logic, which we will represent by **C**. The main feature of intuitionistic logic is that in order to be true a proposition must be constructively verified, i.e. a proof of it must be constructed. For this reason, intuitionism is usually bound to concepts like proof and knowledge, rather than to the traditional classical notion of truth. The set $\mathscr{H} := \{\wedge, \vee, \rightarrow, \bot\}$ denotes the connectives of intuitionistic logic. All the connectives in $\mathscr{H}$ are 2-place connectives, except for the 0-place $\bot$, known as *falsum*. For any formula p, the expression $\neg p$ will be understood as an abbreviation of $p \rightarrow \bot$ and $\top$ is an abbreviation of $\bot \rightarrow \bot$. In addition, in this paper we may use the $\leftarrow$ operator instead of the $\rightarrow$ operator solely as a notational convenience.

For a well known axiomatic formalization of **I** see Mendelson (1987). **I** is strictly contained in classical logic in the sense that all theorems of **I** are also theorems of **C**, but the converse doesn't hold. Note that many well known theorems of **C** are not theorems of **I**. The best known examples usually involve negation, like $p \vee \neg p$, and $p \leftrightarrow \neg\neg p$, but there are also formulas without negation that are theorems of **C** and are not provable in **I**, for example $((p \rightarrow q) \rightarrow p) \rightarrow p$ (*Pierce's law*).

By adding supplementary axioms to intuitionistic logic, we obtain the logics that are usually known in the literature as *intermediate logics*. All of them contain all of the intuitionistic theorems, and are contained in the theorems of classical logic. If this containment is strict, they are called proper intermediate logics. Here we will just use the term *intermediate logic* or **H**-*logic* to refer to any logic that is as least as strong as **I**, but strictly weaker than **C**[2]. Intermediate logics form a lattice in which the supremum of all intermediate logics is the unique lower cover of **C**. Many names for this logic can be found in the literature, like *Smetanich logic*, the logic of *Here and There*, and Gödel 3-valued Logic ($\mathbf{G}_3$).

2.1.1 Gödel multivalued logics

These logics are defined by generalizing the idea of truth tables and evaluation functions of classical logic. Gödel defined the multivalued logics $\mathbf{G}_i$, with values in $\{0, 1, \ldots, i-1\}$, with the following evaluation function I:

- $I(B \leftarrow A) = i - 1$ if $I(A) \leqslant I(B)$ and $I(B)$ otherwise.
- $I(A \vee B) = \max(I(A), I(B))$.
- $I(A \wedge B) = \min(I(A), I(B))$.
- $I(\bot) = 0$.

An interpretation is a function $I : \mathscr{L} \rightarrow \{0, 1, \ldots, i-1\}$ that assigns a truth value to each atom in the language. The interpretation of an arbitrary formula is obtained

[2] In a slight abuse of notation we will use the same symbol to refer to a formal logic system as well as to its theorems.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

by propagating the evaluation of each connective as defined above. Recall that $\neg$ and $\top$ were introduced as abbreviations of other connectives.

For a given interpretation I and a formula F we say that I is a *model* of F if $I(F) = i - 1$. Similarly I is a *model* of a program P if it is a model of each of the formulas contained in P. If F is modeled by every possible interpretation we say that F is a *tautology*.

Notice that $\mathbf{G}_2$ coincides with classical logic. The 3-valued logic $\mathbf{G}_3$ is particularly useful for some of our results.

2.2 Intermediate logics and strong negation extensions

In Nelson (1949) Nelson introduces a logic that extends intuitionistic logic by adding a new negation connective called *strong negation* and represented by $\sim$. Intuitively, the strong negation connective means that something is known to be false, not only assumed false due to the absence of a proof. It allows the constructive falsification of a proposition, in the same sense as intuitionistic logic allows constructive verification. This logic is given by the standard connectives of intuitionistic propositional logic, $\mathscr{H}$, plus the strong negation connective $\sim$.

We will denote by **N** the logic obtained by adding the following axiom schema to Heyting's axiomatization of intuitionistic logic (Mendelson 1987) ($\alpha \leftrightarrow \beta$ stands for $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$).

A1: $\sim(\alpha \rightarrow \beta) \leftrightarrow \alpha \wedge \sim\beta$
A2: $\sim(\alpha \wedge \beta) \leftrightarrow \sim\alpha \vee \sim\beta$
A3: $\sim(\alpha \vee \beta) \leftrightarrow \sim\alpha \wedge \sim\beta$
A4: $\alpha \leftrightarrow \sim\sim\alpha$
A5: $\sim\neg\alpha \leftrightarrow \alpha$
A6: $\sim\alpha \rightarrow \neg\alpha$ for atomic α

Since $\sim\alpha \rightarrow \neg\alpha$ is a theorem in **N** for any α (not only atoms) (van Dalen 1986; Gurevich 1977), the $\sim$ connective is called strong negation. These axioms are given in Vorob'ev (1952a) and Vorob'ev (1952b), where **N** is presented as an extension of intuitionistic logic **I**, i.e. formulas without the $\sim$ connective are theorems of **N** iff they are theorems of **I**. Moreover, the same axioms **A1** to **A6** can be added to any intermediate logic **H** to obtain a *least strong negation extension* of **H**, which we will denote by **N**(**H**). Strong negation extensions of intermediate logics are sometimes called constructive logics, but here we will just call them **N**-logics following Kracht (1998). Sometimes, we will refer to the *intuitionistic fragment* of a **N**-logic **N**, i.e. the intermediate logic whose theorems are exactly those of **N** on the fragment without $\sim$. This fragment will be denoted as **H**(**N**). Least strong negation extensions of intermediate logics form a lattice that was deeply studied by Kracht in Kracht (1998). There he used algebraic techniques to prove many results, some of which will be used in this work. Two particular elements of this lattice will be important for the present work: the logic **N**, extension of **I** and lower bound of the lattice, and the unique upper bound of it, i.e. the extension of $\mathbf{G_3}$ which we will denote by $\mathbf{N_2}$. The logic $\mathbf{N_2}$ can be obtained by adding the axiom schema $\alpha \vee (\alpha \rightarrow \beta) \vee \neg\beta$ to **N**.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

A particular feature of **N** is that the defined connective $\leftrightarrow$ is not in general congruential. For instance, it is known that $\sim \neg a \leftrightarrow a$ is a theorem, however $\sim\sim \neg a \leftrightarrow\sim a$ is not a theorem. For the restricted class of formulas where $\sim$ only applies to atoms, our $\leftrightarrow$ connective is again congruential as discussed in (Ortiz and Osorio 2005). We will assume this result freely in the rest of the paper.

It is well–known that provability in **N** can be reduced to provability in intuitionistic logic following a simple syntactic transformation (Rasiowa 1974). First, strong negation is moved towards the atoms using axioms **A1** to **A5**. Next, for every atom a of the underlying language, a new atom a' is introduced and all the formulas $\sim a$ are replaced by a'. Finally, formulas of the form $a' \rightarrow \neg a$ are added for each of the new atoms a'. For example, if we want to know if $\sim(a \wedge b) \rightarrow \neg b$ is a theorem in **N**, we can just check if $((a' \rightarrow \neg a) \wedge (b' \rightarrow \neg b)) \rightarrow ((a' \vee b') \rightarrow \neg b)$ is a theorem in intuitionistic logic. By abuse of notation we can understand $\sim a$ as a new atom and we only ask if $((\sim a \rightarrow \neg a)\wedge(\sim b \rightarrow \neg b)) \rightarrow (\sim a \vee \sim b) \rightarrow \neg b$ is a theorem in **I**. Similarly, provability in $\mathbf{N_2}$ can be reduced to provability in $\mathbf{G_3}$, the details of this reduction can be found in Ortiz and Osorio (2005). We will assume this result freely in the rest of the paper.

2.3 General notation

We will represent the set of connectives of Nelson's logics by $\mathscr{N} := \mathscr{H} \cup \{\sim\}$, where $\sim$ is an unary connective. **N**-formulas are the formulas constructed as in standard propositional logic using the connectives in $\mathscr{N}$. For any formulas α, β the formula $\alpha \leftrightarrow \beta$ is an abbreviation of $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$. As was mentioned before, the $\neg$ connective can be defined alternatively as $\neg\varphi := \varphi \rightarrow \bot$. Formulas are constructed over a set of atoms $\mathscr{A}$. An *atom* is just a propositional variable. A $\sim$-*literal* is either an atom a or the strong negation of an atom, $\sim a$. A $\neg$-*literal* is either an atom a or the weak negation of an atom, $\neg a$. When we use the term *literal* alone, we mean an $\sim$-literal l. For a literal l, the *complementary literal*, $\sim l$, is $\sim a$ if $l = a$, and a if $l = \sim a$, for some atom a. For a set S of literals, we define $\sim S = \{\sim l \mid l \in S\}$. Similarly, for a set S of formulas we define $\neg S = \{\neg F \mid F \in S\}$. Also, if $S = \{P_1, \ldots, P_n\}$ is a set of formulas, we define $\bigwedge S = F_1 \wedge \ldots \wedge F_n$ and $\bigvee S = F_1 \vee \ldots \vee F_n$. We denote by $Lit_{\mathscr{A}}$ the set $\mathscr{A} \cup \sim\mathscr{A}$ of all literals over a set of atoms $\mathscr{A}$. In addition, we define $\overline{S} = S \cup \neg\{Lit_{\mathscr{A}} \setminus S\}$. A theory is just a finite set of formulas. An *rule* is an expression of the form

$$A \leftarrow B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_n \qquad (1)$$

where A and each B_i are literals, $\top$, or $\bot$. If $B_1, \ldots, B_m, \neg B_{m+1}, \ldots, \neg B_n$ is $\top$ then rule (1) is a fact and may be written simply as A. If A is $\bot$ then rule (1) is a constraint, on the other hand, if A is $\top$ then the rule is a tautology. An *extended logic program* (ELP) (Gelfond and Lifschitz 1991) is a finite set of rules. Usually, $\mathscr{A}$ will be identified with the set of all atoms occurring in a program P, in which case we call it the *signature* of program P. Every rule can be identified with a corresponding formula of the form $\beta_1 \wedge \cdots \wedge \beta_m \wedge \neg\beta_{m+1} \wedge \cdots \wedge \neg\beta_n \rightarrow \alpha$, where α

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

and each β_i are literals. Also, a rule r may be abbreviated as $H(r) \leftarrow B(r)$, where $H(r)$ and $B(r)$ represent the formulas that comprise the head and body of the rule, respectively.

Due to transformations, our programs can surpass the class of extended logic programs. Hence, it is convenient to define an *augmented logic program* (Lifschitz *et al.* 1999) as a finite set of formulas F that adhere to the following CFG:

$$\begin{array}{lll} R & ::= & F \rightarrow l \\ F & ::= & (F\ B\ F) \mid \neg F \mid \bot \mid \top \mid l \text{ , where } l \text{ is a literal} \\ B & ::= & \vee \mid \wedge \end{array}$$

For example, the formula $(a \vee \neg \sim b) \rightarrow \sim p$ can be derived according to the previous CFG. Parentheses may be omitted following the usual conventions regarding the precedence of operators and conjunction may be expressed with the comma operator instead. Henceforth, when we use the term *program* alone, we mean an augmented logic program.

With respect to update operators, $\oplus_f$ stands for a general update operator, while $\oplus_i$ for some i such that $1 \leqslant i \leqslant n$ represents a specific update operator whose definition is introduced.

2.4 Answer sets

Following the characterization of answer sets for logic programs in terms of intermediate logics provided by Pearce in Pearce (1999b), we adopt the following theorem as a definition of an answer set, which is discussed in detail in Osorio *et al.* (2005b).

Let us introduce the notation $T \Vdash_{N_2} M$ to stand for the phrase T is consistent and $T \vdash_{N_2} M$. We also clarify that by a consistent set of literals M, we mean that for all literals $l \in M$ we have either $l \in M$ or $\sim l \in M$, but not both. Finally, we define the complement of M as $\widetilde{M} = Lit_{\mathscr{A}} \setminus M$.

Theorem 1
(Osorio *et al.* 2005b; Ortiz and Osorio 2005). Let P be a program over a set of atoms $\mathscr{A}$, and let M be a consistent set of literals such that $M \subseteq Lit_{\mathscr{A}}$. M is an answer set of P iff $P \cup \neg\widetilde{M} \cup \neg\neg M \Vdash_{N_2} M$.

Our definition yields different results with respect to that given in Gelfond and Lifschitz (1991) only in that according to ours inconsistent programs have no answer sets.

In addition we adopt the notion of strong equivalence, which represents a property that is useful in our work.

Theorem 2
(Lifschitz *et al.* 2001). For any programs P_1 and P_2, $P_1 \equiv_{N_2} P_2$ iff for every program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets.

We also take into account the notion of conservative extension, a property in which a program preserves its original meaning despite the addition of rules.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Definition 1
(Baral 2003) Let P and P' be a pair of programs over sets of atoms $\mathscr{A}$ and $\mathscr{A}'$ respectively such that $P \subseteq P'$. We say that P' is a *conservative extension* of P if the following condition holds: M is an answer set for P iff there is an answer set M' for P' such that $M = M' \cap Lit_{\mathscr{A}}$.

Finally, we will introduce two additional results. But first we introduce the following clarification.

We define that a literal l *occurs* in a formula F recursively as follows

l occurs in l	where l is a literal
l occurs in $\alpha \circ \beta$	iff l occurs in α or l occurs in β

where α and β are formulas and $\circ$ denotes one of the operators with which we construct our formulas as presented in the CFG introduced in Section 2.3. The previous definition allows us to resolve possible ambiguities. For instance, a does not occur in $\sim a$ and similarly $\sim a$ does not occur in a.

We say that a literal occurs under the scope of the $\neg$ symbol when it is affected directly by the $\neg$ symbol, or is part of a formula affected by the $\neg$ symbol. For example, in the formula $a \leftarrow \neg(x \wedge b)$ the literal x occurs under the scope of $\neg$. Now consider the formula $x \leftarrow \neg(x \wedge b)$. In this formula, the first occurrence of x is not under the scope of $\neg$ while the second does occur under the scope of $\neg$. We are interested in the case when a literal may *only* occur under the scope of the $\neg$ symbol for some formula F, and consequently in a program P, where the previous condition holds for all formulas F identified with the rules of P.

The first result can be considered of interest by itself [3].

Lemma 1
Let P be a program over a set of atoms $\mathscr{A}$ and x a literal that does not occur in P. Let F be a formula over a set of atoms $\mathscr{F}$ such that $Lit_{\mathscr{F}} \subseteq Lit_{\mathscr{A}}$. Then $P \cup \{x \leftarrow F\} \equiv P \cup \{x \leftrightarrow F\}$.

Proof
See Appendix B. □

The second result is the following theorem.

Theorem 3
Let P be a program over a set of atoms $\mathscr{A}$ and x a literal that if occurs in P, it occurs only under the scope of the $\neg$ symbol. Let F be a formula over a set of atoms $\mathscr{F}$ such that $Lit_{\mathscr{F}} \subseteq Lit_{\mathscr{A}}$ and x does not belong to $Lit_{\mathscr{F}}$. Then $P \cup \{x \leftarrow F\} \equiv P \cup \{x \leftrightarrow F\}$.

[3] Essentially the same result was introduced and proved in (Ferraris and Lifschitz 2005) as the Completion Lemma.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Proof
For a set of literals M we denote by $P^{M(x)}$ the program obtained by substituting x by $\top$ if $x \in M$ or by $\bot$ if $x \notin M$. The substitution only occurs under the scope of a $\neg$ symbol in P.

M is an answer set of $P \cup \{x \leftarrow F\}$
iff M is an answer set of $P^{M(x)} \cup \{x \leftarrow F\}$
(by Lemma 1, $P^{M(x)} \cup \{x \leftarrow F\} \equiv P^{M(x)} \cup \{x \leftrightarrow F\}$)
iff M is an answer set of $P^{M(x)} \cup \{x \leftrightarrow F\}$
iff M is an answer set of $P \cup \{x \leftrightarrow F\}$. □

2.5 AGM Postulates

Although it is not possible to characterize a revision exclusively on logical terms, the general properties of its associated revision function can be defined and algorithms to compute it can be developed, at least for some cases (Alchourrón *et al.* 1985). In order to prove if an algorithm is adequate, we need to establish whether its associated function follows the properties that would normally be expected to hold in a belief revision process. In (Alchourrón *et al.* 1985) several *rationality postulates* are proposed as a standard for analyzing revision functions. These are referred to as the AGM postulates, in honor of their authors, and are described next.

K-1: for any sentence ϕ and any belief set K, $K \odot \phi$ is a belief set.
K-2: $\phi \in K \odot \phi$.
K-3: $K \odot \phi \subseteq K + \phi$.
K-4: if $\neg\phi \notin K$ then $K + \phi \subseteq K \odot \phi$.
K-5: $K \odot \phi = K_{\bot}$ if and only if $\vdash \neg\phi$.
K-6: if $\vdash \phi \leftrightarrow \psi$ then $K \odot \phi = K \odot \psi$.

In the previous statements $\odot$ represents an arbitrary revision operator as discussed in Alchourrón *et al.* (1985) and not specifically based on answer set semantics, $+$ represents the expansion of a belief set, i.e. adding a sentence regardless of the consistency of the new belief set. A revision operator represents a function taking a belief set and a sentence as arguments and producing a belief set as a result. By belief set we mean a set of sentences closed under logical consequences. The first postulate requires that the output of the revision function must be a belief set. The second postulate guarantees that the new sentence ϕ will be accepted. The third postulate states that a revision knows at most, as much as the expansion. In the case in which the new information ϕ does not contradict the belief set K which means $\neg\phi \notin K$, the **K-4** postulate guarantees, in conjunction with **K-3**, that revision will be equivalent to expansion. Because the goal of revision is to produce a new consistent belief set, postulate **K-5** establishes that the new belief set has to be consistent, unless the input sentence ϕ is inconsistent itself. The content of the input sentence ϕ should determine the revision independently of its syntax. This means belief revisions should be analyzed on the knowledge level and not on the syntactic

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

level, therefore logically equivalent sentences should lead to equivalent revisions as postulate **K-6** says.

Postulates **K-1** to **K-6** establish the fundamental properties that any belief revision function should satisfy, for this reason they are called the *basic* set of postulates. Together, these postulates aim to capture the basic properties that should be respected in an ideal form of belief revision.

In Katsuno and Mendelzon (1992) the authors argue that the AGM postulates are adequate for the incorporation of new information about a static world, but not for new information about changes in the world caused by an agent. They formalize the latter behavior by a set of update postulates. However, all of the basic AGM postulates except **K-1** have direct correspondences in the update postulates presented in Katsuno and Mendelzon (1992), hence they can be considered as a core set of postulates for belief revision as well as for update.

2.6 Update programs

In (Eiter *et al.* 2002) an *update sequence* $\boldsymbol{P}$ is defined as a series $(P_1, P_2, \ldots, P_n)$ of extended logic programs (ELPs). Let us consider the definition of update sequence given in (Eiter *et al.* 2002) but only for the case of two programs and let us make a slight change of notation. Our proposal can be extended to the general case $(P_1, P_2, \ldots, P_n)$ in the iterative form as shown in (Eiter *et al.* 2002). Under certain conditions, which exclude possibilities for local inconsistencies, the iterativity property holds (Eiter *et al.* 2002). $\boldsymbol{P}$ is an update sequence over a set of atoms $\mathscr{A}$ iff $\mathscr{A}$ represents the set of atoms occurring in the rules of the constituting elements P_i of $\boldsymbol{P}$ $(1 \leqslant \mathrm{i} \leqslant 2)$. Given an update sequence $\boldsymbol{P} = (P_1, P_2)$ over $\mathscr{A}$, we assume a set $\mathscr{A}^*$ extending $\mathscr{A}$ by new, pairwise distinct atoms $rej(r)$ and A_i, for each r occurring in $\boldsymbol{P}$, each atom $A \in \mathscr{A}$, and each i, $1 \leqslant i \leqslant 2$. 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

(Eiter *et al.* 2002) Given an update sequence $\boldsymbol{P} = (P_1, P_2)$ over a set of atoms $\mathscr{A}$, we define the update program $\boldsymbol{P}_{\oplus_1} = P_1 \oplus_1 P_2$ over $\mathscr{A}^*$ consisting of the following items:

$$
\begin{array}{ll}
(i) \text{ all constraints in } P_1 \cup P_2; & \\
(ii) \text{ for each } r \in P_1; & \\
\qquad \left.\begin{array}{l} L_1 \leftarrow B(r), \neg rej(r). \\ rej(r) \leftarrow B(r), \sim L_2. \end{array}\right\} & \text{if } H(r) = L; \\
(iii) \text{ for each } r \in P_2; & \\
\qquad L_2 \leftarrow B(r). & \text{if } H(r) = L; \\
(iv) \text{ for each literal } L \text{ occurring in } \boldsymbol{P}; & \\
\qquad L_1 \leftarrow L_2 & \\
\qquad L \leftarrow L_1. &
\end{array}
$$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

This definition relies on the fact that rules in program P_2 cannot be rejected, which allows us to construct a simplified program as was noted in (Eiter *et al.* 2002). It is important to note that the update sequence $\boldsymbol{P}$ is also an extended logic program. The answer sets of $\boldsymbol{P}$ are defined in terms of the answer sets of $\boldsymbol{P}_{\oplus_1}$.

Definition 3
(Eiter *et al.* 2002) Let $\boldsymbol{P} = (P_1, P_2)$ be an update sequence over a set of atoms $\mathscr{A}$. Then, $S \subseteq Lit_{\mathscr{A}}$ is an update answer set of $\boldsymbol{P}$ iff $S = S' \cap \mathscr{A}$ for some answer set S' of $\boldsymbol{P}_{\oplus_1}$. The collection of all update answer sets of $\boldsymbol{P}$ is denoted by $\mathscr{U}(\boldsymbol{P})$.

An important remark in relation to update programs is presented in Eiter *et al.* (2002). Update programs 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 (Eiter *et al.* 2002).

3 Properties

In this section we present some basic properties about update operators. The first four are taken from (Eiter *et al.* 2002) and are all satisfied by the $\oplus_1$ operator. The remaining are adapted versions of the AGM postulates. In the statements that follow $\oplus_f$ denotes an arbitrary update operator and P a program. Given two programs P and P', we write $P \equiv P'$ if they have the same answer sets.

Initialization: $\emptyset \oplus_f P \equiv P$

Eiter *et al.* (2002) present this property as follows: this property states that the update of an initial empty knowledge base yields just the update itself.

Idempotence: $P \oplus_f P \equiv P$

This property means that the update of program P by itself has no effect (Eiter *et al.* 2002).

Noninterference: If P_1 and P_2 are programs defined over disjoint alphabets, then $P_1 \oplus_f P_2 \equiv P_2 \oplus_f P_1$

This property implies that the order of updates which do not interfere with each other is immaterial (Eiter *et al.* 2002).

Augmented update: If $P_1 \subseteq P_2$ then $P_1 \oplus_f P_2 \equiv P_2$

Updating with additional rules makes the previous update obsolete (Eiter *et al.* 2002). Note that the idempotence property represents a special case of this property.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

The following properties are motivated by the AGM postulates. We observe that the interpretation given in (Eiter *et al.* 2002) of the AGM postulates is sometimes very demanding, particularly because the AGM postulates were introduced for monotonic logics. However, since answer set programming is related to the well known $\mathbf{N_2}$ monotonic logic, it makes sense to consider properties that relate the monotonic part of answer sets to its non-monotonic side. That is the objective of the following properties.

Let P, P_1, P_2, and R be programs and x a rule.

BK-1: $P \oplus_f \{x\}$ is a program.
BK-2: $P_1 \oplus_f P_2 \vdash_{N_2} P_2$.
BK-3: $P_1 \cup P_2 \vdash_{N_2} P_1 \oplus_f P_2$.
BK-4: if $P_1 \cup P_2$ has answer sets then $P_1 \cup P_2 \equiv P_1 \oplus_f P_2$.
BK-5: $P_2 \equiv_{N_2} \bot$ implies $(P_1 \oplus_f P_2) \equiv_{N_2} \bot$.
BK-6: if $P_1 \equiv_{N_2} P_2$ then $P \oplus_f P_1 \equiv P \oplus_f P_2$.

In the following we call these properties the **BK-ASP** properties. In Section 2.5 we restated the 6 principles introduced in (Alchourrón *et al.* 1985) (**K-1**, . . . , **K-6**), we now translate principle **K-i** from (Alchourrón *et al.* 1985) to property **BK-i** in our approach. However, it should be noted that this is not the only translation possible. In **BK-1** the notion of belief set is replaced by logic program. In the AGM postulates a sentence is part of a belief set whenever the belief set logically entails the sentence, for **BK-2** and **BK-3** we require that a program has to be inferred in $\mathbf{N_2}$. Like their counterparts, together properties **BK-3** and **BK-4** state that program update should be equivalent to expansion whenever possible, only that equivalence is understood as having the same answer sets. **BK-5** establishes that an inconsistent program in $\mathbf{N_2}$ will lead to an inconsistent update. We adopt only one of the implications of the original **K-5** postulate; because we consider excessive, in our context, to demand an update operator to derive from an inconsistent program, a new consistent program given a possibly completely independent update[4]. **BK-6** says that equivalent programs should lead to equivalent updates, but the programs have to be equivalent in $\mathbf{N_2}$, i.e., strongly equivalent and it is only expected that the updates will have the same answer sets. Finally, we present an additional property and prove that it is in fact equivalent to **BK-6**.

Theorem 4

An update operator $\oplus_f$ satisfies the **BK-6** (WIS) property if and only if it satisfies the following property

BK-0: if $P_2 \vdash_{N_2} R$ then $P_1 \oplus_f P_2 \equiv P_1 \oplus_f (P_2 \cup R)$.

[4] Consider for instance updating the program $\{a, \leftarrow a\}$ with $\{b\}$.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Proof

To prove the right implication we have that

$$P_2 \vdash_{N_2} R \text{ implies } P_2 \equiv_{N_2} P_2 \cup R$$

now by applying **BK-6** we obtain

$$P_1 \oplus_f P_2 \equiv P_1 \oplus_f (P_2 \cup R) \text{ as desired.}$$

For the left implication, by applying **BK-0** we obtain

$$P \oplus_f P_1 \equiv P \oplus_f (P_1 \cup P_2)$$

similarly,

$$P \oplus_f P_2 \equiv P \oplus_f (P_2 \cup P_1)$$

hence

$$P \oplus_f P_1 \equiv P \oplus_f P_2 \text{ as desired.} \quad \square$$

4 A basic definition for updates

In this section we present a simpler new definition for updates. We also prove that it is equivalent to $\oplus_1$ for a class of programs involving tautologies, which we will define as *tau-comp*. The relevance of this approach relies on its simplicity and the insight it provides on the $\oplus_1$ operator, which will become clearer when we introduce a possible refinement of this new operator as well as our alternative definition of $\oplus_1$. The notational conventions presented in Section 2.6 apply to this section as well.

Definition 4

Given an update sequence $\boldsymbol{P} = (P_1, P_2)$ over a set of atoms $\mathscr{A}$, we define the update program $\boldsymbol{P}_{\oplus_2} = P_1 \oplus_2 P_2$ over $\mathscr{A}^*$ consisting of the following items:

(*i*) all constraints in $P_1 \cup P_2$;
(*ii*) for each $r \in P_1$;
$L \leftarrow B(r), \neg \sim L.$ if $H(r) = L$;
(*iii*) all rules $r \in P_2$.

Example 1

To illustrate this definition consider the following update sequence.

Let P_1 be:
sleep ← night, ¬watch-tv, ¬other.
night.
tv-on ← ¬tv-broke.
watch-tv ← tv-on.

Let P_2 be:
∼tv-on ← power-failure.
∼tv-on ← assignment-due, working.
assignment-due.
working.
other ← working.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

If we update P_1 with P_2 we obtain the following program $P_1 \oplus_2 P_2$:

sleep ← night, ¬watch-tv, ¬other, ¬ ∼sleep.
night ← ¬ ∼night.
tv-on ← ¬tv-broke, ¬ ∼tv-on.
watch-tv ← tv-on, ¬ ∼watch-tv.
∼tv-on ← power-failure.
∼tv-on ← assignment-due, working.
assignment-due.
working.
other ← working.

This program has a unique expected answer set, namely

{night, other, assignment-due, working, ∼tv-on}.

Definition 5

We say that a program P is tau-comp w.r.t. a signature $\mathscr{A}$ if every rule of the form $l \leftarrow l$ belongs to P, where l is a literal over $\mathscr{A}$.

We will now prove that $\oplus_1$ and $\oplus_2$ are equivalent for tau-comp programs. The proof is based on results from (Osorio *et al.* 2001; Osorio *et al.* 2004; Pearce 1999a).

Theorem 5

Let P_1 and P_2 be extended logic programs, if P_2 is tau-comp then $P_1 \oplus_1 P_2 \equiv P_1 \oplus_2 P_2$.

Proof

The idea of the proof is to apply transformations to $P_1 \oplus_1 P_2$ with respect to answer sets over the $\mathscr{A}$ signature, to obtain $P_1 \oplus_2 P_2$. At some point we may delete rules involving a particular literal that we regard as temporary, i. e., a literal that does not form part of the $\mathscr{A}$ signature of the programs in the update sequence. This is due to the fact that the program that contains such rules is a conservative extension of the program that excludes them.

Consider a particular positive literal L (i.e. a literal consisting of an atom a not preceded by ∼). By construction of $P_1 \oplus_1 P_2$ (defined in Point (*iv*) of Definition 2 given in Section 2.6) the program includes the formulas $L_1 \leftarrow L_2$ and $L \leftarrow L_1$ for the literal L.

Also, since P_2 is tau-comp then it includes the rules $L_2 \leftarrow L$ for the literal L.

Hence, $P_1 \oplus_1 P_2 \vdash_{N_2} L_1 \leftrightarrow L_2$ and $P_1 \oplus_1 P_2 \vdash_{N_2} L_2 \leftrightarrow L$.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

So we can replace each of the L_1 and L_2 literals by L in the rest of the program as discussed in Rasiowa (1974) and Ortiz and Osorio (2005)[5]. Then, we can eliminate rules in Point (*iv*) of Definition 2 given previously as well as rules of the form $L_2 \leftarrow L$ (L_1 and L_2 are temporary literals).

So, the rules become

$L \leftarrow B(r), \neg rej(r)$ and
$rej(r) \leftarrow B(r), \sim L$ corresponding to program P_1.
$L \leftarrow B(r)$ corresponding to program P_2.

Now, observe that by Theorem 3 the answer sets of the program are the same if we replace $rej(r) \leftarrow B(r), \sim L$ by $rej(r) \leftrightarrow (B(r) \wedge \sim L)$. Observe that this transformation creates a rule whose form is different from that of rules of ELPs, however this situation has already been considered (Osorio *et al.* 2004; Pearce 1999a; Ortiz and Osorio 2005). Then we can replace $rej(r)$ in $L \leftarrow B(r) \wedge \neg rej(r)$ to obtain:

$$L \leftarrow B(r) \wedge \neg(B(r) \wedge \sim L) \qquad (*)$$

then we can delete the rule $rej(r) \leftrightarrow (B(r) \wedge \sim L)$ since $rej(r)$ is a temporary literal. Finally, to obtain the desired program, the rule of the form (*) can be replaced by $L \leftarrow B(r), \neg \sim L$ following the next transformation

$$L \leftarrow B(r) \wedge (\neg B(r) \vee \neg \sim L)$$

by applying the De Morgan law, and

$$L \leftarrow (B(r) \wedge \neg B(r)) \vee (B(r) \wedge \neg \sim L)$$

by applying the distributive property. The previous rule can in turn be divided into two rules

$$L \leftarrow (B(r) \wedge \neg B(r)) \qquad (1)$$
$$L \leftarrow (B(r) \wedge \neg \sim L) \qquad (2)$$

Rule (1) is a tautology and can therefore be eliminated, leaving only rule (2) which has the desired form according to Definition 4. This process can be applied analogously for the rest of the literals. In this way, except for the tautologies of P_2, which have no effect on the answer sets of the derived program, we reach program $P_1 \oplus_2 P_2$ as desired. □

Theorem 6
The update operator $\oplus_2$ satisfies the properties **BK-1**, **BK-2**, **BK-3**, **BK-5**, and **BK-6**.

Proof
Properties **BK-1**, **BK-2**, **BK-3**, and **BK-5** follow directly by construction. The **BK-6** property follows straightforward by construction and Theorem 2. □

[5] For a positive literal consisting of an atom a we also replace the atoms of the corresponding negative literals $\sim a$.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

This allows us to formalize our first result regarding the satisfability of the WIS property for the $\oplus_1$ operator.

Corollary 1
Let P be an ELP program and let P_1 and P_2 be tau-comp ELPs.
If $P_1 \equiv_{N_2} P_2$ then $P \oplus_1 P_1 \equiv P \oplus_1 P_2$.

Proof
The result is immediate from the equivalence of the $\oplus_1$ and $\oplus_2$ operators for tau-comp programs presented in Theorem 5. □

The **BK-4** property is not satisfied in general. Consider the following example inspired from (Banti *et al.* 2003).

Example 2

Let P_1 be:
day ← ¬night.
night ← ¬day.
see-stars ← night, ¬cloudy.
∼see-stars.

The only answer set of this program is {∼see-stars, day}. Now, suppose that P_1 is updated with:

Let P_2 be:
see-stars ← see-venus.
see-venus ← see-stars.

Applying the $\oplus_2$ operator the answer sets of $P_1 \oplus_2 P_2$ are: {see-stars, see-venus, night}, {∼see-stars, day}, and {∼see-stars, night}, while $P_1 \cup P_2$ has the unique answer set {∼see-stars, day}. As we can see, the update process results in additional answer sets, so **BK-4** doesn't hold.

This proposal satisfies the initialization, idempotence, and augmented update properties. The initialization property follows directly by construction. The augmented update property is satisfied because $P_1 \oplus_1 P_2$ is equivalent in $\mathbf{N_2}$ to P_2 whenever $P_1 \subseteq P_2$. Since the idempotence property is actually a special case of the augmented update property, it holds also. As the following example shows, this new update operator ($\oplus_2$) doesn't satisfy the noninterference property.

Example 3
This example shows that the noninterference property is not satisfied by the $\oplus_2$ operator.

Let P_1 be:
day ← ¬night.
night ← ¬day.
see-stars ← night, ¬cloudy.
∼see-stars.

Let P_2 be:
∼tv-on ← power-failure.
power-failure.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

If we update P_1 with P_2 we obtain the following results:

$P_1 \oplus_2 P_2 \neq P_2 \oplus_2 P_1$, because $P_2 \oplus_2 P_1$ has only one answer set, while $P_1 \oplus_2 P_2$ adds more models. Thus, the update operator $\oplus_2$ doesn't satisfy the noninterference property.

With a slight modification in the definition of $\oplus_2$, this property can be satisfied. We simply need to add the weakly negated literals only to those rules r in program P_1 for which there is a conflictive rule r' in P_2, i.e., rules for which $H(r) = \sim H(r')$. We denote this modified update operator by $\oplus'_2$. However, this results in the loss of the **BK-6** property, as the following example shows.

Example 4
Consider the following programs.

Let P be:	$a \leftarrow \neg \sim a.$	Let P_1 be:	$a \leftarrow a.$	Let P_2 be:	$b.$
	$\sim a.$		$b.$		

Here P_1 and P_2 are equivalent in $\mathbf{N_2}$. For the update program $P \oplus'_2 P_1$ we obtain the answer sets $\{a, b\}$ and $\{\sim a, b\}$, while $P \oplus'_2 P_2$ has the unique answer set $\{\sim a, b\}$, therefore **BK-6** (WIS) fails. In addition, the **BK-4** property is still not satisfied, as applying the $\oplus'_2$ operator to the update sequence of Example 2 would show. In the next section we present Example 6, where $\oplus'_2$ also yields unintended answer sets.

5 An alternative definition for updates

In this section we introduce an alternative definition for updates as defined by the $\oplus_1$ operator, prove that it is equivalent for single updates (i.e., update sequences consisting of two programs), and analyze its properties.

Definition 6
Given an update sequence $\boldsymbol{P} = (P_1, P_2)$ over a set of atoms $\mathscr{A}$, we define the update program $\boldsymbol{P}_{\oplus_3} = P_1 \oplus_3 P_2$ over $\mathscr{A}^*$ consisting of the following items:

(i) all constraints in $P_1 \cup P_2$;
(ii) for each $r \in P_1$;
$\quad L \leftarrow B(r), \neg \boldsymbol{sup}(comp(L), P_2).$ if $H(r) = L$;
(iii) all rules $r \in P_2$.

where

$comp(L) = \sim L.$
$comp(\sim L) = L.$

Also, for a program P and a literal L let

$$K_L = \{\ B(r) \mid H(r) = L, r \in P\ \}$$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

and

$$\textbf{\textit{sup}}(L,P) = \begin{cases} \bigvee K_L & \text{if } K_L \neq \emptyset,\ \top \notin K_L. \\ \bot & \text{if } \neg\exists\ r \in P \mid H(r) = L. \\ \top & \text{if } \exists\ r \in P \mid B(r) = \top,\ H(r) = L. \end{cases}$$

Note that ***sup***(L,P) essentially takes a disjunction of the bodies of the rules that have L in their head, or evaluates to $\bot$ in case L does not appear in any of the heads of the rules of program P. Also note that ***sup***(L,P) evaluates to $\top$ if L appears as a fact in P, i.e., a rule of the form $L \leftarrow \top$.

Example 5
We apply this definition to the update sequence presented in Example 1. If we update P_1 with P_2 we obtain the following program $P_1 \oplus_3 P_2$:

sleep ← night, ¬watch-tv, ¬other.
night.
tv-on ← ¬tv-broke, ¬(power-failure ∨ (assignment-due ∧ working))
watch-tv ← tv-on.
∼tv-on ← power-failure.
∼tv-on ← assignment-due, working.
assignment-due.
working.
other ← working.

This program has the same unique answer set that was obtained in Example 1, namely {night, other, assignment-due, working, ∼tv-on}.

It is important to consider that the updated program is no longer an extended logic program. The following lemma shows that our approach is equivalent to the $\oplus_1$ operator.

Theorem 7
For any extended logic programs P_1 and P_2, $P_1 \oplus_1 P_2 \equiv P_1 \oplus_3 P_2$.

Proof
We apply a series of transformations to $P_1 \oplus_1 P_2$ with respect to answer sets over the $\mathscr{A}$ signature, to obtain $P_1 \oplus_3 P_2$. At some point we may delete rules involving a particular literal that we regard as temporary, i. e., a literal that does not form part of the $\mathscr{A}$ signature of the programs in the update sequence. This is due to the fact that the program that contains such rules is a conservative extension of the program that excludes them.

Note: in our proof we will apply the following transformation several times. For a subset of rules $\{L \leftarrow \alpha_1, \ldots, L \leftarrow \alpha_n\}$ of a program P, where L denotes a particular literal and each α_i an arbitrary formula representing the body of its respective rule, we join the rules of the subset into a single rule of the form $L \leftarrow \alpha_1 \vee \cdots \vee \alpha_n$ by

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

taking the disjunction of the bodies of all the rules. Thus, such subset of rules can be replaced by a single rule in program P, as long as all the rules that have L as its head have been taken into consideration. This transformation can also be applied inversely to decompose a rule.

Considering a particular literal L, the rules of $P_1 \oplus_1 P_2$ are of the form

$$\begin{array}{ll} L_1 \leftarrow B(r), \neg rej(r) & * \\ rej(r) \leftarrow B(r), \sim L_2 & \\ L_2 \leftarrow B(r') & * \\ L_1 \leftarrow L_2 & \\ L \leftarrow L_1 & \end{array}$$

where r represents a rule from program P_1, r' a rule from program P_2, and * denotes the possible existence of multiple rules of the same form (i.e. rules with the same head but different bodies). The first two types of rules come from Point (*ii*) of Definition 2, while the third comes from Point (*iii*) and the last two from Point (*iv*). Constraints are omitted since they undergo no changes. First, we apply the *unfolding* transformation (see Baral (2003)). The rule $L_1 \leftarrow L_2$ is unfolded with respect to the rules $L_2 \leftarrow B(r')$, i.e., the rule $L_1 \leftarrow L_2$ is replaced by n rules of the form $L_1 \leftarrow B_1(r')$; . . . ; $L_1 \leftarrow B_n(r')$ resulting from each of the n rules $L_2 \leftarrow B_1(r')$; . . . ; $L_2 \leftarrow B_n(r')$. Thus the following equivalent set of rules is derived

$$\begin{array}{ll} L_1 \leftarrow B(r), \neg rej(r) & * \\ rej(r) \leftarrow B(r), \sim L_2 & \\ L_2 \leftarrow B(r') & * \\ L_1 \leftarrow B(r') & * \\ L \leftarrow L_1 & \end{array}$$

By Theorem 3 the answer sets of the program are the same if we replace the rule $rej(r) \leftarrow B(r), \sim L_2$ by $rej(r) \leftrightarrow (B(r) \wedge \sim L_2)$. Then we can replace $rej(r)$ in all rules $L_1 \leftarrow B(r) \wedge \neg rej(r)$ to obtain rules of the form

$$L_1 \leftarrow B(r) \wedge \neg(B(r) \wedge \sim L_2) \qquad (1)$$

and afterwards we can delete the rule $rej(r) \leftrightarrow (B(r) \wedge \sim L_2)$ since $rej(r)$ is a temporary literal. The rule of the form (1) can be replaced by $L_1 \leftarrow B(r) \wedge \neg \sim L_2$ following the next transformation. First, by applying the De Morgan law we obtain

$$L_1 \leftarrow B(r) \wedge (\neg B(r) \vee \neg \sim L_2)$$

Then, by the distributive property the previous rule becomes

$$L_1 \leftarrow (B(r) \wedge \neg B(r)) \vee (B(r) \wedge \neg \sim L_2)$$

This rule can in turn be divided into two rules

$$\begin{array}{ll} L_1 \leftarrow (B(r) \wedge \neg B(r)) & (a) \\ L_1 \leftarrow (B(r) \wedge \neg \sim L_2) & (b) \end{array}$$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Rule (a) is a tautology and can therefore be eliminated. In this way we arrive to the following set of rules

$$\begin{array}{ll} L_1 \leftarrow B(r), \neg \sim L_2 & * \\ L_2 \leftarrow B(r') & * \\ L_1 \leftarrow B(r') & * \\ L \leftarrow L_1 & \end{array}$$

Now we join all the rules that contain $\sim L_2$ in the head into a single rule

$$\sim L_2 \leftarrow B_1(r'') \vee B_2(r'') \vee \ldots \vee B_n(r'') \qquad \text{(c)}$$

where r'' represents a rule of program P_2. By Theorem 3 this allows us to replace $\sim L_2$ in rules (b) to obtain rules of the form

$$L_1 \leftarrow B(r), \neg(B_1(r'') \vee B_2(r'') \vee \ldots \vee B_n(r'')) \qquad \text{(d)}$$

Note that rule (d) has the form of Point (*ii*) of our definition and that rule (c) can be eliminated since $\sim L_2$ is a temporary literal. Now we join the bodies of all rules (d) and $L_1 \leftarrow B(r')$ into a single rule $L_1 \leftarrow \alpha$ where α abbreviates the composed body. Similarly, we also join the bodies of all the rules $L_2 \leftarrow B(r')$ to obtain the rule $L_2 \leftarrow \beta$. Thus we derive a new set of rules

$$\begin{array}{l} L_1 \leftarrow \alpha \\ L_2 \leftarrow \beta \\ L \leftarrow L_1 \end{array}$$

Then we can delete the rule $L_2 \leftarrow \beta$ since L_2 is a temporary literal. Next we replace L_1 in the third rule for α generating the rule $L \leftarrow \alpha$. At this point the rule $L_1 \leftarrow \alpha$ can be eliminated because L_1 is a temporary literal. By decomposing the remaining rule $L \leftarrow \alpha$ we can generate the rules

$$\begin{array}{ll} L \leftarrow B(r), \neg(B_1(r'') \vee B_2(r'') \vee \ldots \vee B_n(r'')) & * \\ L \leftarrow B(r') & * \end{array}$$

The first rule represents the set of rules of the Point (*ii*) of Definition 6, while the second rule represents the set of rules created with Point (*iii*). We can apply this process analogously for the rest of the literals. After this sequence of transformations we obtain the same set of rules as by applying our definition, thus the two operators are equivalent. □

Initially, it may appear that the $\oplus_3$ operator could be equivalent to the $\oplus'_2$ operator discussed in Section 4. However, as the following example shows, this is not the case.

Example 6

Consider the following programs.

Let P_1 be:	$a \leftarrow \neg \sim a.$	Let P_2 be:	$a \leftarrow c.$
	$\sim a.$		$b.$
	$\sim c.$		

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

The updated program $P_1 \oplus_2' P_2$ has the answer sets $\{\sim a, b, \sim c\}$ and $\{a, b, \sim c\}$ while $P_1 \oplus_3 P_2$ has the unique answer set $\{\sim a, b, \sim c\}$. For this particular example, we can say that the $\oplus_3$ operator incorporates the new knowledge more accurately, since a should not be derived if it depends on c, which is not true in the first place. Nevertheless, for the cases in which there are no conflicting literals in the first of the programs, the operators in question would be equivalent.

Next we will discuss how this definition of update programs relates to the **BK-ASP** properties presented earlier.

Definition 7

A program P is tau-free w.r.t. a signature $\mathscr{A}$ if no rule of the form $l \leftarrow l, \alpha$ belongs to P, where α is a formula which could be empty and l a literal over $\mathscr{A}$. By abuse of notation we define a tau-free rule as a rule that does not follow the structure just described.

Theorem 8

The update operator $\oplus_3$ satisfies the **BK-ASP** properties **BK-1**, **BK-2**, **BK-3** and **BK-5** for any extended logic programs P, P_1, and P_2 as well as **BK-6** if P_1 and P_2 are tau-free ELPs.

Proof

The first four properties follow directly by construction. The proof that **BK-6** holds for tau-free ELPs for the operator $\oplus_1$ is presented in Lemma 7 of Appendix A. Because the operator $\oplus_3$ is equivalent to the operator $\oplus_1$ that result applies as well. □

We now make use of the examples of Section 4 to demonstrate that the properties **BK-4** and **BK-6** are not satisfied in general by the update operator $\oplus_3$ and equivalently by $\oplus_1$.

First, we apply the operator $\oplus_1$ to the update sequence in Example 2. The answer sets of $P_1 \oplus_1 P_2$ are: {see-stars, see-venus, night} and {∼see-stars, day}, while $P_1 \cup P_2$ has the unique answer set {∼see-stars, day}. As we can see, the update process adds a second answer set, so **BK-4** doesn't hold.

Now, when we consider the update sequences of Example 4 we obtain the following results. The programs P_1 and P_2 are equivalent in $\mathbf{N_2}$. For the update program $P \oplus_1 P_1$ we obtain the answer sets $\{a, b\}$ and $\{\sim a, b\}$, while $P \oplus_1 P_2$ has the unique answer set $\{\sim a, b\}$, therefore **BK-6** (WIS) fails.

In order to resolve the previous discrepancies, the update semantics can be refined in the following way

$$P_1 \oplus_3' P_2 = \begin{cases} P_1 \cup P_2 & \text{if } P_1 \cup P_2 \text{ has answer sets} \\ P_1 \oplus_3 P_2' & \text{otherwise} \end{cases}$$

where P_2' is P_2 without tautologies. In this way all of the **BK-ASP** properties we have presented are satisfied.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Theorem 9
The update operator $\oplus'_3$ satisfies the **BK-ASP** properties **BK-1** to **BK-6** for any extended logic programs.

Proof
Because of the refinement just introduced, property **BK-4** holds and the rest of the first five properties follow directly by construction. We prove that **BK-6** holds for the operator $\oplus'_3$ as follows. For the ELPs P_1 and P_2 we assume $P_1 \equiv_{N_2} P_2$. Now if $P \cup P_1$ has answer sets, by Theorem 2 these are the same answer sets of $P \cup P_2$, which is $P \oplus'_3 P_2$ according to our definition. If $P \cup P_1$ does not have answer sets, then Lemma 7 (see Appendix A) applies, since tautologies are removed from the right operand, i.e., the program that introduces the new rules. □

The previous theorem demonstrates that in principle it is possible to define a semantics that satisfies all of the **BK-ASP** properties, which is important from a mathematical viewpoint. However, this semantics is not free of counterintuitive and undesirable behaviors, as was brought to our attention by one of our anonymous reviewers with the following examples (adapted to include strong negation).

Example 7
Consider the following programs.

Let P be: see-stars.
∼see-stars.

Let P_1 be: see-stars ← see-stars.

Let P_2 be: see-stars ← see-venus.
see-venus ← see-stars.

The program $P \oplus'_3 P_1$ does not have answer sets, as would normally be expected. However, the program $P \oplus'_3 P_2$ has the single answer set {see-stars, see-venus}, which intuitively should not have been derived from the new information in P_2.

Example 8
Consider the following programs.

Let P_1 be: open-school.
holiday ← ¬workday.

Let P_2 be: ∼open-school ← holiday.
workday ← ¬holiday.

Let P_3 be: see-stars.

Let P_4 be: ∼see-stars.

Program $P_1 \oplus'_3 P_2$ has the single answer set {open-school, workday}, which is the same single answer set of $P_1 \cup P_2$. However, program $(P_1 \cup P_3) \oplus'_3 (P_2 \cup P_4)$ has two answer sets, namely {open-school, workday, ∼see-stars} and {holiday, ∼open-school, ∼see-stars}. The addition of rules affects the interpretation of other rules, despite having disjoint alphabets. Therefore the $\oplus'_3$ operator violates the general principle that completely independent parts of a program should not interfere with each other.

The existence of such undesirable behaviors suggests the need of analyzing alternative semantics for logic program updates, considering for instance the semantics presented in Banti *et al.* (2005) and (Homola 2005). We consider this to be an interesting direction for future work.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

6 Conclusions

We have presented a set of properties based on $\mathbf{N_2}$ logic that provide a reinterpretation the AGM postulates into the context of answer set programming. These properties hold for update programs following the definition presented in (Eiter *et al.* 2002) after an unsubstantial modification of its semantics. In addition, we introduced an alternative definition for update programs that is equivalent, yet more concise and intuitive. There are many aspects to consider in the theory of belief revision and update. The desired behavior in a particular situation may prove inappropriate in a different setting. It is essential to characterize the properties that ought to be respected in the process of belief change, however, it is unlikely that such characterization can be reduced to a set of generally accepted and applicable postulates. We can also conclude that the approach based on $\mathbf{N_2}$ logic for the characterization of answer sets has proven useful to obtain a better understanding of the answer set semantics, as well as to study many of its interesting properties.

We emphasize that with the properties presented in this paper we aim only to provide a starting point for the analysis of belief revision under answer set semantics. Still more work needs to be done to identify new interesting properties and define alternative operators for belief change.

Acknowledgements

We would like to thank Vladimir Lifschitz for his remark regarding Lemma 1.

Appendix A Analysis of the WIS property

We prove that the operator $\oplus_1$ satisfies the Weak Independence of Syntax property for tau-free programs.

Definition 8
(Eiter *et al.* 2002) Let us call two rules r_1 and r_2 conflicting iff $H(r_1) = \sim H(r_2)$, we denote this by $r_1 \bowtie r_2$.

Definition 9
(Eiter *et al.* 2002) For an update sequence $\boldsymbol{P} = (P1, \ldots, P_n)$ over a set of atoms $\mathscr{A}$ and $S \subseteq Lit_{\mathscr{A}}$ based on the principle of founded rule rejection, we define the rejection set of S by $Rej(S, \boldsymbol{P}) = \cup_{i=1}^{n} Rej_i(S, \boldsymbol{P})$, where $Rej_n(S, \boldsymbol{P}) = \emptyset$ and, for $n > i \geqslant 1$,

$$Rej_i(S, \boldsymbol{P}) = \{r \in P_i \mid \exists r' \in P_j \setminus Rej_j(S, \boldsymbol{P}), \text{ for some } j \in \{i+1, \ldots, n\} \text{ such that } r \bowtie r' \text{ and } \overline{S} \vdash_{N_2} B(r) \wedge B(r')\}.$$

Definition 10
(Eiter *et al.* 2002) For an update sequence $\boldsymbol{P} = (P_1, \ldots, P_n)$ over a set of atoms $\mathscr{A}$ and $S \subseteq Lit_{\mathscr{A}}$, let us define

$$Rej'(S, \boldsymbol{P}) = \cup_{i=1}^{n} \{r \in P_i \mid \exists r' \in P_j, \text{ for some } j \in \{i+1, \ldots, n\} \text{ such that } r \bowtie r' \text{ and } \overline{S} \vdash_{N_2} B(r) \wedge B(r')\}.$$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Note a: Observe that Rej and Rej' coincide for two programs. Furthermore $Rej'(S,(P_1,P_2)) = \{r \in P_1 \mid \exists r' \in P_2$, such that $r \bowtie r'$, and $\overline{S} \vdash_{N_2} B(r) \wedge B(r')\}$. So Rej' only contains clauses from P_1 and not from P_2, i.e., only rules from P_1 are rejected.

We denote by $\cup \boldsymbol{P}$ the set of all rules occurring in $\boldsymbol{P}$, i.e., $\cup \boldsymbol{P} = \cup_{i=1}^{n} P_i$. By P^S we denote the standard reduct of a program P with respect to a set of literals S.

Lemma 2
Let $\boldsymbol{P} = (P_1, P_2)$ be an update sequence over a set of atoms $\mathscr{A}$ and $S \subseteq Lit_{\mathscr{A}}$ a set of literals. Then, S is an answer set of $\boldsymbol{P}$ iff S is an answer set of $(P_1 \backslash Rej'(S, \boldsymbol{P})) \cup P_2$.

Proof
S is an answer set of $\boldsymbol{P}$ iff
S is the minimal model of $(\cup \boldsymbol{P} \setminus Rej(S, \boldsymbol{P}))^S$ (by Theorem 4 given in Eiter *et al.* (2002)) iff
S is an answer set of $(P_1 \cup P_2) \setminus Rej(S, \boldsymbol{P})$ iff
S is an answer set of $(P_1 \setminus Rej'(S, \boldsymbol{P})) \cup P_2$ (by Note a) as desired. □

Lemma 3
Let P be an extended logic program, x a literal, and M a set of literals. If $P \vdash_{N_2} x$ and $\overline{M} \vdash_{N_2} P$, then $\exists\, r \in P$ such that r has the form $x \leftarrow \beta$ and $\overline{M} \vdash_{N_2} \beta$.

Proof
Due to the translation from Nelson logics to intermediate logics, we may suppose to be in $\mathbf{G_3}$. The translation adds more rules to the programs but these are constraints, so the result is not affected. Additionally, this allows us to replace the literal x by an atom a. Assume by contradiction that $\forall r \in P$ such that r has the form $a \leftarrow \beta$, $\overline{M} \vdash_{N_2} \beta$ is false. We need to prove that $P \vdash_{N_2} a$ is false. Let us construct an interpretation I based on $\overline{M}$. If an atom belongs to $\overline{M}$ we assign to it the value 2. If the negation of an atom belongs to $\overline{M}$ we assign to it the value 0. Since $\overline{M} \vdash_{N_2} P$ then clearly I models P. Now we construct another interpretation I' based on I as follows. All atoms but a are assigned the same value as in I, while a is assigned the value 1. It is easy but tedious to check that I' also models P. However, I' does not model a, as desired. □

Lemma 4
Let P_1, P_2, $\{c\}$, and $\{r\}$ be extended logic programs where $r \in P_1$ and c is a tau-free rule. Let M be a set of literals. Suppose $P_2 \vdash_{N_2} c$, $\overline{M} \vdash_{N_2} B(r) \wedge B(c)$, and $\overline{M} \vdash_{N_2} P_2$. Also, r and c are conflicting rules. Then $\exists\, r' \in P_2$ such that r' is a conflicting rule with r and $\overline{M} \vdash_{N_2} B(r')$.

Proof
Let c and r be formulas of the form $x \leftarrow \theta$ and $\sim x \leftarrow \beta$, respectively. We assume that $r \in P_1$, $\overline{M} \vdash_{N_2} \beta$, and that c and r are conflicting rules. Furthermore, we assume $\overline{M} \vdash_{N_2} P_2 \cup \{\theta\}$. Also by hypothesis $P_2 \vdash_{N_2} \theta \rightarrow x$, therefore $P_2 \cup \{\theta\} \vdash_{N_2} x$. Then by Lemma 3, $\exists\, r' \in P_2 \cup \{\theta\}$ such that r' is of the form $x \leftarrow B(r')$ and $\overline{M} \vdash_{N_2} B(r')$. Moreover $r' \in P_2$, due to the restriction that c is a tau-free rule and consequently cannot contain x in its body. □

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Lemma 5

Let P_1, P_2, and $\{c\}$ be extended logic programs where $\{c\}$ is tau-free. Suppose $\overline{S} \vdash_{N_2} P_2$ and $P_2 \vdash_{N_2} c$. Then $Rej'(S,(P_1,P_2)) = Rej'(S,(P_1,P_2 \cup \{c\})$.

Proof

We need to prove two cases:

Case 1) $Rej'(S,(P_1,P_2)) \subseteq Rej'(S,(P_1,P_2 \cup \{c\})$. Clearly holds.

Case 2) $Rej'(S,(P_1,P_2 \cup \{c\}) \subseteq Rej'(S,(P_1,P_2))$

Let $r \in Rej'(S,(P_1,P_2 \cup \{c\}))$ then $\exists r' \in (P_2 \cup \{c\})$ such that $\overline{S} \vdash_{N_2} B(r) \wedge B(r')$. Here we have two cases:

a) If $r' \in P_2$ then $r \in Rej'(S,(P_1,P_2))$ as desired.

b) If $r' = c$, we have $r \in P_1$, $P_2 \vdash_{N_2} c$, $\overline{S} \vdash_{N_2} B(r) \wedge B(c)$, r and c are conflicting rules, and by hypothesis $\overline{S} \vdash_{N_2} P_2$. Then by Lemma 4, $\exists r$" $\in P_2$ such that r and r" are conflicting rules and $\overline{S} \vdash_{N_2} B(r) \wedge B(r$"). Hence, $r \in Rej'(S,(P_1,P_2))$ as desired. □

Note b: $P' \vdash_{N_2} c$ implies $P' \equiv_{N_2} P' \cup \{c\}$.

Lemma 6

Let P_1, P_2, and $\{c\}$ be extended logic programs where $\{c\}$ is tau-free. If $P_2 \vdash_{N_2} c$ then $P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup \{c\})$.

Proof

S is an answer set of $P_1 \oplus_1 P_2$ iff	by Lemma 2
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2))) \cup P_2$ iff	by Lemma 5
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2 \cup \{c\}))) \cup P_2$ iff	by Note b
S is an answer set of $(P_1 \setminus Rej'(S,(P_1,P_2 \cup \{c\}))) \cup (P_2 \cup \{c\})$ iff	by Lemma 2
S is an answer set of $P_1 \oplus_1 (P_2 \cup \{c\})$	

Hence, $P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup \{c\})$ as desired. □

Now we can present our main result.

Lemma 7

Let P_1, P_2, and R be extended logic programs where R is tau-free. Suppose that $P_2 \vdash_{N_2} R$, then $P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup R)$.

Proof

By induction on the size of R.

Base case: Let $R = \emptyset$, the result is immediate.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Induction step: Let P_2, R, and $\{c\}$ be extended logic programs where R and $\{c\}$ are tau-free. We need to show:

$$\text{if } P_2 \vdash_{N_2} R \cup \{c\} \quad \text{then} \quad P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup (R \cup \{c\})).$$

But we know that $P_2 \vdash_{N_2} R \cup \{c\}$ means that $P_2 \vdash_{N_2} R$ and $P_2 \vdash_{N_2} c$, then by applying the induction hypothesis

$$P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup R) \quad \text{(I)}$$

By Lemma 6, if $P_2 \cup R \vdash_{N_2} c$ then $P_1 \oplus_1 (P_2 \cup R) \equiv P_1 \oplus_1 ((P_2 \cup R) \cup \{c\})$ (II)

Now, from (I) and (II) we have $P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 ((P_2 \cup R) \cup \{c\})$

Since $P_1 \oplus_1 ((P_2 \cup R) \cup \{c\}) \equiv P_1 \oplus_1 (P_2 \cup (R \cup \{c\}))$

we obtain $P_1 \oplus_1 P_2 \equiv P_1 \oplus_1 (P_2 \cup (R \cup \{c\}))$ as desired. □

We have proven that the **BK-0** property holds for the $\oplus_1$ operator if R is tau-free. If we apply the proof of Theorem 4 of Section 3 under this restriction, this results in the fact that **BK-6** holds provided P_1 and P_2 are tau-free.

Appendix B Proof of Lemma 1

Recall that an alternative proof is presented in Ferraris and Lifschitz (2005).

Let P be a program over a set of atoms $\mathscr{A}$ and x a literal that does not occur in P. Let F be a formula over a set of atoms $\mathscr{F}$ such that $Lit_{\mathscr{F}} \subseteq Lit_{\mathscr{A}}$. Then $P \cup \{F \leftarrow x\} \equiv P \cup \{x \leftrightarrow F\}$.

If we apply the definition of an answer set given in Theorem 1, then we only need to prove

$$P, F \rightarrow x, \neg\widetilde{M} \cup \neg\neg M \Vdash_{N_2} M \quad \text{iff}$$
$$P, F \leftrightarrow x, \neg\widetilde{M} \cup \neg\neg M \Vdash_{N_2} M$$

First, we apply the transformation described in Section 2.2 to reduce the previous statement to provability in $\mathbf{G_3}$, obtaining

$$P', F' \rightarrow x, \neg(x' \wedge x), C', \neg\widetilde{M} \cup \neg\neg M \Vdash_{G_3} M \quad \text{iff}$$
$$P', F' \leftrightarrow x, \neg(x' \wedge x), C', \neg\widetilde{M} \cup \neg\neg M \Vdash_{G_3} M$$

where P' represents program P after the change of literals $\sim l$ by atoms l', F' the formula F after the same changes, and C' the set of added constraints (excluding the constraint involving x). Now we have two cases.

I) $x \notin M$ so $\neg x \in \neg\widetilde{M}$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

In this case the statement to prove becomes

$$P', F' \to x, \neg x, \neg(x' \wedge x), C', \neg\widetilde{M'} \cup \neg\neg M \Vdash_{G_3} M \qquad \text{iff}$$
$$P', F' \leftrightarrow x, \neg x, \neg(x' \wedge x), C', \neg\widetilde{M'} \cup \neg\neg M \Vdash_{G_3} M$$

where $\widetilde{M'}$ denotes the set $\widetilde{M'}$ excluding the atom x. It is easy to show that $F' \to x, \neg x \equiv_{G_3} F' \leftrightarrow x, \neg x$ so the statement holds for both implications.

II) $x \in M$ so $\neg\neg x \in \neg\neg M$

In this case the statement to prove becomes

$$P', F' \to x, \neg\neg x, \neg(x' \wedge x), C', \neg\widetilde{M} \cup \neg\neg M' \Vdash_{G_3} M \qquad \text{iff}$$
$$P', F' \leftrightarrow x, \neg\neg x, \neg(x' \wedge x), C', \neg\widetilde{M} \cup \neg\neg M' \Vdash_{G_3} M$$

where M' denotes the set M' excluding the atom x. It is easy to show that this statement can be reduced to

$$P', F' \to x, \neg\neg x, \neg x', C', \neg\widetilde{M} \cup \neg\neg M' \Vdash_{G_3} M \qquad \text{iff}$$
$$P', F' \leftrightarrow x, \neg\neg x, \neg x', C', \neg\widetilde{M} \cup \neg\neg M' \Vdash_{G_3} M$$

We will abbreviate the statement above in the following way

$$F' \to x, \neg\neg x, H \Vdash_{G_3} M \qquad \text{iff}$$
$$F' \leftrightarrow x, \neg\neg x, H \Vdash_{G_3} M$$

It is important to remark that x does not occur in H or in F'. For consistency, the right implication clearly holds. We prove the left implication as follows. We assume $F' \to x, \neg\neg x, H$ is consistent and complete. By the results obtained in (Osorio *et al.* 2004; Osorio *et al.* 2001) $F' \to x, H$ is also consistent and complete. So there is an interpretation I such that $I(H) = 2$. We construct a new interpretation I' that differs from I only in that $I'(x) = I(F)$, so I' models $F' \leftrightarrow x, H$. Therefore $F' \leftrightarrow x, H$ is consistent. Since by hypothesis $F' \to x, \neg\neg x, H \Vdash_{G_3} x$ (because $x \in M$) then clearly $F' \to x, H \Vdash_{G_3} x$. So $F' \leftrightarrow x, \neg\neg x, H$ is consistent.

In respect to logical consequence, the right implication clearly holds. We will prove the left implication for every atom $a \in M$. Again from the results obtained in Osorio *et al.* (2004, 2001) we can replace $F' \leftrightarrow x, \neg\neg x, H$ by $F' \leftrightarrow x, H$ in our proof, so we only need to prove

$$\text{if } F' \leftrightarrow x, H \vdash_{G_3} a \text{ then } F' \to x, H \vdash_{G_3} a$$

We will prove its contrapositive, i.e.

$$\text{if } \exists I \; I((F' \leftrightarrow x) \wedge H) > I(a) \text{ then } \exists I' \; I'((F' \to x) \wedge H) > I'(a)$$

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

If $a = x$ then we have several cases. If $I((F' \rightarrow x) \wedge H) = 2$ and $I(a) = I(x) = 0$ then $I(F')$ must evaluate to 0, so with $I' := I$ the statement holds. The case when $I((F' \rightarrow x) \wedge H) = 1$ and $I(x) = 0$ is analogous. For the case when $I((F' \rightarrow x) \wedge H) = 2$ and $I(x) = 1$, if $I(F') = 1$ then we define $I' := I$ as in the previous cases. Now if $I(F') = 0$ we take I' to be equal to I except for the fact that we define $I'(x) = 0$, which satisfies the desired condition. The proof for the case when $x \neq a$ is analogous but simpler, and is therefore omitted. □

References

Alchourrón, C. E., Gärdenfors, P., and 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. and Pereira, L. M. 2002. Logic programming updating - a guided approach. *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*. Springer-Verlag, pp. 382–412.

Banti, F., Alferes, J. J. and Brogi, A. 2003. A principled semantics for logic program updates. In *Nonmonotonic Reasoning, Action, and Change (NRAC'03)*, Brewka and Peppas, Eds.

Banti, F., Alferes, J. J. and Brogi, A. 2005. Operational semantics for dylps. *Progress in Artificial Intelligence: 12th Portuguese Conference on Artificial Intelligence, EPIA 2005*. Covilhã, Portugal.

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

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming 2,* 6, 711–767.

Ferraris, P. and Lifschitz, V. 2005. Weight constraints as nested expressions. *Theory and Practice of Logic Programming 5*, 45–74.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. *5th Conference on Logic Programming*, R. Kowalski and K. Bowen, Eds. MIT Press, pp. 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*,(3/4): 365–386.

Gurevich, Y. 1977. Intuitionistic logic with strong negation. *Studia Logica 36*: 49–59.

Homola, M. 2005. Dynamic logic programming: Various semantics are equal on acyclic programs. *Computational Logic in Multi-Agent Systems: 5th International Workshop, CLIMA V*, J. Leite and P. Torroni, Eds. Number 3487 in Lecture Notes in Computer Science. Springer, pp. 78–95.

Katsuno, H. and Mendelzon, A. O. 1991. Propositional knowledge base revision and minimal change. *Artificial Intelligence 52,* 3: 263–294.

Katsuno, H. and Mendelzon, A. O. 1992. On the difference between updating a knowledge base and revising it. *Belief Revision*, P. Gärdenfors, Ed. Cambridge University Press, 183–203.

Kracht, M. 1998. On extensions of intermediate logics by strong negation. *Journal of Philosophical Logic*: 49–73.

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

Lifschitz, V., Tang, L. R., and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*: 369–389.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press

Mendelson, E. 1987. *Introduction to Mathematical Logic*, Third ed. Wadsworth, Belmont, CA.

Nelson, D. 1949. Constructible falsity. *Journal of Symbolic Logic 14*: 16–26.

Ortiz, M. and Osorio, M. 2005. Nelson's strong negation, safe beliefs and the answer sets semantics. In *Answer Set Programming: Advances in Theory and Implementation ASP'05*. Bath, UK, pp. 70–84.

Osorio, M., Navarro, J. A. and Arrazola, J. 2001. Equivalence in answer set programming. In *Logic Based Program Synthesis and Transformation. 11th International Workshop, LOPSTR 2001*, A. Pettorossi, Ed. Number 2372 in Lecture Notes in Computer Science. Springer, Paphos, Cyprus, pp. 57–75.

Osorio, M., Navarro, J. A. and Arrazola, J. 2004. Applications of intuitionistic logic in answer set programming. *Theory and Practice of Logic Programming 4,* 3: 325–354.

Osorio, M., Navarro, J. A. and Arrazola, J. 2005a. Safe beliefs for propositional theories. *Annals of Pure and Applied Logic 134,* 1: 63–82.

Osorio, M., Navarro, J. A. and Arrazola, J. 2005b. Safe beliefs for propositional theories. *Annals of Pure and Applied Logic 134,* 1: 63–82.

Osorio, M. and Zacarías, F. 2003. Irrelevance of syntax in updating answer set programs. *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC'03) Workshop on Logic and Agents*. Apizaco, México.

Osorio, M. and Zacarías, F. 2004. On updates of logic programs: a properties–based approach. *Proceedings of the Third International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2004)*, D. Seipel and J. Turrul, Eds., pp. 231–241.

Pearce, D. 1999a. From here to there: Stable negation in logic programming. In *What Is Negation?*, D. M. Gabbay and H. Wansing, Eds. Kluwer Academic Publishers, Netherlands, pp. 161–181.

Pearce, D. 1999b. Stable inference as intuitionistic validity. *Logic Programming 38*: 79–91.

Rasiowa, H. 1974. *An Algebraic Approach to Non-Classical Logics*. Elsevier.

van Dalen, D. 1986. Intuitionistic logic handbook of philosophical logic. In *Alternatives to Classical Logic*, D. Gabbay and F. Guenthner, Eds. Vol. 3. D. Reidel.

Vorob'ev, N. 1952a. Constructive propositional calculus with strong negation. *Doklady Akademii Nauk SSSR 85*, pp. 465–468. In Russian.

Vorob'ev, N. 1952b. The problem of deducibility in constructive propositional calculus with strong negation. *Doklady Akademii Nauk SSSR 85*, pp. 689–692. In Russian.

https://doi.org/10.1017/S1471068406003000 Published online by Cambridge University Press