Update Operation in ASP Revisited

J. C. Acosta Guadarrama
Impressum

Publisher: Institut für Informatik, Technische Universität Clausthal
Julius-Albert Str. 4, 38678 Clausthal-Zellerfeld, Germany
Editor of the series: Jürgen Dix
Technical editor: Wojciech Jamroga
Contact: wjamroga@in.tu-clausthal.de
URL: http://www.in.tu-clausthal.de/forschung/technical-reports/
ISSN: 1860-8477

The IfI Review Board

Prof. Dr. Jürgen Dix (Theoretical Computer Science/Computational Intelligence)
Prof. Dr. Klaus Ecker (Applied Computer Science)
Prof. Dr. Barbara Hammer (Theoretical Foundations of Computer Science)
Prof. Dr. Kai Hormann (Computer Graphics)
Prof. Dr. Gerhard R. Joubert (Practical Computer Science)
apl. Prof. Dr. Günter Kemnitz (Hardware and Robotics)
Prof. Dr. Ingbert Kupka (Theoretical Computer Science)
Prof. Dr. Wilfried Lex (Mathematical Foundations of Computer Science)
Prof. Dr. Jörg Müller (Economical Computer Science)
Prof. Dr. Niels Pinkwart (Economical Computer Science)
Prof. Dr. Andreas Rausch (Software Systems Engineering)
apl. Prof. Dr. Matthias Reuter (Modeling and Simulation)
Prof. Dr. Harald Richter (Technical Computer Science)
Prof. Dr. Gabriel Zachmann (Computer Graphics)
Update Operation in ASP Revisited

J. C. Acosta Guadarrama

Institute of Computer Science,
TU-Clausthal, Germany
jguadarrama@gmail.com

Abstract

Revising and updating knowledge bases is an important topic in knowledge representation and reasoning. Various proposals have been made to update logic programs, in particular with respect to Answer Set Programming. Most of these approaches are based on a causal rejection principle, which leads to counterintuitive behaviour. In order to overcome such behaviour, this report presents a collection of structural properties for updates and an operator to satisfy them. Some of the properties include Weak Irrelevance of Syntax and Strong Consistency, and the report shows an example comparing a well-known operator from Eiter et al. Finally, this report presents an alternative to compute updates by means of Preferred Answer Sets.

1 Introduction

A traditional and general goal of belief updates is dealing with contradictory information or with new data. However, there are particular rare challenging (even so possible) situations that might lead to counterintuitive models of the environment that have been subject of recent research and matter of formulation of new principles. As a result, various proposals have been made to update logic programs, in particular with respect to Answer Set Programming, like [EFST02, IS04, ZF05, EFST05, ABBL05]. Some of them, however, are based on the causal rejection principle [ABBL05] that leads to unintuitive behaviour under certain circumstances [ABBL05].

As a result, this report presents a set of properties and a basic alternative semantics to overcome those deficiencies in current approaches explored in [Guad07]. The general goal in this framework to realise those objectives is to depend upon the logical contents of programs rather than on their syntax, based on a formalism of Generalised Answer Sets by [KM90, BG03]. Such a formalism shall prove to be a good foundation to characterise a basic framework by two main structural properties: Weak Irrelevance of Syntax [OZ04] and Strong Consistency. Finally, this report presents a comparison with the well-known upd operator due to Eiter et al. as the most representative semantics of its kind, and show that we satisfy the properties, by getting through its main flaws.
2 Problem Description

Despite several existing semantics for updates, there is still no common agreement on which is the “right” semantics to represent dynamic knowledge. Accordingly, some authors have tackled this problem by a detailed analysis, taxonomy and comparison of different known semantics: [EFST00, EFST05, ZF05]. However, other important properties are necessary to test adequacy of semantics for logic program updates. In particular, this report presents an analysis of important properties for logic program updates. Firstly, Weak Irrelevance of Syntax [OZ04], \( WIS \), suggests that, if one can update a theory \( \tau \) by \( \tau_1 \), the result should only depend upon the logical contents of \( \tau_1 \), and not on the particular syntax used to write \( \tau_1 \). Secondly, the postulate of Strong Consistency states that supplementary rules like \( \{ a \leftarrow b, b \leftarrow a \} \), should not result in any additional answer set, provided that \( a, b \) are already contained in the underlying language.

Moreover, this report recapitulates a definition for updates based on the notion of Minimal Generalised Answer Sets that satisfies the mentioned properties.

In order to illustrate this, consider the following example, inspired from [ABBL05], describing some beliefs about the sky, where differently from the original example, I use two types of negation: strong “\( \sim \)” and default “\( \neg \)”. Notice that the latter is also represented as “not” in logic programming.

**Example 1.** Let \( \Pi_1 \) be:

\[
\begin{align*}
\text{day} & \leftarrow \neg \text{night} \\
\text{night} & \leftarrow \neg \text{day} \\
\text{see}(\text{stars}) & \leftarrow \text{night} \land \neg \text{cloudy} \\
\neg \text{see}(\text{stars}) & \leftarrow \top
\end{align*}
\]

whose unique answer set is \( \{ \text{day}, \neg \text{see}(\text{stars}) \} \). Now consider the following program and update \( \Pi_1 \) with it.

\[
\begin{align*}
\text{see}(\text{stars}) & \leftarrow \text{see}(\text{constellations}) \\
\text{see}(\text{constellations}) & \leftarrow \text{see}(\text{stars})
\end{align*}
\]

It is easy to see that \( \Pi_2 \) contains only one new constant \( \text{constellations} \) and a new atom “\( \text{see}(\text{constellations}) \)” with respect to \( \Pi_1 \). Moreover, \( \text{see}(\text{constellations}) \) is considered synonym of \( \text{see}(\text{stars}) \) by the two defining rules —note there are no other rules mentioning \( \text{see}(\text{constellations}) \). Thus, this can be considered a conservative extension [ONA01] of \( \Pi_1 \): the language is extended and all answer sets should be extensions of the old ones: \( \text{see}(\text{constellations}) \) should be true in any of them if and only if \( \text{see}(\text{stars}) \) is true. However, according to [ABBL05], \( \Pi_2 \) introduces a new answer set for nearly all existing update-semantics\(^1\). That is to say,

\[
\{ \text{see}(\text{stars}), \text{see}(\text{constellations}), \text{night} \}
\]

\(^1\) Note that the semantics presented in [ABBL05] has no strong negation, and thus they use a syntax-
which is clearly counterintuitive. The reason is that although see(stars) cannot be true because of the constraint, introducing the synonym see(constellations) gives another reason for see(stars) to be true.

3 Answer Sets Programming

As a main foundation of this proposal, Answer Set Programming, ASP [GL88] is known for its logical characterisations, as well as for its suitability to represent non-monotonic knowledge in problem solutions that go from typical AI toy examples to yet-preliminary agency applications and planning. Its formal language and some more notation are as follows.

Definition 1 (ASP Language of logic programs, $L_{ASP}$). In the following $L_{ASP}$ is a language of propositional logic with propositional symbols: $a_0, a_1, \ldots$; connectives: “,” (conjunction, also denoted as “∧”), ∨ (disjunction, also denoted as |), ← (derivation), propositional constants ⊥ (falsum), ⊤ (verum), “not” (default negation), “¬” (strong negation, also denoted as “¬” or as “¬”); auxiliary symbols: “(“, |”)” (parentheses). The propositional symbols are also called atoms or atomic propositions. A literal is an atom or a strong-negated atom. A rule is an ordered pair $\text{Head}(\rho) ← \text{Body}(\rho)$ where $\text{Head}(\rho)$ is either null, or a literal, or a disjunction of literals, and $\text{Body}(\rho)$ a possibly-empty finite conjunction of literals and/or default-negated literals.

With the notation just introduced, one may construct clauses of the following general form that are well known in the literature.

Definition 2 (Extended Disjunctive Logic Program, EDLP). An extended disjunctive logic program is a set of rules of form

\[ \ell_1 \lor \ell_2 \lor \cdots \lor \ell_l ← \ell_{l+1}, \ldots, \ell_m, \text{not } \ell_{m+1}, \ldots, \text{not } \ell_n \]  

where $\ell_i$ is a literal and $0 \leq l \leq m \leq n$.

As expected, an extended logic program (or ELP hereafter) is a finite set of rules of form (1) with $l = 1$; while an integrity constraint (also known in the literature as strong constraint) is a rule of form (1) with $l = 0$. In particular, for a literal $\ell$, the complementary literal is $\neg \ell$ and vice versa; for a set $M$ of literals, $\neg M = \{ \neg \ell \mid \ell \in M \}$, and $\text{Lit}_M$ denotes the set $M \cup \neg M$; finally, a signature $L_\Pi$ is a finite set of literals occurring in $\Pi$. Additionally, given a set of atoms $A$, not $A = \{ \text{not } a \mid a \in A \}$; $\neg A = \{ \neg a \mid a \in A \}$; given another set $M \subseteq A$, the complement set $\overline{M} = A \setminus M$.

Definition 3 (Extended Logic Program, ELP). An extended logic program is a set of rules of the form

\[ \ell ← \ell_1, \ell_2, \ldots, \ell_m, \text{not } \ell_{m+1}, \ldots, \text{not } \ell_n \]  

dependent ‘default’ negation in heads! to have a similar effect in updates and to solve problems like Example 1.
where \( \ell \) is a literal and \( 0 \leq m \leq n \).

The well-known semantics of an EDLP consists of reducing general rules to rules without default negation “not” because the latter are more standard.

**Definition 4 (Answer Set Operator).** Let \( \Pi \) be a positive extended disjunctive program and \( \mathcal{L}_\Pi \) the set of all ground literals from \( \Pi \). \( \mathcal{Cn}(\Pi) \) denotes the minimal subset of \( \mathcal{L}_\Pi \) where

1. for each ground clause \( p_0 \lor p_1 \lor \cdots \lor p_l \leftarrow q_1, \ldots, q_m \in \Pi \), \( q_1, \ldots, q_n \in S \) implies \( p_i \in S \) for some \( 0 \leq i \leq l \) and for each ground clause
   \[
   \leftarrow q_1, \ldots, q_m
   \]
   \( \{q_1, \ldots, q_m\} \not\subseteq S \).

2. if \( S \) contains a pair of complementary literals, then \( S = \mathcal{L}_\Pi \).

**Definition 5 (Answer Set).** Suppose \( \Pi \) is an EDLP and \( S \) a set of literals. \( S \) is an answer set of \( \Pi \) if and only if \( S = \mathcal{Cn}(\Pi^S) \).

Although ASP is our main basis, a more flexible means is necessary to set up preferences amongst models, so that one may choose the most appropriate, according to general principles and postulates. One of such intermediate mechanisms is Abductive Logic Programming, due to [KM90], briefly presented in the following.

### 3.1 Minimal generalised Answer Sets

In this section we recapitulate some basic definitions about syntax and semantics of abductive logic programs. These semantics are given by minimal generalised answer sets (MGAS), which provide a more general and flexible semantics than standard answer sets.

**Definition 6 (Abductive Logic Program [KM90]).** An abductive logic program is a pair \( \langle \Pi, A \rangle \) where \( \Pi \) is an arbitrary program and \( A \) a set of literals, called abducibles.

**Definition 7 (Generalised Answer Sets GAS, [KM90]).** \( M(\Delta) \) is a generalised answer set of the abductive program \( \langle \Pi, A \rangle \) iff \( \Delta \subseteq A \) and \( M(\Delta) \) is an answer set of \( \Pi \cup \{ H \leftarrow \top \mid H \in \Delta \} \).

**Definition 8 (Abductive Inclusion Order [KM90]).** We can establish an ordering among generalised answer sets as follows: let \( M(\Delta_1) \) and \( M(\Delta_2) \) be generalised answer sets of \( \langle \Pi, A \rangle \), we define \( M(\Delta_1) \leq_A M(\Delta_2) \) iff \( \Delta_1 \subseteq \Delta_2 \).

**Example 2.** Let \( \{a, b\} \) be abducibles and \( \Pi = \{ a \leftarrow b, b \leftarrow a, c \leftarrow a \} \). Then \( \{a, b, c\}_{\{a\}} \) (that is, the resulting answer set \( \{a, b, c\} \) and the abducible \( \{a\} \)) is a GAS of \( \langle \Pi, \{a, b\} \rangle \), since \( \{a, b, c\} \) is an answer set of \( \Pi \cup \{a\} \), as well as \( \{a, b, c\}_{\{a, b\}} \) and \( \{\} \). Therefore, \( \{a, b, c\}_{\{a, b\}} \leq_A \{a, b, c\}_{\{a, b\}} \) and \( \{\} \). However, \( \{\} \) is the minimal GAS of \( \Pi \), as \( \{\} \) is a subset of any set.
**Definition 9** (Minimal generalised Answer Set MGAS, [BG03]). \( M(\Delta) \) is a minimal generalised answer set of \((\Pi, A)\) iff \( M(\Delta) \) is a generalised answer set of \((\Pi, A)\) and it is minimal w.r.t. abductive inclusion order.

### 4 Updating pairs of programs

Over the last few years several approaches have been defined to update logic programs in Answer Set semantics [EFST00, EFST05, ZF05, OZ04]. According to these proposals, knowledge is given by a sequence of logic programs where each element is considered an update of the previous one. Most of these works are based upon particular notions of causal rejection of rules, which enforces that, in case of conflicts between rules, more recent rules are preferred and older rules are overridden [EFST05, ZF05].

This report presents alternative solutions to the problems in [OZ04], as well as a basic semantics using a mechanism of Minimal Generalised Answer Sets MGAS for updates.

Formally, an update pair is a pair \((\Pi_1, \Pi_2)\) of logic programs, and \( \Pi \) is an update pair over \( A \) iff \( A \) represents the set of atoms occurring in \( \Pi_1 \cup \Pi_2 \).

**Definition 10** (Update Operation). Given an update pair \( \Pi = (\Pi_1, \Pi_2) \) over a set of atoms \( A \), its update program \( \Pi_\oplus = \Pi_1 \oplus \Pi_2 = (\Pi' \cup \Pi_2, A^+ \) over \( A^+ \) (extending \( A \) by new abducible atoms), where \( \Pi' \) is constructed as follows:

1. all constraints in \( \Pi_1 \)
2. for each non-constraint rule \( \rho \in \Pi_1 \) there is an abducible \( \alpha \) (a new unique atom) and the rule is replaced by \( \text{Head}(\rho) \leftarrow \text{Body}(\rho), \neg \alpha \).

where \( \oplus \) represents the update operator.

**Definition 11.** Let \( \Pi = (\Pi_1, \Pi_2) \) be an update pair over a set of atoms \( A \). Then, \( S \subseteq \text{Lit}_A \) is an update answer set of \( \Pi \) if only if \( S = S' \cap \text{Lit}_A \) for some minimal generalised answer set \( S' \) of \( \Pi \).

The following example illustrates a daily update regarding energy flaw, and it is an adaptation from the original ones in [ALP+99] and [EFST02].

---

2 Note that [ABBL05] uses a refined principle of rejection of rules to overcome some of the drawbacks pointed out in this work, but they have to make particular transformations in order to be classified in Answer Set Programming. Unfortunately, the latter it is not reflected in the class of updating programs used as a front end, and their ultimate goal is Well Founded Semantics.
Structural Properties

Example 3.

\[\Pi_1 = \{ \text{sleep} \leftarrow \neg \text{tv(on)}, \text{night} \leftarrow \top, \text{watch(tv)} \leftarrow \text{tv(on)}, \text{tv(on)} \leftarrow \top \} \]

\[\Pi_2 = \{ \neg \text{tv(on)} \leftarrow \text{power(failure)}, \text{power(failure)} \leftarrow \top \} \]

By following Eter’s operator \(\text{EFST02}\), the single answer set of \(\Pi_1 \sqcap \Pi_2\) is just as one would expect:

\[\{ \text{power(failure)}, \neg \text{tv(on)}, \text{sleep, night} \} \quad (4)\]

On the other hand, by codifying this example under \(\odot\) operator, \(\Pi_1\) is transformed as follows: for each rule in \(\Pi_1\), there is a new atom from the set of abducibles \(A^*\). Next, each abducible ought to be default-negated and appended to the body of every rule in \(\Pi_1\). As a consequence, the update program is the abductive program \(\langle \Pi' \cup \Pi_2, A^* \rangle\), where \(A^* = \{ \alpha_1, \alpha_2, \alpha_3, \alpha_4 \}\) and

\[\Pi' \cup \Pi_2 = \{ \text{sleep} \leftarrow \neg \text{tv(on)}, \neg \alpha_1, \text{night} \leftarrow \neg \alpha_2, \text{watch(tv)} \leftarrow \text{tv(on)}, \neg \alpha_3, \text{tv(on)} \leftarrow \neg \alpha_4, \neg \text{tv(on)} \leftarrow \text{power(failure)}, \text{power(failure)} \leftarrow \top \} \]

whose unique update answer set, out of the unique MGAS

\[\{ \text{night, sleep, power(failure), } \neg \text{tv(on)} \}_{\neg \alpha_1, \neg \alpha_2, \neg \alpha_3, \alpha_4} \]

coincides with \((4)\).

5 Structural Properties

As above mentioned, this report is a proposal of just a small set of fundamental properties an update semantics ought to meet, but they should serve as an initial basis of further extensions, like in upcoming papers.

Before listing these properties, the reader should note that a statement like \(\Pi_1 \equiv \Pi_2\) means that both \(\Pi_1\) and \(\Pi_2\) have the same answer sets —or alternatively \(\Pi_1 \equiv_{\text{ASP}} \Pi_2\). Pointing out an abuse of notation, when stating equivalence between updates, indeed it means that they have the same (or different) Update Answer Sets.

Moreover, another useful theorem from \(\text{LPV01}\) is also necessary.
Theorem 1 (\(N_2\) Equivalence [LPV01]). For any programs \(\Pi_1\) and \(\Pi_2\), \(\text{Trans}_{N_2}(\Pi_1) \equiv_{N_2} \text{Trans}_{N_2}(\Pi_2)\) if and only if for every program \(\Pi\), \(\Pi \cup \Pi_1\) and \(\Pi \cup \Pi_2\) have the same Answer Sets.

1. Initialisation [EFST02]: \(\emptyset \odot \Pi \equiv \Pi\).
   This property states that the update of an initial empty knowledge base yields just the update itself.

2. Inertia: If \(\Pi\) has answer sets, \(\Pi \odot \emptyset \equiv \Pi\).
   A consistent theory is in effect unless new evidence states otherwise.

3. Idempotence [EFST02]: \(\Pi \odot \Pi \equiv \Pi\).
   This property means that the update of program \(\Pi\) with itself has no effect.

4. Weak Non-interference, WNI: [EFST02]: If \(\Pi_1\) and \(\Pi_2\) are programs defined over disjoint alphabets, and either both of them have answer sets or do not, then \(\Pi_1 \odot \Pi_2 \equiv \Pi_2 \odot \Pi_1\).
   This property is a specialisation from [EFST02]'s and implies that the order of updates that do not interfere with each other, does not matter.

5. Augmented Update [EFST02]: If \(\Pi_1 \subseteq \Pi_2\) then \(\Pi_1 \odot \Pi_2 \equiv \Pi_2\).
   Updating with additional rules makes the previous update obsolete.

6. Strong Consistency, SC: If \(\Pi_1 \cup \Pi_2\) has at least one answer set, then \(\Pi_1 \odot \Pi_2 \equiv \Pi_1 \cup \Pi_2\).
   The update coincides with the union when \(\Pi_1 \cup \Pi_2\) has answer sets.

7. Weak Irrelevance of Syntax, WIS: Let \(\Pi, \Pi_1,\) and \(\Pi_2\) be logic programs under the same language. If \(\text{Trans}_{N_2}(\Pi_1) \equiv_{N_2} \text{Trans}_{N_2}(\Pi_2)\) then \(\Pi \odot \Pi_1 \equiv \Pi \odot \Pi_2\).
   It means that if we update a program \(\Pi\) with \(\Pi_1\) or with \(\Pi_2\), the result should depend upon the logical contents of \(\Pi_1\) and \(\Pi_2\), rather than the particular syntax to spell them.

Theorem 2. \(\odot\)-operator satisfies the seven properties above mentioned.

Proof. (Initialisation): \(\emptyset \odot \Pi\) has the update program \((\emptyset \cup \Pi, \emptyset)\), whose MGAS \(M_{\emptyset}\) correspond to the answer set \(M\) of \(\Pi\). Hence, \(\emptyset \odot \Pi \equiv \Pi\).

(Inertia): \(\Pi \odot \emptyset\) has the update program \((\Pi \cup \emptyset, \emptyset)\), whose MGAS \(M_{\emptyset}\) correspond to the answer sets \(M\) of \(\Pi\). Therefore, \(\Pi \odot \emptyset \equiv \Pi\).

(Strong Consistency): Assume \(M\) is an answer set of \(\Pi_1 \cup \Pi_2\). Then, \(M\) should be equal to the same model of \(\Pi_1 \odot \Pi_2\). As \(\Pi_1 \odot \Pi_2 = (\Pi' \cup \Pi_2, \mathcal{A})\), and the answer set of \(\Pi' \cup \Pi_2 \cup \{H \leftarrow \top \mid H \in \Delta\}\) is \(M_{\emptyset}(\Delta)\), then the MGAS of \((\Pi' \cup \Pi_2, \mathcal{A})\) w.r.t. its abductive sequence order should be \(M_{\emptyset}(\emptyset)\), by definition. Then we have...
. \mathcal{A} = \emptyset \text{ and } \Pi' \text{ is an ordinary extended logic program that coincides with } \Pi_1. \text{ Therefore, } \Pi_1 \otimes \Pi_2 \equiv \Pi_1 \cup \Pi_2.

(Idempotence): Suppose \Pi \text{ has answer sets. Then, } \Pi \cup \Pi \text{ does too (namely the same). Thus, by Strong Consistency } \Pi \otimes \Pi \equiv \Pi \cup \Pi \equiv \Pi. \text{ Suppose } \Pi \text{ does not have answer sets. Then } \Pi \otimes \Pi \text{ has the update program } (\Pi' \cup \Pi, \mathcal{A}) \text{ that neither does have generalised answer sets. Thus, } \Pi \equiv \Pi \otimes \Pi.

(WIS): Suppose Trans_{N_2}(\Pi_1) \equiv_{N_2} Trans_{N_2}(\Pi_2), \text{ and each } \Pi \cup \Pi_1 \text{ and } \Pi \cup \Pi_2 \text{ have at least an answer set. Then, by Strong Consistency, } \Pi \otimes \Pi_1 \equiv_{ASP} \Pi \cup \Pi_1 \text{ and } \Pi \otimes \Pi_2 \equiv_{ASP} \Pi \cup \Pi_2. \text{ Thus, } Trans_{N_2}(\Pi \cup \Pi_1) \equiv_{N_2} Trans_{N_2}(\Pi \cup \Pi_2). \text{ Therefore, if } Trans_{N_2}(\Pi_1) \equiv_{N_2} Trans_{N_2}(\Pi_2), \text{ then } \Pi \otimes \Pi_1 \equiv_{ASP} \Pi \otimes \Pi_2.

(Augmented Update): Suppose \Pi_1 \subseteq \Pi_2. \text{ This means that } \Pi_1 \cup \Pi_2 = \Pi_2. \text{ Then, by Strong Consistency, } \Pi_1 \otimes \Pi_2 = \Pi_2. \text{ Therefore, } \Pi_2 \equiv \Pi_1 \otimes \Pi_2.

(Weak Non-interference): Assume that \Pi_1 \text{ and } \Pi_2 \text{ are defined over disjoint alphabets and that both } \Pi_1 \text{ and } \Pi_2 \text{ have at least an answer set. Then, } \Pi_1 \cup \Pi_2 \text{ has at least an answer set too. Thus, by Strong Consistency, } \Pi_1 \otimes \Pi_2 \equiv \Pi_1 \cup \Pi_2 \equiv \Pi_2 \cup \Pi_1 \equiv \Pi_2 \otimes \Pi_1. \text{ Now suppose that both } \Pi_1 \text{ and } \Pi_2 \text{ have no answer sets. Then, the update program } (\Pi'_1 \cup \Pi'_2, \mathcal{A}) \text{ never has generalised answer sets. Thus, } \Pi_1 \otimes \Pi_2 \equiv \Pi_2 \otimes \Pi_1 \text{ in either case.} \qed

The reader can find a general formal description of an implementation of this operator, as well as the implementation itself at [http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html](http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html).

6 Computing $\otimes$ with ODLP

As an important element of Logic Programming that distinguishes it over other theoretical approaches, this section presents both foundation and software tools of the implementation of this semantics, coming from Ordered Disjunctive Logic Programming.

Ordered Disjunctive Logic Programming, ODLP by [Bre02, BNS04], may be defined in an intuitive broad way as follows: a simple ordered disjunction program is a set of rules of the form:

$$C_1 \times \cdots \times C_n \leftarrow A_1, \ldots, A_m, \text{not } B_1, \ldots, \text{not } B_k$$

where $C_i, A_j$ and $B_l$ are all ground literals. $C_1, \ldots, C_n$ are usually named the choices of a rule and their intuitive reading is as follows: The ordered disjunction is used only in rule heads to select some of the answer sets of a program as the preferred ones. If $C_1$ is possible, then $C_1$; if $C_1$ is not possible, then try $C_2$; . . . ; if neither $C_1, \ldots, C_{n-1}$ is possible then try $C_n$. Moreover, one may identify some special cases such as: if $n = 0$ the rule is a constraint; and finally, facts are those rules where $m = k = 0$.

In the particular case of $\otimes$-operation, the required codification of ordered disjunctive programs is just $n = 2, m = k = 0.$
Last, as a logic programming topic, it is very important to notice that PSmodels\(^3\) is an ODLP implemented prototype, which consists of an extension to SMODEL\(^4\) to compute preferred stable models of normal logic programs. However, it is also important to point out that the sources themselves need some maintenance to fix some few bugs\(^5\).

In order to compute the MGAS\(^6\)'s of an abductive logic program, there is a translation from [OOZ04] to realise it in ODLP and the translation goes as follows.

### 6.1 Translating to ODLP

The following function is a version of the one by [OOZ04] in my own notation, that takes an abductive logic program and translates it into an ordered-disjunctive one.

**Definition 12** (Ordered Translation, \(O\) [OOZ04]). Let \(\langle \Pi, A^* \rangle\) be an abductive logic program. A translation into an ordered program, denoted as \(O(\Pi, A^*)\), consists of the following. For any literal \(\ell \in A^*\), the clause \(\rho_\ell\) is a rule of the form \(\alpha' \times \alpha \leftarrow ^\top\), where \(\alpha'\) is a literal that does not occur in the original abductive program. Then, 
\[
O(\Pi, A^*) = \Pi \cup \{\rho_\ell | \alpha \in A^*\}.
\]

The intuition behind this is to take an abductive program and to append the abductive atoms (in a form of ODLP rules) to its ELP program. Then, the resulting union is a regular ODLP program. The following example, borrowed from [OOZ04] who in turn was inspired by [BG03]'s, illustrates the just defined translation.

**Example 4** ([OOZ04]). Suppose the abductive program \(\langle \Pi, \{q, s, t\} \rangle\) where

\[
\Pi = \{p \leftarrow \neg q \quad r \leftarrow \neg s \quad q \leftarrow t \quad s \leftarrow t \quad \bot \leftarrow p, r\}
\]

\(^3\) The sources may be downloaded from [http://www.tcs.hut.fi/Software/smodels/priority](http://www.tcs.hut.fi/Software/smodels/priority) and there is a graphical user front end at [http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html](http://www.in.tu-clausthal.de/~guadarrama/updates/psmodels.html) that allows to execute preferred logic programs online.

\(^4\) This solver may be downloaded from [http://www.tcs.hut.fi/Software/smodels/](http://www.tcs.hut.fi/Software/smodels/) and run via online with a graphical user front end at [http://www.in.tu-clausthal.de/~guadarrama/updates/smodels.html](http://www.in.tu-clausthal.de/~guadarrama/updates/smodels.html).

\(^5\) For instance, Version 2.26a crashes with a simple program like \(\{a\}\).
The corresponding ordered program \(O(\Pi, \{q, s, t\})\) is then

\[
\begin{align*}
q' \times q & \leftarrow \top \\
q & \leftarrow \text{not } q \\
q' & \leftarrow t \\
q & \leftarrow t \\
r & \leftarrow \text{not } s \\
s & \leftarrow t \\
s' & \leftarrow \top
\end{align*}
\]

with three preferred answer sets: \(\{s, t, q, q', s'\}; \{r, q, s', t'\}; \{p, s, q', t'\}\). Note that \(\Pi\) is inconsistent. However, the new ordered program is now consistent.

Last, the generalisation of this translation proves to be correct, with a slight correction of a typo from the original lemma:

**Lemma 3** ([OOZ04]). \(M \cap L_\Pi\) is a generalised answer set of an abductive program \((\Pi, A^*)\) if and only if \(M\) is a preferred answer set of \(O(\Pi, A^*)\)

and the validity of set inclusion in ODLP:

**Theorem 4** ([OOZ04]). Let \((\Pi, A^*)\) be an abductive program and \(M\) a set of atoms. \(M \cap L_\Pi\) is a minimal generalised answer set of the abductive program if and only if \(M\) is an i-preferred answer set of \(O(\Pi, A^*)\).

Now it is easy to see how to use ODLP to update an extended logic program with \(\odot\) operation.

### 6.2 Updating with ODLP

Finally, this translation proves to be useful for the context of updates in MGAS with the following formalisation.

**Definition 13** (Ordered Disjunctive Update Program). Given an update program \(\Pi'=\odot\Pi_1\Pi_2\) over a set of atoms \(A\), and its corresponding abductive program \((\Pi'\cup\Pi_2, A^*)\), its Ordered Disjunctive Update Program corresponds to

\[O(\Pi' \cup \Pi_2, A^*)\]

The following examples illustrate how this translation works and how it can be computed with ODLP.
Example 5 (continued). Consider Example 3 again, with its corresponding abductive program \( \langle \Pi' \cup \Pi_2, A^* \rangle \), whose ODLP transformation, by Definition 12, consists of

\[
O(\Pi' \cup \Pi_2, A^*) = \{ \alpha'_1 \times \alpha_1 \leftarrow \top, \alpha'_2 \times \alpha_2 \leftarrow \top, \alpha'_3 \times \alpha_3 \leftarrow \top, \alpha'_4 \times \alpha_4 \leftarrow \top
\]

\[
sleep \leftarrow \neg tv(on), \neg \alpha_1 \\
night \leftarrow \neg \alpha_2 \\
watch(tv) \leftarrow tv(on), \neg \alpha_3 \\
tv(on) \leftarrow \neg \alpha_4 \\
\neg tv(on) \leftarrow power(failure) \\
power(failure) \leftarrow \top
\]

By Theorem 4, the minimal generalised answer sets of every abductive program \( \langle \Pi, A^* \rangle \) correspond to the intended models of some ordered disjunctive program \( \Pi' \) that can be easily run on a computer. As a result, the unique preferred answer set of such an ODLP program,

\[
\{ \text{sleep}, \alpha'_1, \text{night}, \alpha'_2, \alpha'_3, \alpha_4, \neg \text{tv(on)}, \text{pfailure} \}
\]

coincides with our intuition.

Another interesting experiment is one that has to do with an update with inert information.

As an expected result, this kind of translation led to an implemented system at [http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html](http://www2.in.tu-clausthal.de/~guadarrama/updates/pairs.html).

7 Conclusions

This report shows a set of properties for updates that help overcome syntax-dependency problems in other approaches. A basic framework for updates has emphasised the importance of an approach based on key structural properties by satisfying weak irrelevance of syntax and strong consistency, among others. Moreover, the report also has shown equivalence between GAS’s and ODLP, by using extended logic programs. The report also illustrates with examples how to overcome several problems occurring in alternative semantics for updates.

8 Acknowledgement

I am very grateful to Wojciech Jamroga and Jürgen Dix for their discussions, many useful comments and for proof-reading this report to ensure its optimal quality.
References


UPDATE OPERATION IN ASP REVISITED


