A Road Map of Updating in ASP

J. C. Acosta Guadarrama

Ifl Technical Report Series

Ifl-07-16

Department of Informatics
Clausthal University of Technology
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)
A Road Map of Updating in ASP

J. C. Acosta Guadarrama

Institute of Computer Science,
TU-Clausthal, Germany
guadarrama@in.tu-clausthal.de

Abstract

As one of the major and traditional topics of Artificial Intelligence over many years, knowledge representation and reasoning has proved to be a strong theoretical framework for Logic Programming to manage dynamic knowledge bases. In this report, we go through current and some of those past proposals to update ASP programs, by analysing their features and identifying challenges to represent correct evolving knowledge.

1 Introduction

As one of the major and traditional topics in Artificial Intelligence over the last years, knowledge representation and reasoning has proved to be a strong theoretical framework to manage knowledge bases. As a result, this particular topic has become more widely applied in administration of knowledge bases of intelligent (rational) agents, especially in situations of incomplete knowledge from a changing environment, and this area of research is known in the literature as belief updates.

The history of semantics for updates of logic programs is rather long. Indeed, it starts in the days of some of the first versions of Prolog with its commands assert and retract. However, sooner they started to get inconsistencies and other (unexpected) side effects. It was also time of research on databases with publications like [FUV83], and in particular for logical databases: [Win90, EKUV86, Fag95]. Nevertheless, some of the first formalisms to carry out proper changes to monotonic theories have been originally studied by [ACM85, KM89, Mak88, KM91, Lch92], while in the non-monotonic side by [KLM90, Mak94, LM92].

Some years later, [GL88] formulated the Stable Models Semantics (also refereed as Answer Sets Semantics, SM or simply ASP), and more concrete proposals arose within that framework, aimed at the problem of updating knowledge: [ZF95, Zha01, Zha06, SI99, SI03, ALP99, EFST01, EFST02, EFST00b, EFST00a, OZ03].

* This project is mainly supported by a CONACYT Doctorate Grant.
In this technical report, we go through current and some of those past proposals to update logic programs (or alike), by pointing out features as well as some of their limitations to represent correct evolving knowledge.

2 Eiter et alia

To the best of my knowledge, [EFST02] achieve the most complete survey of all known semantics for updating logic programs, by gathering relevant postulates and principles from the literature. This approach first appeared in [EFST00a] with a vast study of well-known and well-accepted postulates and properties, and later refined in [EFST02] and extended to be a main component in more general problems like agents in [EFST05] or preferences in [EFLP02], and they also implemented a solver available at www.kr.tuwien.ac.at/staff/giuliana/project.html#Download that is the main engine of an experimental graphical front end from us at www2.in.tu-clausthal.de/~guadarrama/updates/upd.html.

One of the main assets of [EFST02]'s proposal, as already mentioned, is being one of the first\(^1\) or even the first to realise a deep study of the literature of belief change of logic programs, in particular in ASP. [EFST02] formulate a natural definition for updating logic program sequences on a restricted Answer Sets language by rejecting rules under a causal-rejection principle. The principle is due to [ALP99] that later, however, turned out to be counterintuitive, even to themselves: see [EFST05, ABBL05, OZ03]. This “counter-intuition” comes from their strong dependency in the syntax of programs, according to [EFST05] Section 3 presents further discussion about this claim.

In particular, the natural formula under which [EFST05] analyse and describe update properties comes from [EFST02] as follows.

Given an update sequence \(\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)\) over a set of atoms \(\mathcal{A}\), with \(n \leq 2\), over a set of atoms \(\mathcal{A}\), assume \(\mathcal{A}^*\) as an extension of \(\mathcal{A}\) by new pair-wise unique atoms \(\text{rej}(\rho); \alpha_i\), for each rule \(\rho\) occurring in \(\Pi\); each atom \(\alpha \in \mathcal{A}\), and \(1 \leq i \leq n\). An injective naming function \(\text{Name}(\cdot, \cdot)\) is also assumed, which assigns to each rule \(\rho\) in a program \(\Pi\) a unique name, \(\text{Name}(\rho, \Pi_i)\), provided that \(\text{Name}(\rho, \Pi_i) \neq \text{Name}(\rho', \Pi_j)\) whenever \(i \neq j\). Finally, for a literal \(\ell_i\), denotes the result of replacing an atomic formula \(\alpha\) of \(\ell\) by \(\alpha_i\).

The intuitive idea of \(\text{rej}(\rho)\) is that of an atom that blocks (rejects or inhibits) a related rule \(\rho\) when true, provided that there is another more recent rule \(\rho'\) with conflicting information.

**Definition 1** (Update program [EFST02]). Given an update sequence

\[
\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)
\]

over a set of atoms \(\mathcal{A}\), the update program \(\Pi^\delta = (\Pi_1 \triangleleft \cdots \triangleleft \Pi_n)\) over \(\mathcal{A}^*\) consists of the following items:

\(^1\) They are the first, to the best of my knowledge.
(i) all constraints in $\Pi_i$, $1 \leq i \leq n$;

(ii) for each $\rho \in \Pi_i$, $1 \leq i \leq n$:

$$\ell_i \leftarrow \text{Body}(\rho), \text{not} \ rej(\rho) \quad \text{if} \ Head(\rho) = \ell;$$

(iii) for each $\rho \in \Pi_i$, $1 \leq i < n$:

$$rej(\rho) \leftarrow \text{Body}(\rho), \neg \ell_{i+1} \quad \text{if} \ Head(\rho) = \ell;$$

(iv) for each literal $\ell$ occurring in $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ ($1 \leq i \leq n$):

$$\ell_i \leftarrow \ell_{i+1}; \quad \ell \leftarrow \ell_i.$$

Note that in (iv) the authors write $\ell_1$ rather than $\ell_i$. Moreover, at the same (iv) they write $1 \leq i < n$ instead of $1 \leq i \leq n$. [Zha06] also detected these errors. Lastly, they do not state how to treat double negations that might happen in (iii).

Next, [EFST02] define the intended answer sets of an update sequence $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ in terms of the answer sets of $\Pi_\triangledown = (\Pi_1 \triangledown \cdots \triangledown \Pi_n)$. In other words, the models are back to the original alphabet by filter them out with the original atoms:

**Definition 2** (Answer sets of an update sequence [EFST02]). Let

$$\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$$

be an update sequence over a set of atoms $A$. Then, $S \subseteq \text{Lit}_A$ is an update answer set of $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ if and only if $S = S' \cap A$ for some answer set $S'$ of $\Pi_\triangledown = (\Pi_1 \triangledown \cdots \triangledown \Pi_n)$. The collection of all update answer sets of $\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n)$ is denoted by $U(\Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n))$.

There is a solver available for downloading at [www.kr.tuwien.ac.at/staff/giuliana/project.html#Download](http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download) and I have installed it to run online at [www2.in.tu-clausthal.de/~guadarrama/updates/upd.html](http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html) which also provides a graphic-oriented interface on the server itself. Naturally, no download or installation is necessary to run it.

Supposing the corrected semantics is what the authors wanted, computing their following example is possible:

**Example 1** ([EFST02]). Assume a daily update regarding an energy flaw represented by the sequence $(\Pi_1, \Pi_2)$ where

$$\Pi_1 = \{\text{sleep} \leftarrow \text{night}, \text{not tvon} \}$$

$$\text{night} \leftarrow \top$$

$$\text{watchtv} \leftarrow \text{tvon}$$

$$\text{tvon} \leftarrow \top \}$$

$$\Pi_2 = \{\neg \text{tvon} \leftarrow \text{powerfailure} \}$$

$$\text{powerfailure} \leftarrow \top \}$$
by Definition 1 the update program $\Pi_3 = (\Pi_1 \circ \cdots \circ \Pi_n)$ consists of rules (1)–(16):

$\begin{align*}
\text{sleep}_1 & \leftarrow \text{night}, \text{not tvon}, \text{not rej}(\rho_1) & (1) \\
\text{night}_1 & \leftarrow \text{not rej}(\rho_2) & (2) \\
\text{watchtv}_1 & \leftarrow \text{tvon}, \text{not rej}(\rho_3) & (3) \\
\text{tvon}_1 & \leftarrow \text{not rej}(\rho_4) & (4) \\
\neg \text{tvon}_2 & \leftarrow \text{powerfailure}, \text{not rej}(\rho_5) & (5) \\
\text{powerfailure}_2 & \leftarrow \text{not rej}(\rho_6) & (6) \\
\text{rej}(\rho_1) & \leftarrow \text{night}, \text{not tvon}, \neg \text{sleep}_2 & (7) \\
\text{rej}(\rho_2) & \leftarrow \neg \text{night}_2 & (8) \\
\text{rej}(\rho_3) & \leftarrow \text{tvon}, \neg \text{watchtv}_2 & (9) \\
\text{rej}(\rho_4) & \leftarrow \neg \text{tvon}_2 & (10) \\
\text{sleep}_1 & \leftarrow \text{sleep}_2 & (11) \\
\text{night}_1 & \leftarrow \text{night}_2 & (12) \\
\text{tvon}_1 & \leftarrow \text{tvon}_2 & (13) \\
\text{watchtv}_1 & \leftarrow \text{watchtv}_2 & (14) \\
\neg \text{tvon}_2 & \leftarrow \neg \text{tvon}_3 & (15) \\
\text{powerfailure}_2 & \leftarrow \text{powerfailure}_3 & (16)
\end{align*}$

whose unique answer set is

$\{ \text{sleep}_1, \text{night}, \text{night}_1, \text{rej}(\rho_1), \neg \text{tvon}_2, \text{powerfailure}, \text{powerfailure}_2, \text{sleep}, \neg \text{tvon} \}$

and its update answer set is easily obtained: $\{ \text{night}, \text{powerfailure}, \text{sleep}, \neg \text{tvon} \}$.

However, by following the original Definition 2 in \cite{EFST02}, rules (15)–(16) would not exist, for $i$ should also equal $n$ in Definition 1 item (iv), and the answer set of the resulting program is

$\{ \text{night}, \text{tvon}, \text{night}_1, \text{watchtv}, \text{tvon}_1, \text{powerfailure}_2, \text{watchtv} \}$

(17)

that means the TV is on and the agent is watching it. On the other hand, by changing $i$ to be within the range I suggest and by leaving the second rule in (iv) as the original definition, that is to say, $\ell \leftarrow \ell_1$, the resulting program would have rules

$\begin{align*}
\neg \text{tvon}_2 & \leftarrow \neg \text{tvon}_3 \\
\neg \text{tvon} & \leftarrow \neg \text{tvon}_1 \\
\text{powerfailure}_2 & \leftarrow \text{powerfailure}_3 \\
\text{powerfailure} & \leftarrow \text{powerfailure}_1
\end{align*}$

instead of rules (15)–(16) and would have the same strange answer set in (17).
In other words, by following the original restriction presented in (iv) in Definition 2 in [EFST02], \( \text{powerfailure} \notin \Pi = (\Pi_1, \Pi_2, \ldots, \Pi_n) \) when \( 1 \leq i < n \). Moreover, if the second rule in (iv) of Definition [1] was \( \ell \leftarrow \ell_1 \), a strange answer set would result:

\[
\{ \text{night}, \text{night}_1, \text{powerfailure}_2, \text{watchtv}_1, \text{watchtv}, \text{tvon}_1, \text{tvon} \}
\]

Anyway, their solver at [www.kr.tuwien.ac.at/staff/giuliana/project.html#Download](http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download) seems to behave well. Unfortunately, their solver does not show intermediate transformations to figure out the correct semantic parameters so that I can give a precise statement. Notice that I have only provided a front end to execute their solver in the web with a graphical interface, and I employed the latter as the main engine of my front end: [www2.in.tu-clausthal.de/~guadarrama/updates/upd.html](http://www2.in.tu-clausthal.de/~guadarrama/updates/upd.html).

Back to the corrections I suppose, let us complete Example [1]

**Example 2 (Continued from Example [1]).** Make a second update to the sequence in Example [7] with the program \( \Pi_3 = \{ \neg \text{powerfailure} \} \). Accordingly, the new answer set of the resulting update program is

\[
\{ \text{tvon}_1, \text{tvon}, \text{night}_1, \text{night}, \text{watchtv}_1, \text{watchtv}, \text{rej}(p_6), \neg \text{powerfailure}_3, \neg \text{powerfailure} \}
\]

Finally, by Definition [2] the corresponding update answer sets are

\[
\mathcal{U}(\Pi_1, \Pi_2) = \{ \text{night}, \text{powerfailure}, \text{sleep}, \neg \text{tvon} \}
\]

and

\[
\mathcal{U}(\Pi_1, \Pi_2, \Pi_3) = \{ \text{tvon}, \text{night}, \text{watchtv}, \neg \text{powerfailure} \}
\]

Despite the complete deep nice analysis [EFST02] make of known postulates and principles in the literature, one of the major shortcomings of their approach has to do with syntactic and semantic contents.

Take again, for instance, the example suggested by [ABBL05], that may produce counterintuitive models:

**Example 3.** Suppose an agent who believes that when it is day there is no night and vice versa, and that there are stars when it is night and when there are no clouds. Finally, that at the current moment it is a fact that there are no stars. This short story may be coded into \( \Pi_1 \) as follows:

\[
\Pi_1 = \{ \text{day} \leftarrow \neg \text{night} \\
\text{night} \leftarrow \neg \text{day} \\
\text{stars} \leftarrow \text{night}, \neg \text{cloudy} \\
\neg \text{stars} \}
\]

\[\text{Unfortunately the sources are not available so as to confirm the latest definition.}\]
DyLP and Other Dialects

whose unique answer set is \{\text{day}, \neg \text{stars}\}. Later, the agent acquires new information stating that stars and constellations are the same thing, as coded in $\Pi_2$. So, if the agent updated $\Pi_1$ with program

$$
\Pi_2 = \{ \text{stars} \leftarrow \text{constellations} \\
\text{constellations} \leftarrow \text{stars} \}
$$

the expanded alphabet of the two programs contains only one new extra atom with respect to $\Pi_1$: constellations. As the model of $\Pi_2$ is obviously the empty answer set, constellations is considered synonymous of stars by means of $\Pi_2$, and thus the update should not change the original knowledge base. However, the update introduces an extra answer set in many of the existing update semantics based on the causal-rejection principle see for example [ALP⁺99, EFST02, ABBL05]: \{\text{stars}, \text{constellations}, \text{night}\} which does not coincide with common intuition.

To recapitulate, [EFST02] were very good in gathering postulates and principles from the literature and in analysing them in terms of their proposal. Their proposal, however, suffers from drawbacks owing to its reliance on the causal-rejection principle see [ALP⁺99].

3 DyLP and Other Dialects

One of the earliest approaches in updating logic programs appeared in late 90’s in [ALP⁺99, ALP⁺98] that was extended in an interesting language called LUPS by [APPP02], to specify explicit updates in programs on a semantics that they called Dynamic Logic Programming or DyLP —[ALP⁺99]. Some years later, they refined the latter in [ABBL05], whom over the previous period formulated a principle of rejection (also causal-rejection principle [ALP⁺99, EFST02, ABBL05]) in the above citations.

Informally, the refined principle consists of rejecting rules of previous and up-coming programs in an update sequence whenever there are other rules at the current state with which they conflict. Starting with motivation in [ABBL05], they claim to give a simple example to what they called a tautology (a rule from which they expect no extra models):

$$
\text{not } p \leftarrow \text{not } p
$$

Of course that rule alone does not produce any new model in their semantics —i.e. just the empty model, \{\}. However, it is a clear counterexample why strong negation in our framework should not be a simple replacement to “not ℓ” in heads.

Example 4. Take for example, the program

$$
\{ \neg p \leftarrow \text{not } p \}
$$

whose unique answer set is not the empty set. Namely, the answer set of (19) is just \{\neg p\}. 

INSTITUTE OF COMPUTER SCIENCE
A ROADMAP OF UPDATING IN ASP

What is more, in their article, [ABBL05] explain in a footnote what tautology means: A rule of the form \( \ell \leftarrow \text{Body} \) with \( \ell \in \text{Body} \), where \( \ell \) and \( \text{Body} \) are an atom (or default-negated atom) and the body of a rule, respectively. This high dependency in syntax will prove to be one of their major shortcomings, as explained along this thesis.

Before starting with their proper definitions, a very special notation, taken from [ABBL05], is necessary.

Let \( \mathcal{A} \) be a set of propositional atoms. As before introduced, a default literal is an atom preceded by “not”, while a literal is either an atom or a default literal. A rule \( \rho \) is an ordered pair \( \text{Head}(\rho) \leftarrow \text{Body}(\rho) \) where \( \text{Head}(\rho) \) (the head of the rule) is a literal and \( \text{Body}(\rho) \) is a finite set of literals, and it has the form \( L_0 \leftarrow L_1, \ldots, L_n \). A rule with \( \text{Head}(\rho) = L_0 \) and \( \text{Body}(\rho) = \emptyset \) is called a fact, simply written as \( L_0 \).

A generalised logic program (GLP) \( \Pi \) over \( \mathcal{A} \), is a finite or infinite set of rules, and \( \Pi_0 \) denotes an empty set of rules. If \( \text{Head}(\rho) = a \) (resp. \( \text{Head}(\rho) = \lnot a \)) then not \( \text{Head}(\rho) = a \) (resp. not \( \text{Head}(\rho) = a \)). Two rules \( \rho \) and \( \rho' \) are in conflict, denoted by themselves as \( \rho \bowtie \rho' \), if and only if \( \text{Head}(\rho) = \lnot \text{Head}(\rho') \).

An interpretation \( M \) of \( \mathcal{A} \) is a set of atoms such that \( M \subseteq \mathcal{A} \). An atom \( a \) is true in \( M \), denoted by \( M \models a \), if and only if \( a \in M \), and false otherwise. A default literal \( \lnot a \) is true in \( M \), denoted by \( M \models \lnot a \), if and only if \( a \notin M \), and false otherwise. A set of literals \( L \) is true in \( M \), denoted by \( M \models L \) if and only if each literal in \( L \) is true in \( M \). A rule \( \rho \) is satisfied by an interpretation \( M \) if and only if whenever \( M \models \text{Body}(\rho) \) then \( M \models \text{Head}(\rho) \). An interpretation \( M \) is a model of a program \( \Pi \) if and only if \( M \) satisfies all rules in \( \Pi \). An interpretation \( M \) of \( \mathcal{A} \) is a stable model of a generalised logic program \( \Pi \) if and only if

\[
M = \text{least}(\Pi \cup \{ \lnot a \mid a \notin M \}) \tag{20}
\]

where
\[
\text{least}(\cdot) \quad \text{is the least model of the definite program obtained from the argument program by replacing every default literal } \lnot a \text{ by a new atom } \lnot _a.
\]

With this notation, one can define a dynamic program as follows.

**Definition 3** (Dynamic Logic Program, DyLP [ABBL05]). A dynamic logic program (DyLP) is a sequence of generalised logic programs. Let \( \mathcal{P} = (\Pi_1, \ldots, \Pi_s) \) and \( \mathcal{P}' = (\Pi'_1, \ldots, \Pi'_t) \) be two DyLPs. The expression \( \rho(\Pi) \) denotes the set of all rules appearing in the programs \( \Pi_1, \ldots, \Pi_s \), and \( \mathcal{P} \cup \mathcal{P}' \) denotes the DyLP: \( (\Pi_1 \cup \Pi'_1, \ldots, \Pi_s \cup \Pi'_t) \).

Lastly, the refined interpretation of a DyLP program consists in computing the least model of the positive program that results from the difference of the rejected rules, where the intuition behind \( \text{Rej}(\cdot, \cdot) \) is the set of rules that are in conflict with both

\[\text{Consider that there seems to be a typo in [ABBL05], they typed “not a” rather than “not_a” in (20).}\]

\[\text{Note that indeed this sort of atoms is positive.}\]
DyLP and Other Dialects

current and previous rules in the sequence. Moreover, \(\text{Def}(\cdot, \cdot)\) consists of the positive “default-negated” atoms that do not appear in the intended model.

**Definition 4** (Dynamic Stable Model \[ABBL05\]). Let \(\mathcal{P}\) be a dynamic logic program and \(M\) an interpretation. \(M\) is a dynamic stable model of \(\mathcal{P}\) if and only if

\[
\overline{M} = \text{least}(\rho(\mathcal{P}) \setminus \text{Rej}(\mathcal{P}, M) \cup \text{Def}(\mathcal{P}, M))
\]

(21)

where

\[
\text{Rej}(\mathcal{P}, M) = \{ \rho \mid \rho \in \Pi_i, \exists \rho' \in \Pi_j, i \leq j, \rho \triangleright \rho', M \models \text{Body}(\rho') \}
\]

(22)

and

\[
\text{Def}(\mathcal{P}, M) = \{ \text{not}_\alpha \mid \exists \rho \in \rho(\mathcal{P}), \text{Head}(\rho) = \alpha, M \models \text{Body}(\rho) \}
\]

(23)

\(\oplus_R\) is the corresponding update operator.

This approach has had several implementations for download, including one for the original version before the refined principle, and another for the refined principle. LUPS is also implemented and the following list shows their respective locations:

- [centria.di.fct.unl.pt/~jja/updates/dlp.html](http://centria.di.fct.unl.pt/~jja/updates/dlp.html)
- [centria.di.fct.unl.pt/~banti/FedericoBantiHomepage/refdlp.htm](http://centria.di.fct.unl.pt/~banti/FedericoBantiHomepage/refdlp.htm)
- [centria.di.fct.unl.pt/~jja/updates/lups.html](http://centria.di.fct.unl.pt/~jja/updates/lups.html)

By considering Example 3 again, and inspired from the original example from \[ABBL05\], the reader may rewrite the pair of programs as follows.

**Example 5.** Let \(\Pi_1 \oplus_R \Pi_2\), where

\[
\Pi_1 = \{ \text{day} \leftarrow \text{not night} \\
\text{night} \leftarrow \text{not day} \\
\text{stars} \leftarrow \text{night, not cloudy} \\
\text{not_stars} \}
\]

and

\[
\Pi_2 = \{ \text{stars} \leftarrow \text{constellations} \\
\text{constellations} \leftarrow \text{stars} \}
\]

One of the resulting dynamic stable models is just \(M = \{ \text{day} \}\) because

\[
\overline{M} = \{ \text{day, not_night, not_cloudy, not_stars, not_constellations} \}
\]

\(5\) Notice that it seems they missed the “\_” under their “\text{not }\alpha” in (23).
where
\[ \text{Rej}(\mathcal{P}, M) = \{ \text{stars} \leftarrow \text{night}, \text{not cloudy} \} \]
and
\[ \text{Def}(\mathcal{P}, M) = \{ \text{not night}, \text{not stars}, \text{not cloudy}, \text{not constellations} \} \]
Thus, \( \mathcal{M} = M \cup \{ \text{not night}, \text{not stars}, \text{not cloudy}, \text{not constellations} \} \). However, the interpretation \( M = \{ \text{night}, \text{stars}, \text{constellations} \} \) is also a (refined) dynamic stable model, simply because \( \text{Def}(\mathcal{P}, M) = \{ \text{not cloudy}, \text{not day} \} \) and \( \text{Rej}(\mathcal{P}, M) = \{ \text{not stars} \} \). As a result, one of the least models is
\[ \mathcal{M} = \{ \text{night, stars, constellations} \} \cup \text{Def}(\mathcal{P}, M) = M \cup \text{Def}(\mathcal{P}, M) \]
which is a clear disadvantage, besides the unnecessary emulation of strong negation and default negation in heads.

Although it is true that [ABBLO5] were some of the first people to formulate and implement a semantics for updates, one can easily realise the clear shortcomings this approach has: firstly for the different syntax of the so-called generalised logic programs that is a different case of SM—a non-standard concept of SM [EFST02]; secondly for the principle itself that produces counterintuitive results.

## 4 Sakama & Inoue — SI

According to the authors in [SI99, SI03], there exist three types of updates: inconsistency removal, view updates and theory updates. Each of those types is a special case of updates and revision.

In particular, the present thesis focuses on theory updates, rather than other special cases of making an inconsistent program consistent or differentiating between variant and invariant knowledge. As a result, I do not go through the other types here, although they are sometimes related to the particular problem addressed in this thesis.

Before introducing the definitions for theory updates, some new notation and specialised terminology is necessary to understand their approach. For instance, the authors define their particular framework of abduction, and they call it Extended Abduction. This framework differs from the standard abduction definition in [Poo88, KM90, KKT98]. For instance, besides an explanation to satisfy
\[ K \cup E \models G \]
they also introduce the notion of negative explanations, such that
\[ K \setminus F \models G \]
where \( K \) is a first-order theory; \( E, F \) sets of hypotheses; \( G \) an observation; and \( K \cup E \) and \( K \setminus F \) are consistent.
According to [SI03], an extended abductive program is a pair \( \langle P, A^* \rangle \) where \( P \) and \( A^* \) are DLP’s. An abductive program \( \langle P, A^* \rangle \) is consistent if \( P \) is consistent.

In the process of updating a program with another, the authors define a set of conditions that the intended updated must meet.

**Definition 5** (Theory Updates [SI03]). Given a DLP-program pair \( P \) and \( Q \), \( P' \) accomplishes a theory update of \( P \) by \( Q \) if

1. \( P' \) is consistent,
2. \( Q \subseteq P' \subseteq P \cup Q \),
3. there is no consistent program \( P'' \) such that \( P' \subset P'' \subseteq P \cup Q \).

In words of [SI03], the intended update is the union of the new information and a maximal subset of the original program that is consistent with the update, which obviously is not always unique.

Their update process starts as an extended abductive program \( \langle P \cup Q, P \setminus Q \rangle \). The intuition behind this program consists of merging the update with the original theory and combining the rules of \( P \) that do not belong to \( Q \) so as to get a consistent update.

In order to reduce the set of abducible rules \( P \setminus Q \) to abducible facts and to compute their models in a conventional way, the extended abductive program (as defined in [IS95]) must be transformed into a normal (traditional) abductive program —like in [KKT98]— in which abducibles contain only non-disjunctive facts, as in the below definition. In this manner, the models of an update program (later introduced) will contain both facts and names of rules to remove, rather than the rules themselves.

**Definition 6** (Normalised Abductive Program [SI03]). Given an extended abductive program \( \langle P, A^* \rangle \), and

\[
\mathcal{R} = \{ \Sigma \leftarrow \Gamma \mid (\Sigma \leftarrow \Gamma) \in A^* \text{ and } \Sigma \leftarrow \Gamma \text{ is not a non-disjunctive fact} \}.
\]

Then, let

\[
P^n = (P \setminus \mathcal{R}) \cup \{ \Sigma \leftarrow \Gamma, \gamma_R \mid R = (\Sigma \leftarrow \Gamma) \in \mathcal{R} \} \cup \{ \gamma_R \leftarrow | R \in \mathcal{R} \cap P \},
\]

\[
A^n = (A^* \setminus \mathcal{R}) \cup \{ \gamma_R \mid R \in \mathcal{R} \},
\]

where \( \gamma_R \) is a newly introduced atom (called the name of \( R \)) uniquely associated with each rule \( R \) in \( \mathcal{R} \). For any rule \( R \in \mathcal{R} \), its name comes from the function \( n(R) = \gamma_R \). In particular, any abducible fact \( L \leftarrow \top \) has the name \( L \), i.e., \( n(L) = L \).
A ROADMAP OF UPDATING IN ASP

Once the extended abductive program is normalised, its interpretation is the models of an update program that consists of the rules of the original theory that are not in the normalised abductive set, merged with a new set of update rules, as following specified.

**Definition 7** (Update Rules; Update Atoms [SI03]). *Given an extended abductive program* \( (P, A^*) \), *where* \( A^* \) *contains only (non-disjunctive) facts, the set* \( UR \) *of update rules is constructed as follows.*

1. For any literal \( a \in A^* \), the following rules are in \( UR \):
   \[
   a \leftarrow \neg \pi,
   \pi \leftarrow \neg a,
   \]
   where \( \pi \) is a newly introduced atom uniquely associated with \( a \). The above pair of rules function is \( abl(a) \) hereafter. In addition, yet another semantically equivalent way to represent it, according to [SI03], is by \( a \lor \neg \pi \leftarrow \top \).

2. For any literal \( a \in A^* \setminus P \), the following rule is in \( UR \):
   \[a^{ON} \leftarrow a.\]

3. For any literal \( a \in A^* \cap P \), the following rule is in \( UR \):
   \[a^{OUT} \leftarrow \neg a.\]

where \( a^{ON} \) and \( a^{OUT} \) are atoms uniquely associated with any \( a \in A^* \), so called update atoms, \( UA \).

[SI03] interpret, at a meta-level, that \( a^{ON} \) means making \( a \) true, when it is not in \( P \), while \( a^{OUT} \) means making \( a \) not true when it is in \( P \). In other words, they represent the introduction and deletion of \( a \), respectively. On the other hand, \( \pi \) would mimic an unknown truth value of \( a \), i.e. neither true nor false \( a \). Last, they define the set of all update atoms associated with the abducibles in \( A^* \) by \( UA \). That is to say, \( UA = UA^{ON} \cup UA^{OUT} \), where \( UA^{ON} \) (resp. \( UA^{OUT} \)) is the set of update atoms of the form \( a^{ON} \) (resp. \( a^{OUT} \)).

Next, these update rules take part of the update program of the normalised extended abductive program that is an intermediate DLP. This intermediate program specification is as follows:

**Definition 8** (Update Programs, UP [SI03]). *Given an extended abductive program* \( (P, A^*) \), *its update program* \( UP \) *is defined as a DLP such that*

\[
UP = (P \setminus A^*) \cup UR.
\]

Then they compute the models of an update program, which point out the deletion of facts or rules from the original program in the pair. As a result, a new updated program (or more than one) comes up.
Definition 9 (U-minimal Answer Sets [SI03]). An answer set $S$ of $\mathcal{UP}$ is called U-minimal (U-MAS) if there is no answer set $S'$ of $\mathcal{UP}$ such that $S' \cap \mathcal{UA} \subset S \cap \mathcal{UA}$.

This abduction framework proofs to have nice properties of minimal change when performing particular kinds of updates of non-monotonic theories, and when maintaining their consistency, with a vast analysis of shortcomings in other approaches. Rather than characterising updates through abduction, however, [SI03]'s first goal is the converse, providing a mechanism (an update semantics) to characterise their extended abduction, as they themselves state. Consequently, the approach lacks of a proper analysis of more principles and postulates from the literature. Additionally, they characterise different kinds of updates with their extended abduction, claiming that they can provide an algebra of rules deletion, besides the addition of them, to explain observations.

Let us carry on by recapitulating their approach. They define a called update program out of the normal abductive form of an extended abductive program

$$\langle \mathcal{P} \cup \mathcal{Q}, \mathcal{P} \setminus \mathcal{Q} \rangle$$

whose models are U-MAS's, interpreted from an update program. Last, the interpretation produces one (or more) new programs representing knowledge bases, derived from the addition/deletion of facts that the U-MAS's describe in turn.

In order to illustrate the above definitions, consider the following example extended from the original in [SI03] that shows one of the differences with several approaches.

Example 6. Suppose an update to the knowledge base

$$\Pi_1 = \{ \text{sleep} \leftarrow \text{not tvon} \}
\quad \text{watchtv} \leftarrow \text{tvon}
\quad \text{tvon} \leftarrow \top \}$$

with\(^6\)

$$\Pi_2 = \{ \text{powerfailure} \leftarrow \top
\quad \bot \leftarrow \text{powerfailure, tvon} \}$$

The situation is in the abductive program $\langle \Pi_1 \cup \Pi_2, \Pi_1 \setminus \Pi_2 \rangle$. The update program

\(^6\)Originally, [ALP+99] proposed this example, but it is a little modified in [SI03] to contrast their differences. Moreover, I have extended it here in order to see further details.

\(^7\)In [ALP+99] the rule “$\neg \text{powerfailure}, \text{tvon}$” is given as “$\neg \text{tvon} \leftarrow \text{powerfailure}$”. These two rules are semantically equivalent under the answer set semantics, as the authors explain in [SI03]. However, as later seen in this example, the difference between either expression would result in the existence of $\neg \text{tvon}$ in the corresponding model!
A ROADMAP OF UPDATING IN ASP

**UP** of \((\Pi_1 \cup \Pi_2)^n, (\Pi_1 \setminus \Pi_2)^n)\) is specified as

\[
\begin{align*}
\text{UP:} & \quad \text{powerfailure} \leftarrow \top \\
& \quad \bot \leftarrow \text{powerfailure}, \text{tvon} \\
& \quad \text{sleep} \leftarrow \neg \text{tvon}, \gamma_1 \\
& \quad \text{watchtv} \leftarrow \text{tvon}, \gamma_2 \\
& \quad \text{abd}((\text{tvon}), \text{abd}(\gamma_1), \text{abd}(\gamma_2), \text{tvon}^{\text{OUT}}) \leftarrow \text{tvon} \\
& \quad \gamma_1^{\text{OUT}} \leftarrow \neg \gamma_1 \\
& \quad \gamma_2^{\text{OUT}} \leftarrow \neg \gamma_2 \\
\end{align*}
\]

where \(\gamma_1\) and \(\gamma_2\) are names of the abducible rules in \(\Pi_1 \setminus \Pi_2\). Then, **UP** has the unique U-MAS

\[
\{\text{powerfailure, sleep, tvon, tvon}^{\text{OUT}}, \gamma_1, \gamma_2\}
\]

which represents the deletion of fact tvon from \(\Pi_1 \cup \Pi_2\). As a result, the theory update of \(\Pi_1\) by \(\Pi_2\) becomes

\[
\Pi_3: \quad \text{sleep} \leftarrow \neg \text{tvon} \\
\text{watchtv} \leftarrow \text{tvon} \\
\text{powerfailure} \leftarrow \top \\
\bot \leftarrow \text{powerfailure}, \text{tvon}
\]

whose answer set is just \(\{\text{powerfailure, sleep}\}\).

Next, suppose yet another update

\[
\Pi_4: \quad \neg\text{powerfailure} \leftarrow \top
\]

to \(\Pi_3\), which represents that the power is back again. [[Sto03]] code this new pair by the abductive program \((\Pi_3 \cup \Pi_4, \Pi_3 \setminus \Pi_4)\). So, the update program of \((\Pi_3 \cup \Pi_4)^n, (\Pi_3 \setminus \Pi_4)^n)\) turns into

\[
\begin{align*}
\text{UP:} & \quad \neg\text{powerfailure} \leftarrow \top \\
& \quad \text{sleep} \leftarrow \neg \text{tvon} \gamma_1 \\
& \quad \text{watchtv} \leftarrow \text{tvon} \gamma_2 \\
& \quad \bot \leftarrow \text{powerfailure}, \text{tvon} \gamma_3 \\
& \quad \text{abd}(\text{powerfailure}), \text{abd}(\gamma_1), \text{abd}(\gamma_2), \text{abd}(\gamma_3), \text{powerfailure}^{\text{OUT}} \leftarrow \neg \text{powerfailure} \\
& \quad \neg\gamma_1 \leftarrow \neg \gamma_1, \neg\gamma_2 \leftarrow \neg \gamma_2, \neg\gamma_3 \leftarrow \neg \gamma_3.
\end{align*}
\]

Then, **UP** has the unique U-MAS

\[
\{\neg\text{powerfailure, sleep, }\gamma_1, \gamma_2, \gamma_3, \text{powerfailure, }\neg\text{powerfailure}\}
\]
which implies that the result of the update is

\[(\Pi_3 \cup \Pi_4) \setminus \{\text{powerfailure} \leftarrow \top}\]

As a result, the unique answer set of the unique resulting program is

\[\{\neg\text{powerfailure}, \text{sleep}\}\]

[SI03] propose this example as an argument against other approaches like [ALP+99, EFST02], that bring back previous knowledge of the original theory. That is to say, their interpretation is that the TV turns itself on again and it is possible to watch it as well: \{\text{tvon}, \text{watchtv}, \neg\text{powerfailure}\}, which does not coincide with their intuition. However, this argument seems to be too strong to generalise that all update semantics should behave accordingly, because the authors are differentiating \textit{fluents} and \textit{actions} in a language that does not have such an explicit difference.

In order to illustrate this assumption, let us modify Example 6 in such a way that the language contains only fluents, naturally at a higher abstraction level. Therefore, the new story goes like this.

**Example 7.** Suppose a learning agent whose simple knowledge base states that it is innocent when it is not guilty, and at the beginning it believes it is not guilty, thus innocent. A following update states that the agent is no longer innocent when guilty, that it is guilty when a murderer and now it is a murderer. Thus, it is no longer innocent. However, more relevant rules pop up that state that an agent is not a murderer when self defended; that is self defended when first attacked; and it is a fact that the agent was first attacked. Consequently, common intuition would dictate that the agent’s innocence should be in effect. Under [SI03]’s approach, however, that previous knowledge is lost forever and there is no way to conclude that it is innocent.

Here is how [SI03]’s approach represents this knowledge. The knowledge base consists of an original theory, \(\Pi_1\), as well as an update to it, \(\Pi_2\), where

\[
\Pi_1 : \{\text{innocent} \leftarrow \neg\text{guilty} \}
\]

\[
\Pi_2 : \{\neg\text{innocent} \leftarrow \text{guilty} \}
\]

\[
\text{guilty} \leftarrow \text{murderer} \}
\]

\[
\text{murderer} \leftarrow \top \}
\]
and its normalised abductive program \( (P^n, A^n) \) where

\[
P^n = \{\neg\text{guilty} \leftarrow \top, \\
\gamma_1 \leftarrow \top, \\
murderer \leftarrow \top, \\
innocent \leftarrow \neg\text{guilty}, \gamma_1, \\
\neg\text{innocent} \leftarrow \text{guilty}, \\
guilty \leftarrow \text{murderer}\}
\]

\[
A^n = \{\neg\text{guilty} \leftarrow \top, \\
\gamma_1 \leftarrow \top\}
\]

The update rules and the update program consist respectively of

\[
UR : \{\neg\text{guilty} \lor \neg\text{guilty} \leftarrow \top, \\
\neg\text{guilty}^{\text{OUT}} \leftarrow \text{not} \neg\text{guilty}, \\
\gamma_1 \lor \gamma_1^{\text{OUT}} \leftarrow \top, \\
\gamma_1^{\text{OUT}} \leftarrow \text{not} \gamma_1\}
\]

\[
UP : \{\neg\text{guilty} \lor \neg\text{guilty} \leftarrow \top, \\
\neg\text{guilty}^{\text{OUT}} \leftarrow \text{not} \neg\text{guilty}, \\
\gamma_1 \lor \gamma_1^{\text{OUT}} \leftarrow \top, \\
\gamma_1^{\text{OUT}} \leftarrow \text{not} \gamma_1, \\
murderer \leftarrow \top, \\
innocent \leftarrow \neg\text{guilty}, \gamma_1, \\
\neg\text{innocent} \leftarrow \text{guilty}, \\
guilty \leftarrow \text{murderer}\}
\]

that has two answer sets

\[
\{\text{murderer}, \neg\text{guilty}, \neg\text{guilty}^{\text{OUT}}, \gamma_1, \gamma_1^{\text{OUT}}, \neg\text{innocent}, \text{guilty}\}
\]

\[
\{\text{murderer}, \neg\text{guilty}, \neg\text{guilty}^{\text{OUT}}, \gamma_1, \neg\text{innocent}, \text{guilty}\}
\]

from which the unique U-MAS

\[
\{\text{murderer}, \neg\text{guilty}, \neg\text{guilty}^{\text{OUT}}, \gamma_1, \neg\text{innocent}, \text{guilty}\}
\]

leads to the updated knowledge base where \(\neg\text{guilty}\) is no longer present:

\[
\Pi_3 : \{\text{innocent} \leftarrow \neg\text{guilty}, \\
\neg\text{innocent} \leftarrow \text{guilty}, \\
guilty \leftarrow \text{murderer}, \\
murderer \leftarrow \top\}
\]
Next, the following program represents the second update as:

\[
P_4 : \{ \neg \text{murderer} \leftarrow \text{self\_defence} \\
      \text{self\_defence} \leftarrow \text{attacked} \\
      \text{attacked} \leftarrow \top \}
\]

which, after the same process yields the following update program

\[
UP : \{ \gamma_1 \lor \gamma_1^{\text{OUT}} \leftarrow \top \\
      \neg \gamma_1 \leftarrow \not\not \gamma_1 \\
      \gamma_2 \lor \gamma_2^{\text{OUT}} \leftarrow \top \\
      \neg \gamma_2 \leftarrow \not\not \gamma_2 \\
      \text{murderer} \lor \neg \text{murderer} \leftarrow \top \\
      \neg \text{murderer}^{\text{OUT}} \leftarrow \not \not \text{murderer} \\
      \gamma_3 \lor \gamma_3^{\text{OUT}} \leftarrow \top \\
      \neg \gamma_3 \leftarrow \not\not \gamma_3 \\
      \text{attacked} \leftarrow \top \\
      \text{innocent} \leftarrow \neg \text{guilty}, \gamma_1 \\
      \neg \text{innocent} \leftarrow \text{guilty}, \gamma_2 \\
      \text{guilty} \leftarrow \text{murderer}, \gamma_3 \\
      \neg \text{murderer} \leftarrow \text{self\_defence} \\
      \text{self\_defence} \leftarrow \text{attacked} \}
\]

with the unique U-MAS

\[
\{ \text{attacked}, \gamma_1, \gamma_2, \overline{\text{murderer}}, \text{murderer}^{\text{OUT}}, \gamma_3, \neg \text{murderer}, \text{self\_defence} \}
\]

that produces a knowledge base

\[
P_5 : \{ \text{innocent} \leftarrow \neg \text{guilty} \\
      \neg \text{innocent} \leftarrow \text{guilty} \\
      \text{guilty} \leftarrow \text{murderer} \\
      \neg \text{murderer} \leftarrow \text{self\_defence} \\
      \text{self\_defence} \leftarrow \text{attacked} \\
      \text{attacked} \leftarrow \top \}
\]

that models \{ \text{attacked}, \neg \text{murderer}, \text{self\_defence} \} reflects the loss of previous relevant information—no conclusions about guilt or innocence are available.

If this counterintuitive example was not enough, let us change a bit the original Example 6 in such a way that both actions and fluents are inverted.
Example 8. Suppose a simple scenario in where an agent can see in a room where its blinds are open. Later, new information is at hand and the agent knows that it cannot see when the blinds are closed, that by closing them means they are closed, and that they cannot be closed and open at the same time. Simultaneously, there is also an event of closing the blinds. Following, a program that codes the initial information:

$$\Pi_1 : \{ \text{can see } \leftarrow \text{blinds open} \}$$

updated with

$$\Pi_2 : \{ \neg \text{can see } \leftarrow \text{blinds closed} \}$$

After updating $\Pi_1$ with $\Pi_2$, the update program

$$UP : \{ \text{blinds open } \lor \text{blinds open}^{\text{OUT}} \leftarrow \top \}$$

has the following $U$-MAS:

$$\{ \text{close_blinds, blinds_open, blinds_open}^{\text{OUT}}, \gamma_1, \neg \text{can see, blinds_closed} \}$$

This model means the deletion of fact $\text{blinds_open}$ from the original knowledge base.

Now suppose the agent decides not to close the blinds when it is reading, that it is reading when it wants to read, and that now it wants to read. Then, the updated
program and the new update are

\[
\begin{align*}
&\{ \text{can see} \leftarrow \text{blinds open} \\
&\neg \text{can see} \leftarrow \text{blinds closed} \\
&\bot \leftarrow \text{blinds open, blinds closed} \\
&\text{blinds closed} \leftarrow \text{close blinds} \\
&\text{close blinds} \leftarrow \top \} \\
&\{ \neg \text{close blinds} \leftarrow \text{reading} \\
&\text{reading} \leftarrow \text{want to read} \\
&\text{want to read} \leftarrow \top \}
\end{align*}
\]

whose unique U-MAS

\[
\{ \text{want to read}, \gamma_1, \gamma_2, \gamma_3, \text{close blinds}, \text{close blinds}^\text{OUT}, \gamma_4, \neg \text{close blinds}, \text{reading} \}
\]

produces an updated program

\[
\begin{align*}
&\{ \text{can see} \leftarrow \text{blinds open} \\
&\neg \text{can see} \leftarrow \text{blinds closed} \\
&\bot \leftarrow \text{blinds open, blinds closed} \\
&\text{blinds closed} \leftarrow \text{close blinds} \\
&\neg \text{close blinds} \leftarrow \text{reading} \\
&\text{reading} \leftarrow \text{want to read} \\
&\text{want to read} \leftarrow \top \}
\end{align*}
\]

with an answer set that again reflects a loss of information on the ability to see:

\[
\{ \text{want to read}, \neg \text{close blinds}, \text{reading} \}
\]

Clearly, the objection \cite{SI03} propose against other semantics may have different interpretations in planning scenarios, where there is indeed a formal explicit distinction between fluents and actions. Meanwhile, neither interpretation is correct or incorrect when talking about simple logic-program updates, unless formalising which rules must persist and which must not.

Moreover, although the authors present a deep analysis of their proposal and although it seems to be robust-enough for agent’s changing environment, there is a lack of further and more general properties that makes it hard to compare with other competitors.

5 Zhang’s line

An interesting proposal for updates comes from \cite{Zha06}, where the author identifies three types of problems to solve in an update process: elimination of contradictory
information, conflict resolution and syntactic representation.

Additionally, one of the applications from that line is an interesting language introduced in [CZ05] that is specialised in updates of agent policies and defined at the top of ASP. [CZ05] specify such policies in terms of clauses with a predefined semi-imperative syntactical structure, as well as an initial planning approach.

However, owing to a special focus the work has on policies, the programmer is restricted and obliged to use reserved words like “always”, “implied by”, “with absence”, etc. which, besides constraining the domain to specific applications, it 'reduces' the language and has potentially different meanings in the meta-language. Nevertheless, they already have a fully-fledged system, as they themselves mention it in [CZ05].

5.1 General View

As mentioned above, [Zha06] characterises updates in terms of three main objectives: contradiction elimination, conflict resolution and syntactic representation. The first topic is one of the most obvious in semantics for updates, which should be real by preserving a minimal-change principle and a proper justification. On the other hand, conflict resolution has to do with potential future contradictions an update might yield because of the introduction of the two kinds of negations in logic programs —strong and default negation. Finally, once a semantics meets the two main goals, the author argues that a proper semantics should also preserve as many as possible of the original rules from the updating knowledge base.

In order to realise these three goals, [Zha03] characterises a program update by means of a called prioritised logic program. In an intuitive way, this kind of program consists in preferring the latest update to the original knowledge base.

[Zha05] motivates his proposal by introducing a clever example that exposes the two kinds of problems he studied, and the example I borrow looks as follows:

**Example 9 ([ZF05]).** Suppose

\[
\Pi_0 = \{ \text{member}(a, g) \leftarrow T \} \quad (24)
\]
\[
\text{member}(b, g) \leftarrow T \quad (25)
\]
\[
\text{access}(a, f_2) \leftarrow T \quad (26)
\]
\[
\text{access}(X, f_1) \leftarrow \text{member}(X, g) \quad (27)
\]
\[
\neg \text{access}(X, f_2) \leftarrow \text{member}(X, g), \neg \text{access}(X, f_2) \quad (28)
\]

updated with

\[
\Pi_1 = \{ \text{member}(c, g) \leftarrow T \} \quad (29)
\]
\[
\neg \text{access}(X, f_1) \leftarrow \text{member}(X, g) \quad (30)
\]
\[
\text{access}(X, f_2) \leftarrow \text{member}(X, g), \neg \text{access}(X, f_2) \quad (31)
\]
According to [Zha06], this update ought to have the unique answer set

\[ S = \{ \neg \text{access}(a, f_1), \neg \text{access}(b, f_1), \neg \text{access}(c, f_1), \text{access}(a, f_2), \text{access}(b, f_2), \text{access}(c, f_2) \} \]

Then he claims that rule (31) should override rule (28). That is to say, [Zha06] states that there is information loss in some other semantics, but at the same time, his semantics says both \( b \) and \( c \) have access to \( f_2 \), ignoring the possible situation (world) when they explicitly do not. In fact, one might expect that \( b \) has no access to \( f_2 \) in \( \Pi_0 \) and that such a situation persists.

Regarding the controversy from this syntactical change of rule, his approach proposes a two-fold process of eliminating contradictory information, as well as resolution of conflicting rules and a final syntactic representation stage. Nevertheless, before the introduction of those two main processes, some fundamental definitions are in order.

The following definition can be seen as assuming true the given ground literals in \( S \) to \( \Pi \):

**Definition 10 (e-program, \( e(\Pi, S) \) [Zha06]).** Given a set of ground literals \( S \), \( e(\Pi, S) \) denotes the program obtained from program \( \Pi \) by deleting

1. each rule in \( \Pi \) that has a formula not \( \ell \) in its body with \( \ell \in S \), and
2. all formulas of form \( \ell \) in the bodies of the remaining rules with \( \ell \in S \).

The following example from [Zha06] illustrates the definition.

**Example 10 ([Zha06]).** Given \( S = \{ a, \neg b \} \) and the program

\[ \Pi = \{ c \leftarrow a \}
\]  
\[ \neg d \leftarrow \text{not} \ a \} \]

\( e(\Pi, S) = \{ c \leftarrow \top \} \).

This definition will prove useful to test a coherence concept in [Zha06]'s approach.

**Definition 11 (Coherence [Zha06]).** A set of ground literals \( S \) is coherent with an extended logic program \( \Pi \) if for any answer set \( S' \) of \( e(\Pi, S) \), \( S \cup S' \) is consistent.

By continuing Example 10, the only answer set of \( e(\Pi, S) \) is \( \{ c \} \). Thus, \( S \) is coherent with \( \Pi \).

As another example, let us consider [Zha06]’s: \( S = \{ a, \neg b \} \) is coherent with \( \Pi = \{ c \leftarrow a, \neg d \leftarrow \text{not} \ a \} \) because the only answer set of \( e(\Pi, S) \) is \( \{ c \} \). However, \( \{ a, \neg b, \neg c \} \) is not coherent with \( \Pi \).

These are basic steps towards a general proposal that consists in two main steps to perform an update of two programs. Firstly, eliminating contradictory rules from a previous program with respect to the latest one. Secondly, the semantics solves conflicts between the remaining rules of the programs. The semantics that determines the specifications of such an elimination and conflict resolution is Prioritised Logic Programs.
5.2 Prioritised Logic Programs

In order to specify the algebra for this logic program update proposal, [Zha06] employs an earlier platform called Prioritised Logic Programming [ZF97, Zha03], or simply PLP. Informally, this sort of logic programs consists of a set of preference relations and of a naming function that assigns a name to each rule.

Definition 12 (Prioritized Logic Program PLP [Zha06]). A prioritized logic program \( P \) is a triple \((\Pi, N, <)\), where \( \Pi \) is an extended logic program, \( N \) is a naming function mapping each rule in \( \Pi \) to a name, and \( "<" \) is a strict partial order on names. Moreover, \( \mathcal{P}(<) \) denotes the set of \(<\)-relations of \( \mathcal{P} \).

According to [Zha06], if \( N(\rho) < N(\rho') \) holds in \( \mathcal{P} \), rule \( \rho \) is preferred to be applied over rule \( \rho' \) while evaluating \( \mathcal{P} \). What is “evaluation” of \( \mathcal{P} \) anyhow? The following definitions code what an evaluation is. Meanwhile, it is worth recalling what an extended logic program is, before going any further:

Definition 13 (Extended Logic Program, ELP). An extended logic program is a set of rules of the form

\[
\rho_0 \leftarrow q_1, \ldots, q_m, \text{not } q_{m+1}, \ldots, \text{not } q_n
\]

where \( p_i \) and \( q_i \) are literals and \( m, n \in \mathbb{N} \).

Finally, the definition of a defeated rule looks as follows.

Definition 14 (Defeated rule [Zha03]). Let \( \Pi \) be a ground extended logic program and \( \rho \) a ground rule of form \((32)\) — \( \rho \) does not necessarily belong to \( \Pi \). Rule \( \rho \) is defeated by \( \Pi \) if and only if \( \Pi \) has an answer set and for any answer set \( S \) of \( \Pi \), there exists some \( \ell_i \in S \), where \( m+1 \leq i \leq n \).

For example, given a program

\[
\Pi = \{ a \leftarrow \top \}
\]

whose answer set is \( \{ a, d \} \), \( \Pi \) defeats rules like

\[
\bot \leftarrow \text{not } d;
\]

\[
a \leftarrow \text{not } a;
\]

\[
c \leftarrow b, \text{not } a, \text{not } d, \text{not } e
\]

Similarly to the case of extended logic programs, the evaluation of a PLP shall be on its ground form. Moreover, [Zha06] states that a PLP like \( \mathcal{P}' = (\Pi', N', <') \) is the ground instantiation of \( \mathcal{P} = (\Pi, N, <) \) if (1) \( \Pi' \) is the ground instantiation of \( \Pi \);
and (2) \( <' \) is a strict partial ordering and \( \mathcal{N}'(\rho'_1) <' \mathcal{N}'(\rho'_2) \in \mathcal{P}'(<' \) if and only if there exist rules \( \rho_1 \) and \( \rho_2 \) in \( \Pi \) such that \( \rho'_1 \) and \( \rho'_2 \) are ground instances of \( \rho_1 \) and \( \rho_2 \), respectively, and \( \mathcal{N}'(\rho_1) < \mathcal{N}'(\rho_2) \in \mathcal{P}(<) \).

**Definition 15** (Reduct \( \mathcal{P}^< \) [Zha03]). Let \( \mathcal{P} = (\Pi, \mathcal{N}, <) \) be a prioritized logic program. \( \mathcal{P}^< \) is a reduct of \( \mathcal{P} \) with respect to \( "<" \) if and only if there exists a sequence of sets \( \Pi_i, (i = 0, 1, \ldots) \) such that:

1. \( \Pi_0 = \Pi \).
2. \( \Pi_i = \Pi_{i-1} \setminus \{\rho_1, \rho_2, \ldots\} \) such that the following two conditions hold:
   
   (a) there exists \( \rho \in \Pi_{i-1} \) such that for every \( j \) (\( j = 1, 2, \ldots \)),
       \( \mathcal{N}(\rho) < \mathcal{N}(\rho_j) \in \mathcal{P}(<) \)
       and \( \rho_1, \rho_2, \ldots \) are defeated by \( \Pi_{i-1} \setminus \{\rho_1, \rho_2, \ldots\} \)
   
   (b) there are no rules \( \rho', \rho'', \ldots \in \Pi_{i-1} \) such that
       \( \mathcal{N}(\rho_j) < \mathcal{N}(\rho'), \mathcal{N}(\rho_j) < \mathcal{N}(\rho''), \ldots \)
       for some \( j \) (\( j = 1, 2, \ldots \)) and \( \rho', \rho'', \ldots \) are defeated by \( \Pi_{i-1} \setminus \{\rho', \rho'', \ldots\} \)
3. \( \mathcal{P}^< = \bigcap_{i=0}^{\infty} \Pi_i \).

In Definition 15, \( \mathcal{P}^< \) is an extended logic program that comes from \( \Pi \) by removing some defeated rules from \( \Pi \), by following the order relations in \( \mathcal{P}(<) \) on rules named by \( \mathcal{N} \). Specifically, if \( \mathcal{N}(\rho) < \mathcal{N}(\rho_1), \mathcal{N}(\rho) < \mathcal{N}(\rho_2), \ldots \) and \( \Pi_{i-1} \setminus \{\rho_1, \rho_2, \ldots\} \) defeats \( \{\rho_1, \rho_2, \ldots\} \), then, the rules \( \rho_1, \rho_2, \ldots \) will be out from \( \Pi_{i-1} \) unless a less preferred rule than can be removed in turn: conditions (2a) and (2b). One ought to compute the reduct procedure until a fixed point. Note that it is “less preferred” rather than the opposite for, at this stage, there is no update semantics.

In addition, condition (2b) in Definition 15 is necessary. In its absence, some counterintuitive results may be derived \( \rightarrow \) Zhang’s [Zha03]. For instance, consider the following example from the same author:

\[ \mathcal{P}_1 = (\Pi, \mathcal{N}, <): \]
\[ N_1 : \text{flies}(X) \leftarrow \text{bird}(X), \not\text{flies}(X) \]
\[ N_2 : \not\text{flies}(X) \leftarrow \text{penguin}(X), \not\text{flies}(X) \]
\[ N_3 : \text{bird}(\text{tweety}) \leftarrow \top \]
\[ N_4 : \text{penguin}(\text{tweety}) \leftarrow \top \]
\[ N_2 < N_1 \]

If one added the preference \( N_3 < N_2 \) in \( \mathcal{P}_1 \), then using a modified version of Definition 15 without condition (b),

\[ \{\text{flies}(\text{tweety}) \leftarrow \text{bird}(\text{tweety}), \not\text{flies}(\text{tweety}) \}
\]

\[ \text{bird}(\text{tweety}) \leftarrow \top \]
\[ \text{penguin}(\text{tweety}) \leftarrow \top \]
is a reduct of $P_1$, from which it concludes that Tweety flies. On the other hand, by
considering both the added preference and condition (2b) one will conclude that Tweety
does not fly from a unique reduct that lacks rule $N_3$.

Finally, an interpretation of a PLP is the answer sets of its reduct, as formally speci-
fied in the following definition.

**Definition 16** (Answer Set of $P$ [Zha03]). Let $P = (\Pi, N, <)$ be a PLP and $Lit$ the
set of all ground literals in the language of $P$. For any subset $S$ of $Lit$, $S$ is an answer
set of $P$ if and only if $S$ is an answer set for some reduct $P <$ of $P$.

Using Definition 15 and Definition 16 it is easy to conclude that $P_1$ has a unique
reduct as follows:

$$P_1^c = \{ \neg\text{flies}(\text{tweety}) \leftarrow \text{penguin}(\text{tweety}), \not\text{flies}(\text{tweety}) \\
\text{bird}(\text{tweety}) \leftarrow T \}
\text{penguin}(\text{tweety}) \leftarrow T \}
$$

from which we obtain the following answer set of $P_1$:

$$S = \{ \text{bird}(\text{tweety}), \text{penguin}(\text{tweety}), \neg\text{flies}(\text{tweety}) \}.$$

Let us analyse a complete example inspired from the same reference [Zha06], which
illustrates in detail when a PLP has more than one reduct. Before that, one should be
aware that a PLP may or may not have answer sets. In the first case, the program is
called well-defined program.

**Example 11** ([Zha06]). Suppose a PLP consisting of

$$P = (\Pi, N, >) := \{ N_1 : a \leftarrow T \\
N_2 : b \leftarrow \neg c \\
N_3 : d \leftarrow T \\
N_4 : c \leftarrow \neg b \\
N_1 < N_2, N_3 < N_4 \}$$

By Definition 15 one reduct is constructed as

1. $\Pi_0 = \Pi$

2. $\Pi_1 = \Pi_0 \setminus \{ b \leftarrow \not c \}$ because rule $a \leftarrow T \in \Pi_0$ and with its tag $N_3$, one finds
the relations $N_1 < N_2 \in P(<)$ and $b \leftarrow \not c$ is defeated by $\Pi_0 \setminus \{ b \leftarrow \not c \}$. 
Last, there are no rules $\rho', \rho'', \ldots \in \Pi_0$, whose tag is “greater than” $N_2$, and
defeated by $\Pi_0 \setminus \{ \rho', \rho'', \ldots \}$.

3. Finally, the reduct is the intersection of the two programs:

$$P(<) = \{ a \leftarrow T \hspace{1cm} d \leftarrow T \hspace{1cm} c \leftarrow \not b \}$$
Zhang’s line

and the other reduct as

1. $\Pi_0 = \Pi$

2. $\Pi_1 = \Pi_0 \setminus \{c \leftarrow \neg b\}$ because rule $d \leftarrow T \in \Pi_0$ and with its tag $N_3$ there is the relation $N_3 < N_4 \in P(<)$ and $c \leftarrow \neg b$ is defeated by $\Pi_0 \setminus \{c \leftarrow \neg b\}$ and there are no rules $\rho', \rho'', \ldots \in \Pi_0$, whose tag is “greater than” $N_4$, and defeated by $\Pi_0 \setminus \{\rho', \rho'', \ldots\}$.

3. Finally, $P(<) = \{a \leftarrow T \quad b \leftarrow \neg c \quad d \leftarrow T\}$

This section of Prioritised Logic Programs is the necessary background to give the interpretation of the following procedure for updates under the approach in [Zha06] that, as mentioned before, it consists in two main steps: contradiction elimination and conflict resolution.

5.3 Eliminating Contradictions

The first step in updating two extended logic programs under [Zha06]’s approach is eliminating contradictions by means of an extended simple-fact update program. Informally, the extended simple-fact program consist of establishing a high preference to inertia rules over update rules so that facts in the initial knowledge base may persist after an update. Then, it consists in interpreting the semantics of a resulting (possibly empty) update program(s) that should have a minimal difference with the answer sets of the original program. This interpretation of the update program(s) is the same as for PLP’s.

Before going straight to the main definition, some minor notation is necessary:

Definition 17 (Initial Knowledge; PLP Languages [Zha06]). It is stated that $B$ denotes an initial consistent knowledge base of ground literals of a language $L$; $\Pi$ an update extended logic program over $L$; and $L_{\text{new}}$ an extension to $L$, by propositional literals of the form $\text{new-}\ell | \ell \in L$.

[Zha06] represents an update program through a triple, and that program specifies the changes to an original knowledge base, according to a new update. Formally,

Definition 18 (U_{PLP}-specification, $U_{PLP}(B, \Pi)$ [Zha06]). Let $B$, $\Pi$, $L$, and $L_{\text{new}}$ be as above. The specification of updating $B$ with $\Pi$ is a PLP over $L_{\text{new}}$ denoted as $U_{PLP}(B, \Pi) = (\Pi^*, N, <)$, as follows:

1. $\Pi^*$ consists of following rules:

   **Initial knowledge rules:** for each literal $\ell$ in $B$, there is a rule $\ell \leftarrow T$
Inertia rules: for each predicate symbol $P \in \mathcal{L}$, there are two rules:

$$\text{new}-P(x) \leftarrow P(x), \neg \text{new}-P(x)$$

and

$$\neg \text{new}-P(x) \leftarrow \neg P(x), \text{not new}-P(x)$$

Update rules: for each rule $\ell_0 \leftarrow \ell_1, \ldots, \ell_m$, not $\ell_{m+1}, \ldots, \not\ell_n \in \Pi$ there is a rule

$$\text{new}-\ell_0 \leftarrow \text{new}-\ell_1, \ldots, \text{new}-\ell_m, \text{not new}-\ell_{m+1}, \ldots, \not\text{new}-\ell_n$$

2. Naming function $N$ assigns a unique name $N$ for each rule in $\Pi^*$.  

3. For any inertia rule $\rho$ and update rule $\rho'$, $N(\rho) < N(\rho')$.

According to Zhang [Zha06], an answer set of $\Pi^*$ represents a possible resulting knowledge base from the update of $B$ by $\Pi$, and a literal $\text{new}-\ell$ represents the persistence of $\ell$ if $\ell \in B$ or a change of $\ell$ if $\neg \ell \in B$ or $\ell \notin B$ with respect to the update. For instance, in Example 12 interpreting $\Pi^*$ corresponds to simple-fact update semantics, and it yields two answer sets: $\{\neg a, b, c, \text{new}a, \text{new}c, \neg\text{new}b\}$ and $\{\neg a, b, c, \text{new}a, \text{new}c, \text{new}b\}$, which means that the truth value of $b$ is indefinite by $\text{new}b$ with respect to the update: the new atom $\text{new}b$ is true in one answer set and false in the other.

Up to now, one can transform an initial knowledge base into an initial logic program and inertial rules together with the update rules to form a PLP. In addition, one can establish preference relations among PLP rules, in order to specify a $U_{PLP}(B, \Pi)$. Once one interprets a PLP, another definition is necessary to get the results of the specifications and to eliminate contradictions of the original sets of rules.

In general, the interpretations of an update program $U_{PLP}$ come from the answer sets of its corresponding PLP. Such an interpretation shall lead to one or more possible new knowledge bases, as expressed in the following definition.

**Definition 19** (Possible Resulting Knowledge Base, $S_{PLP}$ [Zha06]). Let $U_{PLP}(B, \Pi)$ be specified as in Definition 18. A set $B'$ of ground literals is called a possible resulting knowledge base with respect to $U_{PLP}(B, \Pi)$, if and only if $B'$ satisfies the following conditions:

---

8 A predicate symbol corresponds to an atom, in my notation.  
9 There is no formal specification when there exist strong-negated atoms. However, supported by the examples, one may state that for every strong-negated atom $\neg \ell$ in the formula, there is a strong-negated $\neg\text{new}-\ell$ atom.
1. if $U_{PLP}(B, \Pi)$ has a consistent answer set $S$, then $B' = \{ \ell \mid \text{new}\cdot \ell \in S \}$;

2. if $U_{PLP}(B, \Pi)$ does not have a consistent answer set (i.e., $U_{PLP}(B, \Pi)$ is not well defined), then $B' = B$.

The name $S_{PLP}(U_{PLP}(B, \Pi))$ denotes the set of all resulting knowledge bases of $U_{PLP}(B, \Pi)$.

Now, let us start with a simple but representative and thorough example (proposed by [Zha06]) that illustrates this process.

**Example 12 ([Zha06]).** Suppose the initial knowledge base $B = \{ \neg a, b, c \}$ and the update program

$$\Pi = \{ \neg b \leftarrow \neg b, a \leftarrow c \}$$

By Definition 18, the corresponding PLP specification is $U_{PLP}(B, \Pi) = (\Pi^*, N, <)$, consisting of the following rules:

**Initial knowledge:** $\Pi_0$:

$$\neg a \leftarrow \top, \quad b \leftarrow \top, \quad c \leftarrow \top$$

**Inertia rules:**

$$i_1 : \text{newa} \leftarrow a, \neg \text{newa}$$
$$i_2 : \neg \text{newa} \leftarrow \neg a, \neg \text{newa}$$
$$i_3 : \text{newb} \leftarrow b, \neg \text{newb}$$
$$i_4 : \neg \text{newb} \leftarrow \neg b, \neg \text{newb}$$
$$i_5 : \text{newc} \leftarrow c, \neg \text{newc}$$
$$i_6 : \neg \text{newc} \leftarrow \neg c, \neg \text{newc}$$

**Update rules:**

$$u_1 : \neg \text{newb} \leftarrow \neg \text{newb}$$
$$u_2 : \text{newa} \leftarrow \text{newc}$$

**Rule preferences** By Definition 18, $i_i < u_j$ with $i, j > 0$ are

$$N(i_1) < N(u_1), N(i_1) < N(u_2), N(i_2) < N(u_1)$$
$$N(i_2) < N(u_2), N(i_3) < N(u_1), N(i_3) < N(u_2)$$
$$N(i_4) < N(u_1), N(i_4) < N(u_2), N(i_5) < N(u_1)$$
$$N(i_5) < N(u_2), N(i_6) < N(u_1), N(i_6) < N(u_2)$$

INSTITUTE OF COMPUTER SCIENCE 26
With these specifications, one may compute the two answer sets of $\Pi^*$. Namely,

\[
\{\neg a, b, c, \text{newa, newc, newb}\} \quad (33)
\]

\[
\{\neg a, b, c, \text{newa, newc, } \neg \text{newb}\} \quad (34)
\]

However, the prioritised logic program $P$ has the unique answer set from its unique reduct because there is only one rule ($u_1$) defeated by $\Pi_0 \setminus \{u_1\}$ with which one may establish the relations $N(i_3) < N(u_1) \in P$ and there are no less-preferred rules than $u_i$ in $P(<)$ —Definition 15. As a consequence, $S_{PLP}(B, \Pi) = \{a, b, c\} = S_{(\Pi_0, \Pi_1)}$ as expected from Definition 19, and the transformed program from $\Pi_0$ with respect to $\Pi_1$ that is a maximal subset of $\Pi_0$ and is coherent with $S_{(\Pi_0, \Pi_1)}$ is just \{b, c\}. From this program, the $UZ(\Pi_0, \Pi_1)$ specification corresponds to the following $P$:

\[
\begin{align*}
\rho_1 : & \quad \neg b \leftarrow \text{not b} \\
\rho_2 : & \quad a \leftarrow c \\
\rho_3 : & \quad b \leftarrow \top \\
\rho_4 : & \quad c \leftarrow \top \\
\rho_1 < & \rho_3 \quad \rho_1 < \rho_4 \\
\rho_2 < & \rho_3 \quad \rho_2 < \rho_4
\end{align*}
\]

whose unique answer set out of the unique reduct is \{a, b, c\}. That is because defeated rules may not derive from a simple fact update, and here is the difference with the extended simple fact update.

This simple-fact update approach originally appeared in [MT94, MT98]. However, it is not adequate for practical applications for the simple reason that, as its name suggests, its definition does not deal with general (non-factual) rules. As a result, the author in [Zha06] reformulates the approach to allow updating ELP’s, rather than only facts, by means of the following two definitions, where the first one eliminates contradictory rules between $\Pi_0$ and $\Pi_1$.

**Definition 20** (Transformed Program, $\Pi(\Pi_0, \Pi_1)$ [Zha06].) Given two consistent programs $\Pi_0$ and $\Pi_1$, with $S_{\Pi_0}$ as an answer set of $\Pi_0$ and $S_{(\Pi_0, \Pi_1)}$ as an answer set of the update of $\Pi_0$ with $\Pi_1$. Suppose $S_{(\Pi_0, \Pi_1)} \in S_{PLP}(U_{PLP}(S_{\Pi_0}, \Pi_1))$. An extended logic program $\Pi(\Pi_0, \Pi_1)$ is called a transformed program from $\Pi_0$ with respect to $\Pi_1$, if $\Pi(\Pi_0, \Pi_1)$ is a maximal subset of the ground instantiation of $\Pi_0$ such that $S_{(\Pi_0, \Pi_1)}$ is coherent with $\Pi(\Pi_0, \Pi_1)$.

Once there is a transformed program, a set of preferences between its rules is to solve possible conflicts.

\[\text{Note that the original definition must have a typographical error when reading } S_{\Pi_0} \text{ rather than } S_{(\Pi_0, \Pi_1)}.]\]
Definition 21 (Update Specification, \( U_Z(\Pi_0, \Pi_1) \) [Zha06]). Let \( \Pi_{(\Pi_0, \Pi_1)} \) be defined as in Definition 20. A specification of updating \( \Pi_0 \) with \( \Pi_1 \) is a PLP, denoted as \( U_Z(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0, \Pi_1)}, \mathcal{N}, <) \), where, for each rule \( \rho \) in \( \Pi_1 \) and each rule \( \rho' \) in \( \Pi_{(\Pi_0, \Pi_1)} \), there is a preference relation \( \mathcal{N}(\rho) < \mathcal{N}(\rho') \).

Up to now, a transformed program can eliminate contradictions between an original knowledge base and its update. On the other hand, there are circumstances that do not cause contradiction, but indefinite results that must be observed.

5.4 Solving Conflicts

In the process of updating a knowledge base with a logic program, there are rules that might cause conflict when producing indefinite results. The way in which [Zha06] deals with this problem is by overriding old conflicting rules with the new ones, coded in the preferences of a transformed program, and by producing a called possible resulting program.

Definition 22 (Possible Resulting Program [Zha06]). A program \( \Pi_0' \) is a possible resulting program of \( U_Z(\Pi_0, \Pi_1) \) after updating \( \Pi_0 \) with \( \Pi_1 \) if \( \Pi_0' \) is a reduct of the ground instantiation of \( U_Z(\Pi_0, \Pi_1) \).

A mandatory test is Example 3, which produces counterintuitive results in many of the existing semantics for updates. So, I will compute it under [Zha06]'s approach as follows.

Example 13. Suppose an initial program

\[
\Pi_0 = \{ \text{day} \leftarrow \neg \text{night}, \text{night} \leftarrow \neg \text{day}, \text{stars} \leftarrow \text{night}, \neg \text{cloudy}, \neg \text{stars} \leftarrow \top \}
\]

updated with

\[
\Pi_1 = \{ \text{stars} \leftarrow \text{constellations}, \text{constellations} \leftarrow \text{stars} \}
\]

Its corresponding PLP specification, \( U_{PLP}(S_{\Pi_0}, \Pi_1) = (\Pi^*, \mathcal{N}, <) \), is as follows:

Initial Knowledge:

\[
i_0 : \text{day} \leftarrow \top
\]

\[
i_0 : \neg \text{stars} \leftarrow \top
\]
A ROADMAP OF UPDATING IN ASP

Inertial Rules:

\begin{align*}
i_1 : & \quad \text{newday} \leftarrow \text{day}, \neg \text{newday} \\
i_2 : & \quad \neg \text{newday} \leftarrow \neg \text{day}, \neg \text{newday} \\
i_3 : & \quad \text{newstars} \leftarrow \text{stars}, \neg \text{newstars} \\
i_4 : & \quad \neg \text{newstars} \leftarrow \neg \text{stars}, \neg \text{newstars} \\
i_7 : & \quad \text{newconstellations} \leftarrow \text{constellations}, \neg \text{newconstellations} \\
i_8 : & \quad \neg \text{newconstellations} \leftarrow \neg \text{constellations}, \neg \text{newconstellations}
\end{align*}

Update Rules:

\begin{align*}
u_1 : & \quad \text{newstars} \leftarrow \text{newconstellations} \\
u_2 : & \quad \text{newconstellations} \leftarrow \text{newstars}
\end{align*}

Rule Preferences:

\begin{align*}
N(i_1) < N(u_1) & \quad N(i_1) < N(u_2) \\
N(i_2) < N(u_1) & \quad N(i_2) < N(u_2) \\
\vdots & \quad \vdots \\
N(i_8) < N(u_1) & \quad N(i_8) < N(u_2)
\end{align*}

where its unique \( S_{PLP}(U_{PLP}(S_{\Pi_0}, \Pi_1)) \) = \( \{ \text{day}, \neg \text{stars} \} \). In this case, \( \Pi_{(\Pi_0, \Pi_1)} \) coincides with \( \Pi_0 \) because \( S_{PLP}(U_{PLP}(S_{\Pi_0}, \Pi_1)) \) is coherent with \( \Pi_0 \)—resp. \( \Pi_{(\Pi_0, \Pi_1)} \), where

\[
e(\Pi_0, S_{PLP}(U_{PLP}(S_{\Pi_0}, \Pi_1))) = \{ \text{day} \leftarrow \text{not night} \\
\text{stars} \leftarrow \text{night}, \text{not cloudy} \\
\neg \text{stars} \leftarrow \top \}
\]

and its answer set is \( \{ \text{day}, \neg \text{stars} \} \), which is consistent with \( S_{PLP}(U_{PLP}(S_{\Pi_0}, \Pi_1)) \).

Thus, \( \Pi_{(\Pi_0, \Pi_1)} \) is a maximal subset of \( \Pi_0 \).

Finally, its update specification \( U_Z(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0, \Pi_1)}, N, \prec) \), whose possible resulting program is just

\[
\Pi_1 \cup \Pi_{(\Pi_0, \Pi_1)} \setminus \{ \text{night} \leftarrow \neg \text{day} \} = \\
\{ \text{stars} \leftarrow \text{constellations} \\
\text{constellations} \leftarrow \text{stars} \\
\text{day} \leftarrow \text{not night} \\
\text{stars} \leftarrow \text{night}, \text{not cloudy} \\
\neg \text{stars} \leftarrow \top \}
\]

with its expected answer set \( \{ \text{day}, \neg \text{stars} \} \).
Conclusions

Despite this nice behaviour, one of the counter-intuitive examples to Zha06’s approach has to do with solving conflicts between rules and not with models, where most of the current semantics differ:

Example 14. Suppose an initial knowledge base $\Pi_0 = \{p \leftarrow \textit{not } q\}$ being updated with $\Pi_1 = \{q \leftarrow \textit{not } p\}$. The update specification corresponds to $U_{\text{PLP}}(S_{\Pi_0}, \Pi_1) = (\Pi^*, N, <)$, where

Initial Knowledge:

\[ i_0 : \ p \leftarrow \top \]

Inertial Rules:

\[ i_1 : \ \textit{newp} \leftarrow p, \textit{not } \textit{newp} \]
\[ i_2 : \ \textit{not } \textit{newp} \leftarrow \textit{not } p, \textit{not } \textit{newp} \]
\[ i_3 : \ \textit{newq} \leftarrow q, \textit{not } \textit{newq} \]
\[ i_4 : \ \textit{not } \textit{newq} \leftarrow q, \textit{not } \textit{newq} \]
\[ i_5 : \ \textit{newp} \leftarrow p, \textit{not } \textit{newp} \]
\[ i_6 : \ \textit{not } \textit{newp} \leftarrow \textit{not } p, \textit{not } \textit{newp} \]

Update Rule

\[ u_1 : \ \textit{newq} \leftarrow \textit{not } \textit{newp} \]

Preferences

\[ N(i_1) < N(u_1) \]
\[ N(i_2) < N(u_1) \]
\[ N(i_3) < N(u_1) \]
\[ N(i_4) < N(u_1) \]
\[ N(i_5) < N(u_1) \]
\[ N(i_6) < N(u_1) \]

So, $S_{\text{PLP}}(U_{\text{PLP}}(S_{\Pi_0}, \Pi_1)) = \{p\}$ and $\Pi_{(\Pi_0, \Pi_1)} = \Pi_0$, where $U_{\text{Z}}(\Pi_0, \Pi_1) = (\Pi_1 \cup \Pi_{(\Pi_0, \Pi_1)}, N, <)$ whose reduct $q \leftarrow \textit{not } p$ is the most preferred one and does not coincide with our intuition.

Last, besides not satisfying some of the postulates already pointed out, the major drawback of this approach is being limited to only one update to a knowledge base. Namely, it is undefined neither for update sequences nor for successive updates, which does not seem to lead to immediate practical applications.
6 Conclusions

This report has presented a survey of semantics in ASP that have made long way by going from simple-fact updates of logic programs, to updates of unlimited programs in a sequence. Some of these works present a vast collection of postulates and principles, and/or implementation. However, all the proposals here introduced still present drawbacks either for limiting to one-step update, or for relying on syntactical changes to the original logic program that leads to counterintuitive results, which suggests challenging areas of research.

7 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


References


A ROADMAP OF UPDATING IN ASP


References


A ROADMAP OF UPDATING IN ASP
