Consistent Transformations of Knowledge Bases in Answer Set Programming

J.C. Acosta-Guadarrama, IIT, UACJ; LANTI, Mexico

Abstract—One of the major and traditional topics of Artificial Intelligence over many years has been knowledge representation and reasoning. It has proved to be a strong theoretical framework for Logic Programming to manage dynamic knowledge bases. In this work, one of two parts, we go through current and some of those past proposals to update knowledge coded into Answer Set Programming knowledge bases, by analysing their features and identifying challenges to represent correct evolving knowledge.

Index Terms—Consistency, Belief Revision, Belief Update, Knowledge Bases.

1 INTRODUCTION

There is a preliminary version of this paper has first appeared as a technical report in [3]. Here we revisit it and update it to current approaches, to classify them and compare them.

One of the significant and traditional topics in Artificial Intelligence over the last years has been knowledge representation and reasoning; this issue has proved to be a robust theoretical framework to manage knowledge bases. This particular topic has become more widely studied in the administration of knowledge bases of intelligent (rational) agents, especially in situations of incomplete knowledge from a changing environment. This area of research is known in the literature as belief change, and its relevance to program transformation is fundamental if we are interested in producing correct knowledge bases, especially if they represent critical systems requirements.

The history of semantics for updates of logic programs (in our context, formalisms for program transformations) is rather long. Indeed, it begins in the days of some of the first versions of PROLOG with its commands assert and retract. However, sooner they began to yield conflicting information and other (perhaps unexpected) side effects. It was also time of research on databases with publications like [21], and in particular for logical databases [19], [20], [36]. Nevertheless, some of the first formalisms to carry out proper changes to monotonic theories have been originally studied by [6], [23], [25], [27], [29], whereas in the non-monotonic side by [26], [28], [30].

In particular, [22] 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 represented in ASP [2], [9], [14], [15], [16], [17], [32], [34], [35], [38], [39], [42].

In this work we introduce current and some of those past proposals to transform knowledge bases represented by logic programs. We point out features as well as some of their limitations to represent correct evolving knowledge and of course, correct program transformations. Nevertheless, this survey is just a small thread of a massive research over more than two decades, and is by no means exhaustive. It just takes into account those proposals that are the most relevant and of interest in our opinion.

The rest of our paper is divided into a very quick glimpse of basic background (Section 2) necessary to understand program transformations in ASP; the approaches to update logic programs are classified into several different categories (Sections 3–6) and a section for discussion and final remarks—Section 7. Each of the approaches show a few particular examples to illustrate their definitions, as well as common observations to show disadvantages and to compare with the others.

2 PRELIMINARIES

A main foundation of these proposals is the well-known Answer Sets Semantics, also known as Stable-Models Semantics. In this paper it is assumed, though, that the reader is familiar with basic notions of logic programming and (extended) disjunctive logic programs, DLP, EDLP, which are easily available in the literature.

2.1 Logic Programming and Answer Sets

As we represent knowledge by means of ASP programs for being one of the most studied and founded successful semantics to reason about incomplete (unknown) information, in the following we give a very-short description of Answer Sets Programming (ASP), which is identified with other names like Stable Logic Programming or Stable Model Semantics [22] and A-Prolog. Its formal language and some more notation are introduced from the literature as follows.

Definition 1 (ASP Language, \( L_{ASP} \)). In the following \( L_{ASP} \) is a language of propositional logic with sym-
bols: \(a_0, a_1, \ldots\); connectives: “,” (conjunction) and meta-
connective “○”; disjunction \(\lor\), also denoted as \(\mid\); \(\leftarrow\)
(and its counterpart, \(\rightarrow\)); \(\neg\) (default negation or weak
negation, also denoted with the word not); “¬” (strong
negation, equally denoted as “¬¬”). The propositional
symbols are also called atoms or atomic propositions.
A literal is an atom or a strong-negated atom. A rule \(\rho\) is
an ordered pair \(H(\rho) \leftarrow B(\rho)\), where \(H(\rho)\) is a possibly-
empty finite set of literals in disjunction and \(B(\rho)\) a
possibly-empty finite set of literals (or default-negated
literals) in conjunction.

The meaning we give the propositional constants is the
same meaning than having an empty set in either compon-
ent of a rule. Finally, a logic program (or just program) is a
possibly empty finite set of rules, also known as knowledge
base.

With the notation just introduced in Definition 1, one
may construct program clauses of several forms that are well
known in the literature, such as Extended Logic Program
\((ELP)\), Extended Disjunctive Logic Program \((EDLP)\), etc.

Informally, the semantics of such programs consists of
reducing the general rules to rules without default negation
“¬”, because the latter are universally well understood. For
page limitation, we just skip the formal definition of such
reduct, which can be easily found in the literature.

Formally, we say that a program is inconsistent if and
only if it has no answer sets, and consistent otherwise.

**Definition 2 (EDLP).** An extended disjunctive logic program
is a set of rules of form

\[
\ell_1 \lor \ell_2 \lor \ldots \lor \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \neg \ell_{m+1}, \ldots, \neg \ell_n
\]

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

Naturally, an extended logic program \((ELP)\) 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 \(l = 0\); while a fact is a rule of the
same form with \(l = m = n\). In particular, given a set of
literals \(A\), for a literal \(\ell \in A\), the complementary literal is \(\sim \ell\)
and vice versa; for a set \(M\) of literals, \(\sim M = \{ \sim \ell \mid \ell \in M\}\),
and \(Lit_M\) denotes the set \(M \cup \sim M\); finally, a signature \(\Sigma_K\)
is a finite set of literals occurring in a knowledge base, \(K\).
Additionally, given a set of literals \(M \subseteq A\), the complement
set \(\overline{M} = A \setminus M\).

The well-known semantics of an EDLP consists of redu-
cing general rules to rules without default negation “¬”
because the latter can be interpreted in classical logic
by means of the well-known Herbrand models. In particu-
lar, the reduced rules with no default negation \(Mon\) of a rule of the
form (1) is

\[
\ell_1 \lor \ell_2 \lor \ldots \lor \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m
\]

where \(\ell_i\) are literals and \(0 \leq l \leq m\). This kind of rules is
known in the literature as monotonic counterpart or positive
program. Additionally, the monotonic counterpart of a set of
rules is the set of the monotonic counterparts of its rules.

Now let us introduce the meaning of programs with both
monotonic and nonmonotonic counterparts.

Suppose a finite ground program \(K\), consisting of clauses
of form (1). For any set \(\mathcal{S} \subseteq \Sigma_K\), the answer-sets reduct \(K^S\)
corresponds to

\[
K^S = \{ \ell_1 \lor \ell_2 \lor \ldots \lor \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m : \\
\{\ell_{m+1}, \ldots, \ell_n\} \cap \mathcal{S} = \emptyset \}
\]

Stating \(\mathcal{S}\) as a set of literals rather than atoms, makes one of
the differences with Stable-models semantics.

Next, the meaning of a monotonic counterpart corre-
ponds to its minimal classical model as follows.

**Definition 3 (Minimal Closure, \(\text{Cn}(K)\)).** Let \(K\) be a positive
extended disjunctive program and \(\Sigma_K\) the signature (set of
all ground literals) from \(K\). The set \(\text{Cn}(K)\) denotes the
minimal subset of \(\Sigma_K\) where,

1) for each ground clause \(p_0 \lor p_1 \lor \ldots \lor p_l \leftarrow q_1, \ldots, q_m\)
in \(K\), \(q_1, \ldots, q_m \in \mathcal{S}\) implies \(p_l \in \mathcal{S}\) for some \(0 \leq i \leq l\); and for each ground clause of the form

\[
\bot \leftarrow q_1, \ldots, q_m
\]

2) If \(S\) contains a pair of complementary literals, then
\(S = \Sigma_K\).

Note that item (2) in Definition 3 extends Stable Models by
giving a meaning to strong negation.

Finally, an answer set of a given program \(K\) is a minimal
reduct of its reduct as following stated.

**Definition 4 (Answer Set).** Suppose \(K\) is a EDLP and \(\mathcal{S}\)
a set of literals. Then, \(\mathcal{S}\) is an answer set of \(K\) if and only if
\(\mathcal{S} = \text{Cn}(K^S)\).

Notice that all stable models can be viewed as minimal
Herbrand models of a set of first-order sentences, but not the
converse. Additionally, \(\mathcal{S}\) is a consistent answer set of a given
program \(K\) if it does not contain a complementary pair of
literals.

Although we have introduced ASP as propositional
(ground) programs, fixed non-ground ASP-programs of arbiter
arity are also considered in the same way than [13] do.
Accordingly, non-ground ASP-programs with variables
or constants as arguments can be seen as a simplified expres-
sions of larger ground (propositional) ones without
variables, where each ground program \(K\) is a set of its ground
rules \(\rho \in K\). In addition, a ground rule is the set obtained
by all possible substitutions of variables in \(\rho\) by constants
occurring in \(K\) [13].

In general, ASP is the necessary background and main
foundation that is common to all the approaches here
presented. Yet another framework employed by a few of the
approaches to update knowledge in ASP is called General-
ized Answer Sets.

This is the basic background to understand the follow-
ing approaches to update knowledge represented in ASP
programs. So, let us begin with the different proposals.

3 Eiter’s Team

To the best of our knowledge, [17] achieved the most com-
plete survey of most known semantics for updates of logic
programs, by gathering relevant postulates and principles
from the literature. Their approach first appeared in [14] with a vast study of well-known and well-accepted postulates and properties, and later refined [17] and extended to be a main component in more general problems like agents [18] or preferences. They also implemented a solver\(^2\) that is the main engine of an experimental graphical front end we have implemented\(^3\).

In [17] they 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 [9] that later, however, turned out to be counterintuitive. See [8], [18], [32]. The problem comes up from a strong dependency upon the syntax of programs, first noted by [18], [33], [37], and is further discussed in Section 4. In particular, the formula under which they, [17], [18], analyze and describe update properties is as follows.

Given an update sequence \((K_1, K_2, \ldots, K_n)\), with \(2 \leq n\), over a set of atoms \(A\), assume \(A_n\) as an extension of \(A\) by new pair-wise unique atoms \(\text{rej}(\rho)\) and \(\alpha_i\), for each rule \(\rho\) occurring in \(K_i\), each atom \(\alpha \in A\), and \(1 \leq i \leq n\). An injective naming function \(\text{Name}(-, -)\) is also assumed, which assigns to each rule \(\rho\) in a program \(K_i\), a unique name, \(\text{Name}(\rho, K_i)\), provided that \(\text{Name}(\rho, K_i) \neq \text{Name}(\rho', K_j)\) whenever \(i \neq j\). Finally, for a literal \(\ell\), they use \(\ell_i\) to denote 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 the former is true, provided that there is another more recent rule \(\rho'\) with conflicting information.

Then [17] define the intended answer sets of an update sequence \((K_1, K_2, \ldots, K_n)\) in terms of the answer sets of \(K_n = (K_1 < \cdots < K_n)\). In other words, the models are back expressed in the original alphabet by the intersection of them and the original atoms:

**Definition 5 ([17])**. Given an update sequence, \((K_1, K_2, \ldots, K_n)\) over a set of atoms \(A\), then \(S \subseteq \text{Lit}_A\) is an update answer set of \((K_1, K_2, \ldots, K_n)\) if and only if \(S = S' \cap A\) for some answer set \(S'\) of \(K_n = (K_1 < \cdots < K_n)\). The collection of all the update answer sets of \((K_1, K_2, \ldots, K_n)\) is denoted by \(U(K_1, K_2, \ldots, K_n)\).

In addition to the declarative version, this semantics is also supported by a solver available both for downloading and running online\(^4\), which is yet another valuable asset worth considering when comparing the approach with others.

Let us present the following example, inspired in [17], to illustrate their approach. Later we will reuse it with other approaches:

**Example 1.** Suppose we have the following simple-but-illuminative two sets of system requirements (translated into ASP), \((K_1, K_2)\), representing the initial and current knowledge of an intelligent greenhouse, which acts autonomously under specific circumstances, where

\[ K_1 = \{ \text{notify} \leftarrow \text{night}, \text{not wSystem} \} \]
\[ wPlants \leftarrow \text{wSystem} \]
\[ K_2 = \{ \text{not wSystem} \leftarrow \text{blackout} \} \]

Program \(K_1\) might represent the following configuration:

- Notify when it’s night and there isn’t evidence of the water system working.
- It’s night now.
- Water the plants when the water system is working.
- The system is working now.

The unique model of such requirements is

\[ \{ \text{night}, \text{wPlants}, \text{wSystem} \} \]

Now suppose that the systems engineer needs to incorporate a new rule that states to not water plants when the ground is flood: \(\sim \text{wPlants} \leftrightarrow \text{gFlood}\). By their definition, the transformed update program \(K_3 = (K_1 < \cdots < K_3)\) consists of the following rules—amongst the rest of the rules that we skip for page constraints:

\[ \cdots \sim \text{wSystem}_1 \leftarrow \text{not rej}(\rho_4) \]
\[ \sim \text{wSystem}_2 \leftarrow \text{blackout}, \text{not rej}(\rho_5) \cdots \]
\[ \cdots \text{rej}(\rho_5) \leftarrow \text{not wSystem}_2 \cdots \]
\[ \text{notify}_1 \leftarrow \text{notify}_2, \text{not rej}(\rho_5) \cdots \]
\[ \sim \text{wSystem}_1 \leftarrow \text{wSystem}_2, \text{wSystem} \leftarrow \text{wSystem}_1 \cdots \]
\[ \sim \text{wSystem}_3 \leftarrow \sim \text{wSystem}_4, \sim \text{wSystem} \leftarrow \sim \text{wSystem}_2 \]
\[ \text{blackout}_2 \leftarrow \text{blackout}_3, \text{blackout} \leftarrow \text{blackout}_2 \]

whose unique answer set is

\[ \{ \text{notify}_1, \text{night}, \text{night}_1, \text{rej}(\rho_4), \sim \text{wSystem}_3, \text{blackout}, \text{blackout}_2, \text{notify}, \sim \text{wSystem} \} \]

and its update answer set is just: \(\{ \text{night}, \text{blackout}, \text{notify}, \sim \text{wSystem} \}\).

Let us consider Example 1 again and perform a second update to the sequence with program \(K_3 = \{ \sim \text{blackout} \}\). Accordingly, the new answer set of the resulting update program is

\[ \{ \text{wSystem}_1, \text{wSystem}, \text{night}_1, \text{night}, \text{wPlants}_1, \text{wPlants}, \text{rej}(\rho_6), \sim \text{blackout}_3, \sim \text{blackout}_4 \} \]

As a result, by Definition 5 the corresponding update answer sets are

\[ U(K_3, K_2) = \{ \text{night}, \text{blackout}, \text{notify}, \sim \text{wSystem} \} \]

\[ U(K_3, K_2, K_3) = \{ \text{wSystem}, \text{night}, \text{wPlants}, \sim \text{blackout} \} \]

However, this result does not coincide with common intuition, just because one of the possible models contradicts the latest fact (\(\sim \text{blackout}\)) stating that it is no longer happening! On the other hand, the second model says that system is working, back again for no obvious reason, which is counterintuitive too.
Despite the satisfactory deep nice analysis they realize of known postulates and principles from the literature and their available solver, yet another major disadvantage of [17]'s approach has to do with syntactic and semantic contents, as illustrated by the following example inspired from [8] and modified by [37] that may produce counterintuitive models:

**Observation 2.**

Suppose an agent who believes that when it is day it is not 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 simple story may be coded\(^5\) into a logic program \(K_1\) as follows:

\[
K_1 = \{(\text{day} \leftarrow \neg \text{night}), \quad (\text{night} \leftarrow \neg \text{day}) \\
(\text{stars} \leftarrow \text{night}, \neg \text{cloudy}), \quad \neg \text{stars}\}
\]

whose unique answer set is \(\{\text{day}, \neg \text{stars}\}\). Later, the agent acquires new information stating that stars and constls\(^6\) are the same thing, as coded in \(K_2\). As soon as the agent updates \(K_1\) with program

\[
K_2 = \{(\text{stars} \leftarrow \text{constls}), \quad (\text{constls} \leftarrow \text{stars})\}
\]

the augmented alphabet of the two programs contains only one new extra atom with respect to \(K_1\): constls. As the model of \(K_2\) is obviously the empty answer set, constls is considered synonym of stars by means of \(K_2\), and thus the update should not change the original beliefs. However, the update yields an extra answer set in some of the existing update semantics based on the causal rejection principle see [8], [9], [17]: \(\{\text{stars, constls, night}\}\), which does not coincide with common intuition.

The reason is that, although stars can not be true, introducing constls gives another possibility for stars to be true. Thus, the additional answer set is derived [37].

In general, these supplementary rules in the update are a conservative extension [31] to \(K_1\): the original language is extended and all answer sets ought to be extensions of the old answer sets. In this specific situation, constls should be true if and only if stars is true.

So, Observation 2 means that the proposed semantics is inappropriate to model the corresponding kind of problems.

Finally, updating knowledge through a sequence of logic programs does not seem to be natural in our opinion. For instance, consider Observation 1 again and try to perform a new update. Although the authors might have had other goals and intentions, their current approach does not allow to do that (i.e. you would have to ”initialize” the sequence and append the new update to it.

To recap, [17] were very good at gathering relevant postulates and principles from the literature and at analysing them in terms of their proposal. Their approach, however, suffers from a few disadvantages for our interests owing to its reliance on the causal rejection principle (see [9]) and to its sequence-based approach.

---

5. Notice that there are other ways to represent the story. The problem is, however, what to do in this particular situation, when the agent's original knowledge base runs across a redundant piece of information.

6. i.e. constellations.

---

4 **DyLP and Other Early Proposals**

One of the earliest approaches to update logic programs through transformations appeared in the late 90’s from [9], [10] that was extended to an interesting language called LUPS, by [11], to specify explicit updates in programs on a semantics that they called Dynamic Logic Programming or DyLP [9]. Some years later, though, [8] refined the latter, whom over the previous periods formulated a principle of rejection, also known as causal rejection principle [8], [9], [17] and [7].

Informally, the called refined principle of rejection consists in rejecting rules of previous and upcoming programs in an update sequence, whenever there are other conflicting rules at the current state.

To begin with, [8] motivation comes from a simple example to what they themselves called a tautology (a rule from which they expect no models):\(^7\)

\[
\neg p \leftarrow \not \neg p
\]

One may verify that the rule alone produces no models in their semantics\(^8\)—i.e. just the empty model, \(\emptyset\), which actually contains all non-positive (default-negated) atoms—and updating knowledge with this rule should have no effect, and that is why they introduced the refined principle of rejection.

Additionally, [8] explain in a footnote what tautology means: A rule of the form \(\ell \leftarrow B\) with \(\ell \in B\), where \(\ell\) is an atom (or a default-negated atom) and the body of a rule, respectively. Although the authors might have had other goals for their approach, this high dependency on syntax will prove to be one of their major disadvantages as a semantics for updates in our opinion, as explained along this paper.

Before introducing their proper definitions for their semantics, a special notation is necessary, which can be obtained from the literature and from [8].

Intuitively, a refined interpretation of a DyLP program is a dynamic stable model if the following happens. The least model of the positive program, which results from the difference of the rejected rules and the union of the default assumptions, is the same than the union of the interpretation and the default-negated literals that do not appear in the latter.

Definition 6 introduces the model of the transformed program from the original dynamic logic program. The intuition behind Rej\((\cdot, \cdot)\) is the set of rules that conflict \(\triangleright\) with both current and previous ones in the sequence. Moreover, Def\((\cdot, \cdot)\) consists of the positive “default-negated” atoms that do not appear in the intended model.

Formally, two rules, \(\rho_1, \rho_2\), are in conflict, denoted as \(\rho_1 \triangleright \rho_2\), if and only if \(H(\rho_1) = \not B(\rho_2)\).

---

7. Note that the kind of rule in (5) is invalid in \(\mathcal{LP}_{\triangleright}\), and it also illustrates why strong negation “\(\neg\)~” should not be a simple replacement to “\(\not\)~” in the head. Take for example, the program \(\{\neg p \leftarrow \not p\}\) whose unique answer set differs from the empty set: \(\{\neg p\}\).

8. The details are easily available in the literature, like [9].
Definition 6 (Dynamic Stable Model, [8]). Let \( P \) be a dynamic logic program and \( M \) an interpretation. \( M \) is a dynamic stable model of \( P \) if and only if it satisfies:

\[
\text{Rej}(P, M) = \{ \rho \mid \rho \in K_i, \exists \rho' \in K_j, i \leq j, \rho \gg \rho', M \models B(\rho') \};
\]
\[
\text{Def}(P, M) = \{ \text{not}_a \mid \exists \rho \in \rho(\rho), H(\rho) = a, M \models B(\rho) \};
\]
\[
\overline{M} = \text{least}(\rho(P) \setminus \text{Rej}(P, M) \cup \text{Def}(P, M)).
\]

This approach has had several implementations, including one for the original version before the so-called refined principle, and another for the principle itself. Additionally, LUPS is also implemented, which we consider a major asset, and the following list shows their respective locations:

- http://centria.di.fct.unl.pt/~jia/updates
- http://centria.di.fct.unl.pt/~banti/
- FedericoBantiHomepage/refdlp.htm

Now let us analyze an example inspired by [35], originally proposed by [9].

Observation 3. The above framework has missing information. Let us code the story introduced in Example 1 and Observation 1 as follows: \( P = K_1 \oplus_R K_2 \oplus_R K_3 \), where:

\[
K_1 = \{ (\text{notify} \leftarrow \text{not wSystem}), (\text{wPlants} \leftarrow \text{wSystem}) \},
\]
\[
K_2 = \{ (\text{blackout}), (\text{not wSystem} \leftarrow \text{blackout}) \},
\]
\[
K_3 = \{ \text{not blackout} \}.
\]

\( M = \{ \text{wSystem}, \text{wPlants} \} \) is a refined dynamic stable model of the update sequence, since the unique rejected rule is \( \text{blackout} \) and the unique default is \( \text{not NOTIFY} \), where the least model is

\[
\{ \text{wPlants, wSystem, not blackout, not notify} \} = \overline{M}.
\]

In a similar example [35] argue, however, that the resulting model is counterintuitive, because after the first update, the refined dynamic stable model was \{ \( \text{blackout, notify} \) \} and thus there is no reason to believe that once the power is restored, the system should be working back again! A similar counterintuitive result is given in Observation 1.

In addition to that, there is still a simple example, first suggested by [17], that still causes counterintuitive results in this DyLP-semantics.

Observation 4. Suppose an initial knowledge base \( K_0 = \{ (c \leftarrow r), (r) \} \) updated with \( K_1 = \{ \text{not r} \leftarrow \text{not c} \} \). Firstly, the initial generalized stable model of \( K_0 \) is \{ \( c, r \) \}, and one would expect no further changes after the update because, according to [17], such an update should be irrelevant to the known fact \( r \) and the derived \( c \). However, the update \( P = K_0 \oplus_R K_1 \) has the extra model \( M = \{ \text{not}_c, \text{not}_r \} \) because

\[
\text{Rej}(P, M) = \{ r \}; \text{Def}(P, M) = \{ \text{not}_c \}; \text{and least}(K_0 \cup K_1) \setminus \text{Rej}(P, M) \cup \text{Def}(P, M) = \{ \text{not}_c, \text{not}_r \} = \overline{M}.
\]

Although [9] were some of the first researchers to formulate and implement a semantics for updates through program transformations, they still have quite a few disadvantages like the ones pointed out in this section: Firstly, for the particular syntax and semantics of their transformed generalized logic programs, which is a different variant of Stable-Models Semantics, or in other words, a non-standard concept of ASP [17]; secondly, for their causal rejection principle that produces the mentioned counterintuitive results. Finally, their sequence-of-knowledge-bases approach is in our opinion counterintuitive, as we have already commented at the end of Section 3.

5 Minimal Changes

In [34], [35] they propose three types of updates through program transformation, to which they call inconsistency removal, view updates and theory updates. Each of them corresponds to a special case of updates and revision in the literature.

For page and comparison reasons, this paper is focused on theory updates, rather than other special cases such as making an inconsistent program consistent or differentiating between variant and invariant knowledge. As a result, this section does not include other types of belief changes, which are easily available in the literature.

Once the extended abductive program is normalized, its interpretations shall correspond to transformed update programs that consist of the rules of the original theory that don’t belong to the normalized abductive set, merged with a new set of update rules, as specified in [35].

Next, this transformation of update rules takes part of a new transformation called update program of the normalized extended abductive program that is an intermediate EDLP. This intermediate program specification is as follows, where \( P \) is an EDLP over \( \Lambda \), and \( \Lambda_0 \) a set of abducibles, such that \( \Lambda \cap \Lambda_0 = \emptyset \):

Definition 7 (Update Program, [35]). Given an abductive program \( \langle P, \Lambda_0 \rangle \), its update program, \( UP \), is defined as an EDLP such that \( UP = (P \setminus \Lambda_0) \cup UR \).

Then the models of an update program denote the deletion of facts or rules from the original program in the pair.

Definition 8 (U-minimal Answer Sets, [35]). An answer set \( S \) of \( UP \) is called U-minimal (U-MAS) if there is no answer set \( S' \) of \( UP \) such that \( S' \cap \Lambda \alpha \subset S \cap \Lambda \alpha \).

As a result, there is one or more new corresponding updated programs to the U-MAS.

5.1 Discussion

This abduction framework proves to have nice properties of a syntactical minimal change of rules in the original non-monotonic theory—Theory Updates Definition in [35]—by means of consistent interpretations of hypothetical changes. With this framework, [35] can perform particular kinds of updates and maintenance of knowledge-bases consistency, and can also present a vast analysis of disadvantages in other comparable approaches.
In general, [35]’s first goal is to provide an update semantics to compute their extended abduction. Secondly, they also characterize updates through the extended abduction, as they themselves state it. Consequently, the approach lacks of a proper analysis of more principles and postulates from the literature of program transformation or belief change. Additionally, they characterize 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 recapitulate [35]’s approach in a few words. They construct their update program out of the normal abductive form of an extended abductive program \( \langle P \cup Q, P \setminus Q \rangle \) whose models are U-MAS’s interpreted from an update program. Last, the interpretations correspond to one (or more) new programs representing knowledge bases, derived from the addition and/or deletion of facts that the U-MAS’s describe in turn.

In [35] they propose an example as an argument against other approaches—like [9], [17]—that brings back previous knowledge of the original theory. That is to say, their interpretation is that a TV in their example\(^{10}\) turns itself on again and it is possible to watch it as well with no reason to do so: \( \{ \text{tvon, watchtv, \sim blackout}\} \), which does not coincide with their intuition. This is similar to our conclusions in Observation 1, where the system is working back again upon no justification. However, this argument seems to be too strong to generalize that all update semantics should behave accordingly, because [35] are differentiating fluents and actions in a language that does not have such an explicit difference.

Finally, there is a simple example that might represent another disadvantage of this approach.

**Observation 5.** Suppose the initial knowledge base \( K = \emptyset \) updated by a simple fact \( K_1 = \{x\} \). Following [35]’s framework, the answer sets of its update program is empty: \( U^P = (\emptyset \setminus \{x\}) \cup UR \), where \( UR \) is clearly empty because the extended abductive program from the update pair has no abducibles: \( A_\alpha = \emptyset \setminus \{x\} \).

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 and a lack of a solver, which make it hard to compare with other alternate approaches. [42] pointed out that this approach is classified into a syntax-based semantics. As a result, it has no general semantic foundation that justifies its updates [42], and by interpreting the resulting knowledge bases with a given semantics might interfere with the ASP semantics that performs the update operation. It is clear that they justify their updates with an extended abductive framework, which is still a specific problem and then leaves the mentioned absence of update characterisation.

Finally, a minor disadvantage is that the approach is undefined to update a knowledge base with an inconsistency. [35] state that such a kind of update “makes no sense”, which clearly does not mean that an agent or whoever is updating the knowledge base will never come across an originally inconsistent observation. Nevertheless, they do consider cases where an initial knowledge base is originally inconsistent, and they identify such case as inconsistency removal. This method consists in updating an inconsistency knowledge base with an empty update.

### 6 Zhang’s

An interesting proposal for updates through program transformation comes from [42], 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 [12] is an interesting language that is specialized in updates of agent’s policies and defined at the top of ASP. In [12], they 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 specialisation 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 [12].

#### 6.1 General View

In [42] he characterizes program-transformation 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. That is because of the introduction of two kinds of negations in logic programs—strong and default negation. Finally, once the process 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 realize these three goals, [41] performs program update transformation by means of a called prioritized logic program. In an intuitive way, this kind of program consists in preferring the latest update to the original knowledge base including non-contradictory but conflicting rules. His approach definitions and plenty of examples are easily accessible in the literature, from which we also recommend [3].

A mandatory test is the example in Observation 2, which produces counterintuitive results in many of the existing semantics for updates. This is not the case for Zhang’s semantics, whose expected answer set is just what intuition would tell us: \( \{ \text{day, \sim stars}\} \).

Despite this approach is well behaved, though, one of the counter-intuitive examples to [42]’s approach has to do with solving conflicts between rules, where most of the current semantics differ, as first pointed out by [17]:

\[^{10}\text{See [35] for further details.}\]
Suppose an initial knowledge base $K_0 = \{p \leftarrow \neg q\}$ being updated by $K_1 = \{q \leftarrow \neg p\}$. Its simple-fact update specification corresponds to

$$U_{PLP}(S_{K_0}, K_1) = (K^*, N, <),$$

where

Initial Knowledge:

$$i_0 : p$$

Inertial Rules:

$$i_1 : \text{newp} \leftarrow p, \text{not } \neg \text{newp}$$
$$i_2 : \neg \text{newp} \leftarrow \neg p, \text{not } \text{newp}$$
$$i_3 : \text{newq} \leftarrow q, \text{not } \neg \text{newq}$$
$$i_4 : \neg \text{newq} \leftarrow \neg q, \text{not } \text{newq}$$

Update Rule

$$u_1 : \text{newq} \leftarrow \text{not } \text{newp}$$

Preferences

$$N(i_1) < N(u_1), N(i_2) < N(u_1), N(i_3) < N(u_1), N(i_4) < N(u_1)$$

As a result, its unique answer set $S_{PLP}(U_{PLP}(S_{K_0}, K_1)) = \{p\}$ and none of its $<$-relations are used. Next, the minimal subset of $K_0$ that is coherent with its answer set is just $K_{(K_0,K_1)} = K_0$. Then, the update specification of $K_0$ and $K_1$ is $U_{PLP}(K_0, K_1) = (K_1 \cup K_{(K_0,K_1)} \setminus N, <)$, where

$$(t : p \leftarrow \neg q), (u : q \leftarrow \neg p), (u < t)$$

Finally, its unique reduct $q \leftarrow \neg p$ comes from

$$(K_1 \cup K_{(K_0,K_1)}) \setminus \{t\}$$

defeating rule $t$. Therefore, the conclusion of such an update is just $\{q\}$, what we would not expect.

Last, besides not satisfying some of the principles already pointed out, one of the major drawbacks of this approach is being limited to only one update to a knowledge base. Namely, it is undefined for update sequences and for successive updates, which does not seem to lead to immediate practical use. Although [42] also suggests an extension to one of his earliest approaches in [40] to deal with multiple updates, his proposal still makes the same strong assumptions when deciding between multiple models, as in Observation 6.

### 8 Acknowledgments

The author is very grateful to J. Dix and W. Jamroga for their many useful comments and proofreading some preliminary versions of this work. Special thanks are due to E. Martínez too, for proofreading this paper.

### References


