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[1] 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, $\mathcal{L}_{ASP}$). In the following $\mathcal{L}_{\mathsf{ASP}}$ is a language of propositional logic with sym-

This project is mainly supported by The National Council for Science and Technology, CONACYT, 280712, Consolidación de Laboratorios Nacionales 2017. We are also grateful to The Autonomous University of Juarez

- *Acosta-Guadarrama is with the Department of Electrical and Computer Engineering, Engineering Institute of Technology, Juarez, Mexico, 32310. E-mail: see http://www.jguadarrama.link*

Manuscript received October 10, 2016 at ENC 2016; revised October 24, 2016.

1. For historical reasons, in this paper we call it update, although it's actually belief revision. A study on the difference was first introduced by [24].

bols: $a_0, a_1, \ldots$; connectives: "," (conjunction) and meta-connective ";"; disjunction $\vee$, also denoted as $|$; $\leftarrow$ (and its counterpart, $\rightarrow$); $\neg$ (default negation or weak negation, also denoted with the word not); "$\sim$" (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* ρ 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 component 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 "$\neg$", 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

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m, \neg\ell_{m+1}, \ldots, \neg\ell_n \quad (1)$$

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

Naturally, an *extended logic program* (or ELP hereafter) is a finite set of rules of form (1) with $l = 1$; while an *integrity constraint* (also known in the literature as *strong constraint*) is a rule of form (1) with $l = 0$; while a *fact* is a rule of the same form with $l = m = n$. In particular, given a set of literals $\mathcal{A}$, for a literal $\ell \in \mathcal{A}$, the *complementary literal* is $\sim\ell$ and vice versa; for a set $\mathcal{M}$ of literals, $\sim\mathcal{M} = \{\sim\ell \mid \ell \in \mathcal{M}\}$, and $Lit_{\mathcal{M}}$ denotes the set $\mathcal{M} \cup \sim\mathcal{M}$; finally, a *signature* $\mathfrak{L}_{\mathcal{K}}$ is a finite set of literals occurring in a knowledge base, $\mathcal{K}$. Additionally, given a set of literals $\mathcal{M} \subseteq \mathcal{A}$, the complement set $\overline{\mathcal{M}} = \mathcal{A} \setminus \mathcal{M}$.

The well-known *semantics of an EDLP* consists of reducing general rules to rules without default negation "$\neg$" because the latter can be interpreted in *classical logic* by means of the well-known *Herbrand models*. In particular, the reduced rules with no default negation Mon of a rule of the form (1) is

$$\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m \quad (2)$$

where ℓ_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 $\mathcal{K}$, consisting of clauses of form (1). For any set $\mathcal{S} \subseteq \mathfrak{L}_{\mathcal{K}}$, the *answer-sets reduct* $\mathcal{K}^{\mathcal{S}}$ corresponds to

$$\mathcal{K}^{\mathcal{S}} = \{\ell_1 \vee \ell_2 \vee \ldots \vee \ell_l \leftarrow \ell_{l+1}, \ldots, \ell_m : \{\ell_{m+1}, \ldots, \ell_n\} \cap \mathcal{S} = \emptyset\} \quad (3)$$

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 corresponds to its *minimal classical model* as follows.

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

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

$$\perp \leftarrow q_1, \ldots, q_m \quad (4)$$

$\{q_1, \ldots, q_m\} \not\subseteq \mathcal{S}$.

2) If $\mathcal{S}$ contains a pair of complementary literals, then $\mathcal{S} = \mathfrak{L}_{\mathcal{K}}$.

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 $\mathcal{K}$ is a *minimal closure* of its reduct as following stated.

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

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 $\mathcal{K}$ 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 arbitrary *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 expressions of larger ground (propositional) ones without variables, where each *ground program* $\mathcal{K}$ is a set of its *ground rules* $\rho \in \mathcal{K}$. In addition, a ground rule is the set obtained by all possible substitutions of variables in ρ by constants occurring in $\mathcal{K}$ [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 Generalized Answer Sets.

This is the basic background to understand the following 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 complete 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 $(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$, with $2 \leq n$, over a set of atoms $\mathcal{A}$, assume $\mathcal{A}_\alpha$ as an extension of $\mathcal{A}$ by new pair-wise unique atoms $\mathsf{rej}(\rho)$ and α_i, for each rule ρ occurring in $\mathcal{K}$; each atom $\alpha \in \mathcal{A}$, and $1 \leq i \leq n$. An injective naming function $\mathsf{Name}(\cdot, \cdot)$ is also assumed, which assigns to each rule ρ in a program $\mathcal{K}_i$ a unique name, $\mathsf{Name}(\rho, \mathcal{K}_i)$, provided that $\mathsf{Name}(\rho, \mathcal{K}_i) \neq \mathsf{Name}(\rho', \mathcal{K}_j)$ whenever $i \neq j$. Finally, for a literal ℓ, they use ℓ_i to denote the result of replacing an atomic formula α of ℓ by α_i.

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

Then [17] define the intended answer sets of an update sequence $(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$ in terms of the answer sets of $\mathcal{K}_\triangleleft = (\mathcal{K}_1 \triangleleft \cdots \triangleleft \mathcal{K}_n)$. 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,

$$(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$$

over a set of atoms $\mathcal{A}$, then $\mathcal{S} \subseteq Lit_\mathcal{A}$ is an *update answer set* of $(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$ if and only if $\mathcal{S} = \mathcal{S}' \cap \mathcal{A}$ for some answer set $\mathcal{S}'$ of $\mathcal{K}_\triangleleft = (\mathcal{K}_1 \triangleleft \cdots \triangleleft \mathcal{K}_n)$. The collection of all the update answer sets of $(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$ is denoted by $\mathcal{U}(\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_n)$.

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-illustrative two sets of *system requirements* (translated into ASP), $\langle \mathcal{K}_1, \mathcal{K}_2 \rangle$, representing the *initial and current knowledge* of an intelligent greenhouse, which acts autonomously under specific circumstances, where

$$\begin{aligned} \mathcal{K}_1 = \{ & \mathit{notify} \leftarrow \mathit{night}, \mathrm{not}\ \mathit{wSystem} \\ & \mathit{night} \\ & \mathit{wPlants} \leftarrow \mathit{wSystem} \\ & \mathit{wSystem} \} \\ \mathcal{K}_2 = \{ & {\sim}\mathit{wSystem} \leftarrow \mathit{blackout} \\ & \mathit{blackout} \} \end{aligned}$$

Program $\mathcal{K}_1$ 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

$$\{\mathit{night}, \mathit{wPlants}, \mathit{wSystem}\}.$$

Now suppose that the systems engineer needs to incorporate a new rule that states to not water plants when the ground is flood: ${\sim}\mathit{wPlants} \leftarrow \mathit{gFlood}$. By their definition, the transformed update program $\mathcal{K}_\triangleleft = (\mathcal{K}_1 \triangleleft \cdots \triangleleft \mathcal{K}_n)$ consists of the following rules—amongst the rest of the rules that we skip for page constraints:

$$\begin{aligned} \cdots\ \mathit{wSystem}_1 \leftarrow \mathrm{not}\ \mathsf{rej}(\rho_4). \\ {\sim}\mathit{wSystem}_2 \leftarrow \mathit{blackout}, \mathrm{not}\ \mathsf{rej}(\rho_5) \cdots \\ \cdots\ \mathsf{rej}(\rho_4) \leftarrow \mathrm{not}\ \mathit{wSystem}_2 \cdots \\ \mathit{notify}_1 \leftarrow \mathit{notify}_2. \quad \mathit{notify} \leftarrow \mathit{notify}_1 \cdots \\ \mathit{wSystem}_1 \leftarrow \mathit{wSystem}_2. \quad \mathit{wSystem} \leftarrow \mathit{wSystem}_1 \cdots \\ {\sim}\mathit{wSystem}_2 \leftarrow {\sim}\mathit{wSystem}_3. \quad {\sim}\mathit{wSystem} \leftarrow {\sim}\mathit{wSystem}_2 \\ \mathit{blackout}_2 \leftarrow \mathit{blackout}_3. \quad \mathit{blackout} \leftarrow \mathit{blackout}_2 \end{aligned}$$

whose unique answer set is $\{\mathit{notify}_1, \mathit{night}, \mathit{night}_1, \mathsf{rej}(\rho_4), {\sim}\mathit{wSystem}_2, \mathit{blackout}, \mathit{blackout}_2, \mathit{notify}, {\sim}\mathit{wSystem}\}$ and its **update answer** set is just: $\{\mathit{night}, \mathit{blackout}, \mathit{notify}, {\sim}\mathit{wSystem}\}$.

Let us complete Example 1, which is one of the major disadvantages the framework exhibits [35]:

Observation 1 (Continued Example 1). Now let us consider Example 1 again and perform a *second update* to the sequence with program $\mathcal{K}_3 = \{{\sim}\mathit{blackout}\}$. Accordingly, the new answer set of the resulting update program is $\{\mathit{wSystem}_1, \mathit{wSystem}, \mathit{night}_1, \mathit{night}, \mathit{wPlants}_1, \mathit{wPlants}, \mathsf{rej}(\rho_6), {\sim}\mathit{blackout}_3, {\sim}\mathit{blackout}\}$.

As a result, by Definition 5 the corresponding **update answer sets** *are* $\mathcal{U}(\mathcal{K}_1, \mathcal{K}_2) = \{\mathit{night}, \mathit{blackout}, \mathit{notify}, {\sim}\mathit{wSystem}\}$ and $\mathcal{U}(\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3) = \{\mathit{wSystem}, \mathit{night}, \mathit{wPlants}, {\sim}\mathit{blackout}\}$. However, this result does not coincide with common intuition, just because one of the possible models contradicts the latest fact (${\sim}\mathit{blackout}$) 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.

2. Available at http://www.kr.tuwien.ac.at/staff/giuliana/project.html#Download.

3. The graphical online front end to [18] command-line solver can be found at: http://logic-lab.sourceforge.net/upd.html.

4. We have installed an online version, which also provides a graphic-oriented interface on the server itself—http://logic-lab.sourceforge.net/upd.html. Obviously, no download or installation is necessary to run the latter version.

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 $\mathcal{K}_1$ as follows:

$$\mathcal{K}_1 = \{(day \leftarrow \text{not } night), \qquad (night \leftarrow \text{not } day)$$
$$(stars \leftarrow night, \text{not } cloudy), \qquad {\sim}stars\}$$

whose unique answer set is $\{day, {\sim}stars\}$. Later, the agent acquires new information stating that *stars* and *constls*[6] are the same thing, as coded in $\mathcal{K}_2$. As soon as the agent updates $\mathcal{K}_1$ with program

$$\mathcal{K}_2 = \{(stars \leftarrow constls), \qquad (constls \leftarrow stars)\}$$

the *augmented alphabet* of the two programs contains only one new *extra atom* with respect to $\mathcal{K}_1$: $constls$. As the model of $\mathcal{K}_2$ is obviously the empty answer set, $constls$ is considered *synonym* of $stars$ by means of $\mathcal{K}_2$, 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]: $\{stars, constls, night\}$, 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 $\mathcal{K}_1$: 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]

$$\text{not } p \leftarrow \text{not } p \tag{5}$$

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 ℓ 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 $\mathsf{Rej}(\cdot,\cdot)$ is the set of rules that conflict ($\bowtie$) with both current and previous ones in the sequence. Moreover, $\mathsf{Def}(\cdot,\cdot)$ consists of the positive "default-negated" atoms that do not appear in the intended model.

Formally, two rules, ρ_1, ρ_2, are in conflict, denoted as $\rho_1 \bowtie \rho_2$, if and only if $H(\rho_1) = \text{not } B(\rho_2)$.

7. Note that the kind of rule in (5) is invalid in $\mathcal{L}_{\mathsf{ASP}}$, and it also illustrates why strong negation "${\sim}$" should not be a simple replacement to "not" in the head. Take for example, the program $\{{\sim}p \leftarrow \text{not } p\}$ whose unique **answer set** differs from the empty set: $\{{\sim}p\}$.

8. The details are easily available in the literature, like [9].

Definition 6 (Dynamic Stable Model, [8]). Let $\mathcal{P}$ be a dynamic logic program and $\mathcal{M}$ an interpretation. $\mathcal{M}$ is a *dynamic stable model* of $\mathcal{P}$ if and only if[9]

$$\begin{aligned}\mathsf{Rej}(\mathcal{P},\mathcal{M}) &= \\ \{\rho \mid \rho \in \mathcal{K}_i, \exists \rho' \in \mathcal{K}_j, i \leq j, \rho \bowtie \rho', \mathcal{M} \models B(\rho')\}; \\ \mathsf{Def}(\mathcal{P},\mathcal{M}) &= \\ \{not\_a \mid \nexists \rho \in \rho(\mathcal{P}), H(\rho) = a, \mathcal{M} \models B(\rho)\}; \\ \overline{\mathcal{M}} = \mathsf{least}(\rho(\mathcal{P}) \setminus \mathsf{Rej}(\mathcal{P},\mathcal{M}) \cup \mathsf{Def}(\mathcal{P},\mathcal{M}))\}.\end{aligned}$$

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/~jja/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: $\mathcal{P} = \mathcal{K}_1 \oplus_R \mathcal{K}_2 \oplus_R \mathcal{K}_3$, where:

$$\begin{aligned}\mathcal{K}_1 = \{(notify \leftarrow \text{not } wSystem), (wPlants \leftarrow wSystem), \\ (wSystem)\}, \\ \mathcal{K}_2 = \{(blackout), (\text{not } wSystem \leftarrow blackout)\}, \\ \mathcal{K}_3 = \{\text{not } blackout\}.\end{aligned}$$

$\mathcal{M} = \{wSystem, wPlants\}$ is a *refined dynamic stable model* of the update sequence, since the unique rejected rule is $blackout$ and the unique default is not_notify, where the least model is

$$\{wPlants, wSystem, not\_blackout, not\_notify\} = \overline{\mathcal{M}}.$$

In a similar example [35] argue, however, that the resulting model is *counterintuitive*, because after the first update, the refined dynamic stable model was $\{blackout, notify\}$ 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 $\mathcal{K}_0 = \{(c \leftarrow r), (r)\}$ updated with $\mathcal{K}_1 = \{\text{not } r \leftarrow \text{not } c\}$. Firstly, the initial generalized stable **model** of $\mathcal{K}_0$ 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 $\mathcal{P} = \mathcal{K}_0 \oplus_R \mathcal{K}_1$ has the **extra model** $\mathcal{M} = \{not\_c, not\_r\}$ because $\overline{\mathcal{M}} = \{not\_c, not\_r\}$; $\mathsf{Rej}(\mathcal{P},\mathcal{M}) = \{r\}$; $\mathsf{Def}(\mathcal{P},\mathcal{M}) = \{not\_c\}$; and $\mathsf{least}[(\mathcal{K}_0 \cup \mathcal{K}_1) \setminus \mathsf{Rej}(\mathcal{P},\mathcal{M}) \cup \mathsf{Def}(\mathcal{P},\mathcal{M})] = \{not\_c, not\_r\} = \overline{\mathcal{M}}$.

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 correspond 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 $\mathcal{A}$, and $\mathcal{A}_\alpha$ a set of abducibles, such that $\mathcal{A} \cap \mathcal{A}_\alpha = \emptyset$:

Definition 7 (Update Program, [35]). Given an abductive program $\langle P, \mathcal{A}_\alpha \rangle$, its *update program*, $\mathcal{UP}$, is defined as an EDLP such that $\mathcal{UP} = (P \setminus \mathcal{A}_\alpha) \cup \mathcal{UR}$.

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 $\mathcal{S}$ of $\mathcal{UP}$ is called *U-minimal* (U-MAS) if there is no answer set $\mathcal{S}'$ of $\mathcal{UP}$ such that $\mathcal{S}' \cap \mathcal{UA} \subset \mathcal{S} \cap \mathcal{UA}$.

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.

9. Notice that it seems they have missed the "_" and have typed "not a" rather than "not_a" in (6) from the original paper in [8]. Moreover, (6) should be a set of rules, rather than a set of literals, to be sound with the equality in (6)! In the rest of the section, it is assumed the former, as the authors do.

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: $\{tvon, watchtv, \sim blackout\}$, 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 $\mathcal{K} = \emptyset$ updated by a simple fact $\mathcal{K}_1 = \{x\}$. Following [35]'s framework, the answer sets of its update program is empty: $\mathcal{UP} = (\emptyset \setminus \{x\}) \cup \mathcal{UR}$, where $\mathcal{UR}$ is clearly empty because the extended abductive program from the update pair has no abducibles: $\mathcal{A}_\alpha = \emptyset \setminus \{x\}$.

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*.

10. See [35] for further details.

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: $\{day, \sim stars\}$.

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]:

Observation 6. Suppose an initial knowledge base $\mathcal{K}_0 = \{p \leftarrow \text{not}\, q\}$ being updated by $\mathcal{K}_1 = \{q \leftarrow \text{not}\, p\}$. Its simple-fact update specification corresponds to

$$\mathsf{U}_{\mathsf{PLP}}(\mathcal{S}_{\mathcal{K}_0}, \mathcal{K}_1) = (\mathcal{K}^*, \mathcal{N}, <),$$

where

Initial Knowledge:

$$i_0 : p$$

Inertial Rules:

$$\begin{aligned} i_1 :& \quad newp \leftarrow p, \text{not} \sim newp \\ i_2 :& \quad \sim newp \leftarrow \sim p, \text{not}\, newp \\ i_3 :& \quad newq \leftarrow q, \text{not} \sim newq \\ i_4 :& \quad \sim newq \leftarrow \sim q, \text{not}\, newq \end{aligned}$$

Update Rule

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

Preferences

$$\begin{array}{ll} N(i_1) < N(u_1) & N(i_2) < N(u_1) \\ N(i_3) < N(u_1) & N(i_4) < N(u_1) \end{array}$$

As a result, its unique answer set $\mathsf{S}_{\mathsf{PLP}}(\mathsf{U}_{\mathsf{PLP}}(\mathcal{S}_{\mathcal{K}_0}, \mathcal{K}_1)) = \{p\}$ and none of its $<$-relations are used. Next, the minimal subset of $\mathcal{K}_0$ that is coherent with its answer set is just $\mathcal{K}_{(\mathcal{K}_0, \mathcal{K}_1)} = \mathcal{K}_0$. Then, the update specification of $\mathcal{K}_0$ and $\mathcal{K}_1$ is $\mathsf{U}_{\mathsf{PLP}}(\mathcal{K}_0, \mathcal{K}_1) = (\mathcal{K}_1 \cup \mathcal{K}_{(\mathcal{K}_0, \mathcal{K}_1)}, \mathcal{N}, <)$, where

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

Finally, its unique reduct $q \leftarrow \text{not}\, p$ comes from

$$(\mathcal{K}_1 \cup \mathcal{K}_{(\mathcal{K}_0, \mathcal{K}_1)}) \setminus \{t\}$$

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.

7 Conclusions

This work is a critique of current and past proposals for update semantics, as an attempt to classify them and to bring out new challenges. This Part I starts with a set of early proposals. All of them update and revise knowledge by means of program transformations in ASP or similar semantics. They run from a wide range from *simple-fact one-step updates* of logic programs, up to unlimited updates either in a sequence or by an iterative process. Some of these works present a vast collection of postulates and principles for correct knowledge-base changes, and/or implementation. However, nearly all the proposals here introduced still present drawbacks either for being limited to only one update, or for relying on syntactical principles to change the original logic program that leads to *counterintuitive results*. Nearly all of the issues here pointed out have been surpassed by current approaches, like [5], where they have found nice properties and applications— [1], [4]. On the other hand, for practical and commercial reasons, the proposals here presented are often called semantics for *updates*, although there are implicit operations in some of them that may suggest a different classification, say *semantics for belief revisions*.

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

[1] J. C. Acosta-Guadarrama, Rogelio Dávila Pérez, Mauricio Osorio, and Victor Saldivar. *Modeling Natural Language Metaphors with an Answer Set Programming Framework*, volume LNAI 8856 of *ISSN 0302-9743 ISBN 978-3-319-13646-2*, chapter IV, pages 28–36. Springer-Verlag, 2014.

[2] Juan C. Acosta-Guadarrama. Maintaining knowledge bases at the object level. In Alexander Gelbukh and Angel F. Kuri Morales, editors, *Special Session of the 6th International MICAI Conference*, pages 3–13, Aguascalientes, Mexico, November 2007. IEEE Computer Society. ISBN: 978-0-7695-3124-3.

[3] Juan C. Acosta-Guadarrama. A road map of updating in ASP. ISSN: 1860-8477 IfI-07-16, TU-Clausthal, Clausthal, Germany, December 2007. 35pp.

[4] Juan C. Acosta-Guadarrama. Towards a logic-programming system to debug ASP knowledge bases. In Mauricio Osorio, Claudia Zepeda, Iván Olmos, José L. Carballido, José Arrazola, and Carolina Medina, editors, *7th Latin American Workshop on Non-Monotonic Reasoning 2011*, volume Vol-804, pages 3–12, Germany, November 2011. CEUR Workshop Proceedings.

[5] Juan C. Acosta-Guadarrama. Towards a unified framework for declarative knowledge-change—principles and consistency. In Mauricio Osorio, Claudia Zepeda, Ivan Olmos, José Luis Carballido, and Carolina Medina, editors, *8th Latin American Workshop on Non-Monotonic Reasoning 2012*, pages 51–62. Dirección de Fomento Editorial, Benemérita Universidad Autónoma de Puebla, 2012.

[6] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. On the logic of theory change: Partial meet contraction and revision functions. *The Journal of Symbolic Logic*, 50(2):510–530, June 1985.

[7] José Júlio Alferes, Federico Banti, Antonio Brogi, and Pascal Hitzler. The well supported semantics for multidimensional dynamic logic programs. In Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors, *Logic Programming and Nonmonotonic Reasoning*, volume 3662/2005 of *LNCS*, pages 356–368, Diamante, Italy, September 2005. Springer Berlin/Heidelberg.

[8] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.

[9] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *Journal of Logic Programming*, 45(1–3):43–70, 1999.

[10] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Teodor C. Przymusinski, and Halina Przymusinska. Dynamic logic programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 98–111, San Francisco, June 2–5 1998. Morgan Kaufmann Publishers.

[11] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS —A language for updating logic programs. *Artificial Intelligence*, 138(1–2):87–116, June 2002.

[12] Vino Fernando Crescini and Yan Zhang. Policy updater: A system for dynamic access control. *International Journal of Information Security*, 5(3):145–165, 2005.

[13] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, September 2001.

[14] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On updates of logic programs: Semantics and properties. Technical Report INFSYS RR-1843-00-08, TU Wien, Institute für Informationssysteme, 2000.

[15] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Considerations on updates of logic programs. In Manuel Ojeda-Aciego, Inman P. de Guzmán, Gerhard Brewka, and L. Moniz Pereira, editors, *Logics in Artificial Intelligence, European Workshop, JELIA 2000*, pages 2–20, Malaga, Spain, 2000. Springer Verlag.

[16] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. A framework for declarative update specifications in logic programs. In Bernhard Nebel, editor, *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI*, volume I, pages 649–654, Seattle, Washington, 2001. Morgan Kaufmann.

[17] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6):711–767, 2002.

[18] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. Reasoning about evolving nonmonotonic knowledge bases. *ACM Transactions on Computational Logic*, 6(2):389–440, 2005.

[19] Ronald Fagin. *Reasoning about Knowledge*. The MIT Press, Cambridge, Massachusetts, USA, 1995.

[20] Ronald Fagin, Gabriel M. Kuper, Jeffrey D. Ullman, and Moshe Y. Vardi. Updating logical databases. *Advances in Computing Research*, 3:1–18, 1986.

[21] Ronald Fagin, Jeffrey D. Ullman, and Moshe Y. Vardi. On the semantics of updates in databases. In *PODS '83: Proceedings of the 2nd ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 352–365, New York, NY, USA, 1983. ACM Press.

[22] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming, Proceedings of the Fifth International Conference and Symposium ICLP/SLP*, pages 1070–1080, Seattle, Washington, 1988. MIT Press.

[23] Hirofumi Katsuno and Alberto O. Mendelzon. A unified view of propositional knowledge base updates. In N. S. Sridharan, editor, *the 11th International Joint Conference on Artificial Intelligence, IJCAI-89*, pages 1413–1419, Detroit, Michigan, USA, 1989. Morgan Kaufmann.

[24] Hirofumi Katsuno and Alberto O. Mendelzon. On the difference between updating a knowledge base and revising it. In *KR'91*, pages 387–394, Cambridge, Massachusetts, USA, 1991. Morgan Kaufmann Publishers.

[25] Hirofumi Katsuno and Alberto O. Mendelzon. Propositional knowledge base revision and minimal change. *Artificial Intelligence*, 52(3):263–294, 1991.

[26] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic Reasoning, Preferential Models and Cumulative Logics. *Artificial Intelligence*, 44(1):167–207, 1990.

[27] Daniel Lehmann. Plausibility Logic. In E. Börger, G. Jäger, H. Kleine-Büning, and M. M. Richter, editors, *Computer Science Logic, 5th Workshop, CSL 91, Berne, Switzerland*, LNCS 626, pages 227–241, Berlin, 1992. Springer.

[28] Daniel Lehmann and Menachem Magidor. What does a conditional knowledge base entail? *Artificial Intelligence*, 55:1–60, 1992.

[29] David Makinson. General theory of cumulative inference. In *Non-Monotonic Reasoning, 2nd International Workshop*, pages 1–18, Grassau, Federal Republic of Germany, 1988. Springer.

[30] David Makinson. General Patterns in Nonmonotonic Reasoning. In D. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming, Nonmonotonic and Uncertain Reasoning*, volume 3, chapter 3, pages 35–110. Oxford University Press, Oxford, UK, 1994.

[31] Mauricio Osorio, Juan Antonio Navarro, and José Arrazola. Equivalence in answer set programming. In Alberto Pettorossi, editor, *Logic Based Program Synthesis and Transformation: 11^{th} International Workshop; selected papers / LOPSTR 2001, Paphos, Cyprus*, LNCS 2372, pages 57–75, Paphos, Cyprus, November 2001. Springer-Verlag Berlin.

[32] Mauricio Osorio and Fernando Zacarías. Irrelevance of syntax in updating answer set programs. In *Proceedings of the Fourth Mexican International Conference on Computer Science (ENC' 03) In Workshop on Logic and Agents*, Apizaco, Mexico, 2003. IEEE Computer Society.

[33] Mauricio Osorio and Fernando Zacarías. On updates of logic programs: A properties-based approach. In *FoIKS*, pages 231–241, Wilhelminenburg Castle, Austria, 2004. Springer.

[34] Chiaki Sakama and Katsumi Inoue. Updating extended logic programs through abduction. In Michael Gelfond, Nicole Leone, and Gerald Pfeifer, editors, *LPNMR '99: Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1730 of *LNCS*, pages 147–161, El Paso, Texas, USA, 1999. Springer-Verlag.

[35] Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming*, 3(6):671–715, 2003.

[36] Marianne Winslett. *Updating logical databases*. Cambridge University Press, New York, NY, USA, 1990.

[37] Fernando Zacarías, Mauricio Osorio, Juan C. Acosta-Guadarrama, and Jürgen Dix. Updates in Answer Set Programming Based on Structural Properties. In Sheila McIlraith, Pavlos Peppas, and Michael Thielscher, editors, *7th International Symposium on Logical Formalizations of Commonsense Reasoning*, pages 213–219, Corfu, Greece, May 2005. Fakultät Informatik, TU-Dresden, ISSN 1430-211X.

[38] Yan Zhang. On propositional knowledge base updates. *Australian Journal of Intelligent Information Processing Systems*, 2:20–29, 1995.

[39] Yan Zhang. The complexity of logic program updates. In *AI '01: Proceedings of the 14th Australian Joint Conference on Artificial Intelligence*, pages 631–642, London, UK, 2001. Springer-Verlag.

[40] Yan Zhang. Minimal change and maximal coherence for epistemic logic program updates. In *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 112–120, Acapulco, Mexico, 2003. Morgan Kaufmann.

[41] Yan Zhang. Two results for prioritized logic programming. *Theory and Practice of Logic Programming*, 3(2):223–242, March 2003.

[42] Yan Zhang. Logic program-based updates. *ACM Transactions on Computational Logic*, 7(3):421–472, July 2006.