Robust Equivalence Models for Semantic Updates of Answer-Set Programs

Martin Slota and **João Leite**
CENTRIA & Departamento de Informática
Universidade Nova de Lisboa
2829-516 Caparica, Portugal

Abstract

Existing methods for dealing with knowledge updates differ greatly depending on the underlying knowledge representation formalism. When Classical Logic is used, update operators are typically based on manipulating the knowledge base on the model-theoretic level. On the opposite side of the spectrum stand the semantics for updating Answer-Set Programs where most approaches need to rely on rule syntax. Yet, a unifying perspective that could embrace all these approaches is of great importance as it enables a deeper understanding of all involved methods and principles and creates room for their cross-fertilisation, ripening and further development.

This paper bridges these seemingly irreconcilable approaches to updates. It introduces a novel monotonic characterisation of rules, dubbed *RE-models*, and shows it to be a more suitable semantic foundation for rule updates than SE-models. A generic framework for defining semantic rule update operators is then proposed. It is based on the idea of viewing a program as the *set of sets of RE-models* of its rules; updates are performed by introducing additional interpretations to the sets of RE-models of rules in the original program. It is shown that particular instances of the framework are closely related to both belief update principles and traditional approaches to rule updates and enjoy a range of plausible syntactic as well as semantic properties.

1 Introduction

In this paper we propose a novel generic method for specifying rule update operators. By viewing a logic program as the *set of sets of models* of its rules, and seeing updates as exceptions to those sets, we are able to define concrete operators which simultaneously obey many (syntactic) properties usually discussed in the rule update literature, and many (semantic) properties discussed in the belief change area of research, until now considered irreconcilable.

One of the main challenges for knowledge engineering and information management is to efficiently and plausibly deal with the incorporation of new, possibly conflicting knowledge and beliefs. In their seminal work, Alchourrón, Gärdenfors, and Makinson (1985) addressed the issues related to this process in the context of Classical Logic, resulting in the ample research area of *belief change*.

Update is a belief change operation that consists of bringing a knowledge base up to date when the world it describes changes (Winslett 1990; Katsuno and Mendelzon 1991). The study of update operators commenced with the introduction of *KM postulates for update* (Katsuno and Mendelzon 1991). Though the desirability of half of the eight KM postulates has later been questioned by many (Brewka and Hertzberg 1993; Boutilier 1995; Doherty, Lukaszewicz, and Madalinska-Bugaj 1998; Herzig and Rifi 1999), the fourth postulate, that we refer to as *syntax-independence*, is generally considered very desirable as it guarantees that insignificant syntactic differences in the representation of knowledge do not affect the result of an update (Herzig and Rifi 1999).

Updates were later studied in the context of Answer-Set Programs. Earlier methods were based on literal inertia (Marek and Truszczynski 1998) but proved not sufficiently expressive. Though the state-of-the-art approaches are guided by the same basic intuitions and aspirations as belief update, they build upon fundamentally different principles and methods. While many are based on the *causal rejection principle* (Leite and Pereira 1997; Alferes et al. 2000; Eiter et al. 2002; Alferes et al. 2005; Osorio and Cuevas 2007), others employ syntactic transformations and other methods, such as abduction (Sakama and Inoue 2003), forgetting (Zhang and Foo 2005), prioritisation (Zhang 2006), preferences (Delgrande, Schaub, and Tompits 2007), or dependencies on defeasible assumptions (Šefránek 2011; Krümpelmann 2012).

Despite the variety of techniques used in these approaches, certain properties are common to all of them. First, the stable models assigned to a program after one or more updates are always *supported*: for each true atom p there exists a rule in either the original program or its updates that has p in the head and whose body is satisfied. Second, all mentioned rule update semantics coincide when it comes to *updating sets of facts* by newer facts. We conjecture that any reasonable rule update semantics should indeed be in line with the basic intuitions regarding *support* and *fact update*.

But in difference to belief update, rule updates exercise rule inertia instead of literal inertia. Rather than operating on the models of a logic program, they refer to its syntactic structure: the individual rules and, in many cases, also the literals in heads and bodies of these rules. These properties render them seemingly irreconcilable with belief update

where *literal inertia* and *syntax-independence* are central.

Yet, a unifying framework that could embrace both belief and rule updates is of great importance as it enables a deeper understanding of all involved methods and principles and creates room for their cross-fertilisation, ripening and further development. It is also important for the development of update semantics for *Hybrid Knowledge Bases* – recently introduced tight semantic and computational frameworks for the combination of decidable fragments of first-order logic, such as Description Logics, with Answer-Set Programs (Motik and Rosati 2010; Knorr, Alferes, and Hitzler: 2011; de Bruijn et al. 2010; 2011).

Moreover, we argue that syntax-independence, central to belief updates, is essential and should be pursued at large in order to encourage a logical underpinning of all update operators and so facilitate analysis of their semantic properties. When equivalence with respect to classical models is inappropriate, as is the case with rules, syntax-independence should be retained by finding an appropriate notion of equivalence, specific to the underlying formalism and its use.

With these standpoints in mind, we proceed with our previous work addressing the logical foundations of rule updates. Though it has previously been shown that *strong equivalence* and an associated monotonic characterisation of rules, *SE-models*, can be used to define syntax-independent rule revision (Delgrande et al. 2008) and rule update (Slota and Leite 2010) operators, it also turned out that these approaches are severely limited because the resulting operators cannot respect both support and fact update (Slota and Leite 2010). This can be demonstrated on programs $P = \{ p., q. \}$ and $Q = \{ p., q \leftarrow p. \}$ which are strongly equivalent, so, due to syntax independence, an update asserting that p is now false ought to lead to the same stable models in both cases. Because of fact update, such an update on P must lead to a stable model where q is true. But in case of Q such a stable model would be unsupported.

This led us to the study of stronger notions of program equivalence. In (Slota and Leite 2011) we proposed to view a program as the *set of sets of models of its rules* in order to acknowledge rules as the atomic pieces of knowledge and, at the same time, abstract away from unimportant differences between their syntactic forms, focusing on their semantic content. In this paper we develop these ideas further and

- introduce a novel monotonic characterisation of rules, *RE-models*, and show that they form a more suitable semantic foundation for rule updates than SE-models;
- propose a generic framework for defining semantic rule update operators: a program, viewed as the *set of sets of RE-models* of its rules, is updated by introducing additional interpretations to those sets of RE-models;
- identify instances of the framework that bridge belief update with rule update semantics: they combine *syntax-independence* with *support* and *fact update* and have other desirable *syntactic* as well as *semantic* properties.

This paper is structured as follows: We introduce the necessary theoretical background in Sect. 2 and in Sect. 3 we define RE-models and associated notions of equivalence. Section 4 introduces the framework for semantic rule updates, defines some of its instances instances, and analyses their theoretical properties. We point at possible future directions in Sect. 5.

2 Background

Propositional Logic. We consider a propositional language over a finite set of propositional variables $\mathcal{L}$ and the usual set of propositional connectives to form propositional formulae. A (two-valued) interpretation is any $I \subseteq \mathcal{L}$. Each atom p is assigned one of two truth values in I: $I(p) = \mathsf{T}$ if $p \in I$ and $I(p) = \mathsf{F}$ otherwise. This assignment is generalised in the standard way to all propositional formulae. The set of all models of a formula ϕ is denoted by $[\![\phi]\!]$. We say ϕ is *complete* if $[\![\phi]\!]$ is a singleton set. For two formulae ϕ, ψ we say that ϕ *entails* ψ, denoted by $\phi \models \psi$, if $[\![\phi]\!] \subseteq [\![\psi]\!]$, and that ϕ *is equivalent to* ψ, denoted by $\phi \equiv \psi$, if $[\![\phi]\!] = [\![\psi]\!]$.

Logic Programs. The basic syntactic building blocks of rules are also propositional atoms from $\mathcal{L}$. A *negative literal* is an atom preceded by $\sim$ denoting default negation. A *literal* is either an atom or a negative literal. As a convention, double default negation is absorbed, so that $\sim\sim p$ denotes the atom p. Given a set of literals B, we introduce the following notation: $B^+ = \{ p \in \mathcal{L} \mid p \in B \}$, $B^- = \{ p \in \mathcal{L} \mid \sim p \in B \}$, $\sim B = \{ \sim L \mid L \in B \}$.

A *rule* is a pair of sets of literals $\rho = \langle H(\rho), B(\rho) \rangle$. We say that $H(\rho)$ is the *head of* ρ and $B(\rho)$ is the *body of* ρ. Usually, for convenience, we write ρ as

$$H(\rho)^+; \sim H(\rho)^- \leftarrow B(\rho)^+, \sim B(\rho)^-.$$

A rule is called *non-disjunctive* if its head contains at most one literal; *a fact* if its head contains exactly one literal and its body is empty. A *program* is any set of rules. A program is *non-disjunctive* if all its rules are; *acyclic* if it satisfies the conditions set out in (Apt and Bezem 1991).

Throughout the rest of the paper, we need to refer to a single, fixed representative of the class of all tautological rules. We call this representative the *canonical tautology*.

Definition 1 (Canonical Tautology). *Let p_τ be a fixed atom from $\mathcal{L}$. The* canonical tautology *τ is the rule $p_\tau \leftarrow p_\tau$.*

In the following, we define stable models of a logic program (Gelfond and Lifschitz 1988; 1991) as well as two monotonic model-theoretic characterisations of rules. One is that of classical models, where a rule is simply treated as a classical implication. The other, *SE-models* (Turner 2003), is based on the logic of Here-and-There (Heyting 1930; Pearce 1997) and is expressive enough to capture both classical models and stable models.

Classical models, or *C-models*, of a rule ρ are defined as models of the propositional formula obtained from ρ by treating the arrow as material implication and default negation as classical negation. The set of all C-models of a rule ρ is denoted by $[\![\rho]\!]^{\mathsf{C}}$ and for any program P, $[\![P]\!]^{\mathsf{C}} = \bigcap_{\rho \in P} [\![\rho]\!]^{\mathsf{C}}$. A program P is *consistent* if $[\![P]\!]^{\mathsf{C}} \neq \emptyset$. For a set of literals B and interpretation J, we write $J \models B$ if $J(p) = \mathsf{T}$ for all $p \in B^+$ and $J(p) = \mathsf{F}$ for all $p \in B^-$.

The stable and SE-models are defined in terms of reducts. Given a rule ρ and an interpretation J, the *reduct of ρ w.r.t. J*, denoted by ρ^J, is $(H(\rho)^+ \leftarrow B(\rho)^+.)$ if all atoms from

$B(\rho)^-$ are false in J and all atoms from $H(\rho)^-$ are true in J; otherwise it is the canonical tautology τ. The *reduct of a program P w.r.t. J* is defined as $P^J = \{\, \rho^J \mid \rho \in P \,\}$.

An interpretation J is a *stable model* of a program P if J is a subset-minimal C-model of P^J. The set of all stable models of P is denoted by $[\![P]\!]^{\mathsf{SM}}$.

SE-models are semantic structures that can be seen as three-valued interpretations. In particular, we call a pair of interpretations $X = \langle I, J\rangle$ such that $I \subseteq J$ a *three-valued interpretation*. Each atom p is assigned one of three truth values in X: $X(p) = \mathsf{T}$ if $p \in I$; $X(p) = \mathsf{U}$ if $p \in J \setminus I$; $X(p) = \mathsf{F}$ if $p \in \mathcal{L} \setminus J$. The set of all three-valued interpretations is denoted by $\mathcal{X}$. A three-valued interpretation $\langle I, J\rangle$ is an *SE-model* of a rule ρ if J is a C-model of ρ and I is a C-model of ρ^J. The set of all SE-models of a rule ρ is denoted by $[\![\rho]\!]^{\mathsf{SE}}$ and for any program P, $[\![P]\!]^{\mathsf{SE}} = \bigcap_{\rho\in P}[\![\rho]\!]^{\mathsf{SE}}$. Note that J is a stable model of P if and only if $\langle J, J\rangle \in [\![P]\!]^{\mathsf{SE}}$ and for all $I \subsetneq J$, $\langle I, J\rangle \notin [\![P]\!]^{\mathsf{SE}}$. Also, $J \in [\![P]\!]^{\mathsf{C}}$ if and only if $\langle J, J\rangle \in [\![P]\!]^{\mathsf{SE}}$. We say that a rule ρ is **(SE-)tautological** if $[\![\rho]\!]^{\mathsf{SE}} = \mathcal{X}$. Note that the canonical tautology (c.f. Def. 1) is tautological.

Belief Update. A belief update operator is a function that assigns a formula to each pair of formulae. The following postulates were suggested by Katsuno and Mendelzon (1991) for belief update operators:

(U1) $\phi \diamond \mu \models \mu$.

(U2) If $\phi \models \mu$, then $\phi \diamond \mu \equiv \phi$.

(U3) If $[\![\phi]\!] \neq \emptyset$ and $[\![\mu]\!] \neq \emptyset$, then $[\![\phi \diamond \mu]\!] \neq \emptyset$.

(U4) If $\phi \equiv \psi$ and $\mu \equiv \nu$, then $\phi \diamond \mu \equiv \psi \diamond \nu$.

(U5) $(\phi \diamond \mu) \wedge \nu \models \phi \diamond (\mu \wedge \nu)$.

(U6) If $\phi \diamond \mu \models \nu$ and $\phi \diamond \nu \models \mu$, then $\phi \diamond \mu \equiv \phi \diamond \nu$.

(U7) $(\phi \diamond \mu) \wedge (\phi \diamond \nu) \models \phi \diamond (\mu \vee \nu)$ if ϕ is complete.

(U8) $(\phi \vee \psi) \diamond \mu \equiv (\phi \diamond \mu) \vee (\psi \diamond \mu)$.

However, update operators defined in the literature do not always satisfy all eight postulates. In fact, half of the postulates, namely (U2), (U5), (U6) and (U7), are controversial: the first three lead to undesirable behaviour while the last one is very hard to explain intuitively and is satisfied only by a minority of update operators (Herzig and Rifi 1999). Note also that (U2) entails the following three weaker principles:

(U2.$\top$) $\phi \diamond \top \equiv \phi$.

(U2.1) $\phi \wedge \mu \models \phi \diamond \mu$.

(U2.2) $(\phi \wedge \mu) \diamond \mu \models \phi$.

The first two are uncontroversial as they are satisfied by all update operators. In addition, the latter two together are powerful enough to entail (U2). Thus, the controversial part of (U2) is (U2.2) (Herzig and Rifi 1999).

Rule Update. Rule update semantics assign stable models to pairs or sequences of programs where each component represents an update of the preceding ones. In the following, we formalise some of the intuitions behind these semantics. Subsequent sections will then present and tackle the issues with finding syntax-independent rule update operators that satisfy these intuitions.

We start with the basic concepts. A *dynamic logic program* (DLP) is a finite sequence of non-disjunctive programs. Given a DLP $\mathcal{P}$, we use $\mathsf{all}(\mathcal{P})$ to denote the multiset of all rules appearing in components of $\mathcal{P}$. We say that $\mathcal{P}$ is *acyclic* if $\mathsf{all}(\mathcal{P})$ is acyclic. A rule update semantics SEM defines *SEM-stable models* for every DLP $\mathcal{P}$.

As indicated in the introduction, rule update semantics implicitly follow certain basic intuitions. Particularly, they produce *supported models* and their behaviour coincides when it comes to *updating sets of facts* by newer facts. In the following we formalise these two properties w.r.t. rule update semantics for DLPs.[1] We call them *syntactic* because their formulation requires that we refer to the syntax of the respective DLP.

In the static setting, support (Apt, Blair, and Walker 1988; Dix 1995) is one of the basic conditions that Logic Programming semantics are intuitively designed to satisfy. Its generalisation to the dynamic case is straight-forward.

Syntactic Property 1 (Support). *Let P be a program, p an atom and J an interpretation. We say that P* supports *p in J if $p \in H(\rho)$ and $J \models B(\rho)$ for some rule $\rho \in P$.*

A rule update semantics SEM respects support *if for every DLP $\mathcal{P}$ and every SEM-stable model J of $\mathcal{P}$ the following condition is satisfied: Every atom $p \in J$ is supported by $\mathsf{all}(\mathcal{P})$ in J.*

Thus, if a rule update semantics SEM respects support, then there is at least *some* justification for every atom that is true in a SEM-stable model.

The second syntactic property that is generally adhered to is the usual expectation regarding how facts are to be updated by newer facts. It enforces a limited notion of atom inertia but only for the case when both the initial program and its updates are consistent sets of facts.

Syntactic Property 2 (Fact Update). *A rule update semantics SEM* respects fact update *if for every finite sequence of consistent sets of facts $\mathcal{P} = \langle P_i\rangle_{i<n}$, the unique SEM-stable model of $\mathcal{P}$ is the interpretation*

$$\{\, p \mid \exists j : (p.) \in P_j \wedge (\forall i : j < i < n \Longrightarrow (\sim p.) \notin P_i) \,\} \quad .$$

We also introduce two further syntactic properties that are more tightly bound to approaches based on the causal rejection principle (Leite and Pereira 1997; Alferes et al. 2000; Eiter et al. 2002; Alferes et al. 2005). The first one states the principle itself, under the assumption that a *conflict* between rules occurs if and only if the rules have opposite heads.

Syntactic Property 3 (Causal Rejection). *A rule update semantics SEM* respects causal rejection *if for every DLP $\mathcal{P} = \langle P_i\rangle_{i<n}$, every SEM-stable model J of $\mathcal{P}$, all $i < n$ and all rules $\rho \in P_i$, if J is not a C-model of ρ, then there exists a rule $\sigma \in P_j$ with $j > i$ such that $H(\rho) = \sim H(\sigma)$ and $J \models B(\sigma)$.*

Intuitively, the principle requires that all *rejected* rules, i.e. rules that are not satisfied in a SEM-stable model J, must

[1] Note that although many rule update semantics disallow default negation in rule heads and consider programs with strong negation, both of these properties can be naturally adjusted to such situations. This is, however, outside the scope of this contribution.

be in conflict with a more recent rule whose body is satisfied in J. This rule then provides a *cause* for the rejection.

The final syntactic property stems from the fact that all rule update semantics based on causal rejection coincide on acyclic DLPs (Homola 2004; Alferes et al. 2005). Thus, the behaviour of any rule update semantics on acyclic DLPs can be used as a way to compare it to all these semantics simultaneously. In order to formalise this property, we reproduce here the historically first of these update semantics, the *justified update semantics* (Leite and Pereira 1997).

The set of rejected rules in $\mathcal{P}$ w.r.t. an interpretation J is

$$\mathsf{rej}(\mathcal{P}, J) = \{\, \rho \mid \exists i\, \exists j\, \exists \sigma : \rho \in P_i \wedge \sigma \in P_j \wedge i < j \wedge H(\rho) = {\sim}H(\sigma) \wedge J \models B(\sigma) \,\} \ .$$

We say that J is a *justified update model of* $\mathcal{P}$ if J is a stable model of $\mathsf{all}(\mathcal{P}) \setminus \mathsf{rej}(\mathcal{P}, J)$. We denote the set of all justified update models of $\mathcal{P}$ by $[\![\mathcal{P}]\!]^{\mathsf{JU}}$.

The final syntactic property is then stated as follows:

Syntactic Property 4 (Acyclic Justified Update). *A rule update semantics* SEM respects acyclic justified update *if for every acyclic DLP $\mathcal{P}$, the set of* SEM*-stable models is $[\![\mathcal{P}]\!]^{\mathsf{JU}}$.*

Program Equivalence. While in propositional logic equivalence under classical models is *the* equivalence, there is no such single notion of program equivalence. When considering Answer-Set Programs, the first choice is *stable equivalence* (or *SM-equivalence*) that compares programs based on their sets of stable models.

In many cases, however, SM-equivalence is not strong enough because programs with the same stable models, when augmented with the same additional rules, may end up having completely different stable models. This gives rise to the notion of *strong equivalence* (Lifschitz, Pearce, and Valverde 2001) which requires that stable models stay the same even in the presence of additional rules. It is a well-known fact that programs are strongly equivalent if and only if they have the same set of SE-models (Turner 2003). Thus, we refer to strong equivalence as *SE-equivalence*.

But even SE-equivalence is not satisfactory when used as a basis for syntax-independent rule update operators because such operators cannot respect both support and fact update. In the following we state this result formally.

By a rule update operator we understand a function that assigns a program to each pair of programs. A rule update operator $\oplus$ is extended to DLPs as follows: $\bigoplus\langle P_0\rangle = P_0$; $\bigoplus\langle P_i\rangle_{i<n+1} = (\bigoplus\langle P_i\rangle_{i<n}) \oplus P_n$. Note that such an operator naturally induces a rule update semantics $\mathsf{SEM}_{\oplus}$: given a DLP $\mathcal{P}$, the $\mathsf{SEM}_{\oplus}$-stable models of $\mathcal{P}$ are the stable models of $\bigoplus \mathcal{P}$. In the rest of this paper we exercise a slight abuse of notation by referring to the operators and their associated update semantics interchangeably. The syntax-independence of a rule update operator w.r.t. SE-equivalence, denoted by $\equiv_{\mathsf{SE}}$, can be captured by a reformulation of the belief update postulate (U4), stating that for all programs P, Q, U, V,

(PU4)$_{\mathsf{SE}}$ If $P \equiv_{\mathsf{SE}} Q$ and $U \equiv_{\mathsf{SE}} V$, then $P \oplus U \equiv_{\mathsf{SE}} Q \oplus V$.

The results of (Slota and Leite 2010) entail the following:

Theorem 2. *A rule update operator that satisfies (PU4)$_{\mathsf{SE}}$ cannot respect both support and fact update.*

Thus, in order to arrive at syntax-independent rule update operators that respect both support and fact update, we need to search for a notion of program equivalence that is stronger than SE-equivalence. One candidate is the *strong update equivalence* (or *SU-equivalence*) (Inoue and Sakama 2004), which requires that under both additions and removals of rules, stable models of the two programs in question remain the same. It has been shown in (Inoue and Sakama 2004) that this notion of equivalence is very strong – programs are SU-equivalent only if they contain exactly the same non-tautological rules and, in addition, each of them may contain some tautological ones. Thus, this notion of program equivalence seems perhaps *too strong* as it is not difficult to find rules such as ${\sim}p \leftarrow p.$ and $\leftarrow p.$ that are syntactically different but carry the same meaning.

This observation resulted in the definition of *strong rule equivalence* (or *SR-equivalence*) in (Slota and Leite 2011) that, in terms of strength, falls between SE-equivalence and SU-equivalence. It is based on the idea of viewing a program P as the set of sets of SE-models of its rules $\langle\!\langle P \rangle\!\rangle^{\mathsf{SE}} = \{\, [\![\rho]\!]^{\mathsf{SE}} \mid \rho \in P \,\}$. Formally:

Definition 3 (Program Equivalence). *Let P, Q be programs, $P^\tau = P \cup \{\, \tau \,\}$ and $Q^\tau = Q \cup \{\, \tau \,\}$. We write*

$P \equiv_{\mathsf{SM}} Q$ *whenever* $[\![P]\!]^{\mathsf{SM}} = [\![Q]\!]^{\mathsf{SM}}$;
$P \equiv_{\mathsf{SE}} Q$ *whenever* $[\![P]\!]^{\mathsf{SE}} = [\![Q]\!]^{\mathsf{SE}}$;
$P \equiv_{\mathsf{SR}} Q$ *whenever* $\langle\!\langle P^\tau \rangle\!\rangle^{\mathsf{SE}} = \langle\!\langle Q^\tau \rangle\!\rangle^{\mathsf{SE}}$;
$P \equiv_{\mathsf{SU}} Q$ *whenever* $[\![(P \setminus Q) \cup (Q \setminus P)]\!]^{\mathsf{SE}} = \mathcal{X}$.

We say that P is X-equivalent to *Q if $P \equiv_{\mathsf{X}} Q$.*

So two programs are SR-equivalent if they contain the same rules, modulo SE-models; the canonical tautology τ is added to both programs so that presence or absence of tautological rules in a program does not influence program equivalence – without it, programs such as $\emptyset$ and $\{\, \tau \,\}$ would not be considered SR-equivalent.

To formally capture the comparison of strength between these notions of program equivalence, we write $\equiv_{\mathsf{X}} \preceq \equiv_{\mathsf{Y}}$ if $P \equiv_{\mathsf{Y}} Q$ implies $P \equiv_{\mathsf{X}} Q$ and $\equiv_{\mathsf{X}} \prec \equiv_{\mathsf{Y}}$ if $\equiv_{\mathsf{X}} \preceq \equiv_{\mathsf{Y}}$ but not $\equiv_{\mathsf{Y}} \preceq \equiv_{\mathsf{X}}$. We obtain the following (Slota and Leite 2011):

Proposition 4. $\equiv_{\mathsf{SM}} \prec \equiv_{\mathsf{SE}} \prec \equiv_{\mathsf{SR}} \prec \equiv_{\mathsf{SU}}$.

3 Robust Equivalence Models

In (Slota and Leite 2011) we also studied the expressivity of SE-models with respect to a single rule. On the one hand, SE-models turned out to be a useful means of stripping away irrelevant syntactic details. On the other hand, a rule with a negative literal in its head is indistinguishable from an integrity constraint (Inoue and Sakama 1998; Janhunen 2001; Cabalar, Pearce, and Valverde 2007). For example, the rules

$$\leftarrow p, q. \qquad {\sim}p \leftarrow q. \qquad {\sim}q \leftarrow p. \tag{1}$$

have the same set of SE-models. In a static setting, these rules indeed carry essentially the same meaning: "it must not be the case that p and q are both true". But in a dynamic context, the latter two rules may, in addition, express that the truth of one atom gives a *reason* for the other atom to *become false* (Leite and Pereira 1997; Alferes et al. 2000;

2005). These classes of rules can be formally captured as follows:

Definition 5 (Constraint and Abolishing Rule). *A rule ρ is a* constraint *if $H(\rho) = \emptyset$ and $B(\rho)^+$ is disjoint with $B(\rho)^-$.*

A rule ρ is abolishing *if $H(\rho)^+ = \emptyset$, $H(\rho)^- \neq \emptyset$ and the sets $H(\rho)^-$, $B(\rho)^+$ and $B(\rho)^-$ are pairwise disjoint.*

So in the context of updates, what we need is a semantic characterisation of rules that

1) distinguishes constraints from related abolishing rules;
2) discards irrelevant syntactic details (akin to SE-models);
3) is clearly related to stable models (akin to SE-models).

In the following we introduce a novel monotonic semantics that exactly meets these criteria. We show that it possesses the desired properties and use it to introduce a notion of program equivalence that is strong enough as a basis for syntax-independent rule update operators.

Without further ado, *robust equivalence models*, or *RE-models* for short, are defined as follows:

Definition 6 (RE-Model). *A three-valued interpretation $\langle I, J\rangle$ is an* RE-model *of a rule ρ if I is a C-model of ρ^J.*

The set of all RE-models of a rule ρ is denoted by $[\![\rho]\!]^{\mathsf{RE}}$ and for any program P, $[\![P]\!]^{\mathsf{RE}} = \bigcap_{\rho\in P}[\![\rho]\!]^{\mathsf{RE}}$.

A rule ρ is RE-tautological *if $[\![\rho]\!]^{\mathsf{RE}} = \mathcal{X}$. Rules ρ, σ are* RE-equivalent *if $[\![\rho]\!]^{\mathsf{RE}} = [\![\sigma]\!]^{\mathsf{RE}}$;*

Thus, unlike with SE-models, it is not required that J be a C-model of ρ in order for $\langle I, J\rangle$ to be an RE-model of ρ. As a consequence, RE-models can distinguish between rules in (1): while both $\langle\{\,q\,\},\{\,p, q\,\}\rangle$ and $\langle\{\,p\,\},\{\,p, q\,\}\rangle$ are RE-models of the constraint, the former is not an RE-model of the first abolishing rule and the latter is not an RE-model of the second abolishing rule. This result holds in general, establishing requirement 1):

Proposition 7. *If ρ, σ are two different abolishing rules or an abolishing rule and a constraint, then ρ, σ are not RE-equivalent.*

As for requirement 2), we first note that RE-equivalence is a refinement of SE-equivalence – there are no rules that are RE-equivalent but not SE-equivalent. The following result also shows that it is *only* the ability to distinguish between constraints and abolishing rules that is introduced by RE-models. Rules that are not RE-equivalent to abolishing rules are distinguishable by RE-models if and only if they are distinguishable by SE-models. Furthermore, the class of tautological rules is the same under SE- and RE-models, so we can simply use the word *tautological* without ambiguity.

Proposition 8 (RE- vs. SE-Equivalence). *If two rules are RE-equivalent, then they are SE-equivalent.*
If two rules, neither of which is RE-equivalent to an abolishing rule, are SE-equivalent, then they are RE-equivalent.
A rule is RE-tautological if and only if it is SE-tautological.

The affinity between SE-models and stable models is fully retained by RE-models, which establishes requirement 3).

Proposition 9 (RE-Models vs. Stable Models). *An interpretation J is a stable model of a program P if and only if $\langle J, J\rangle \in [\![P]\!]^{\mathsf{RE}}$ and for all $I \subsetneq J$, $\langle I, J\rangle \notin [\![P]\!]^{\mathsf{RE}}$.*

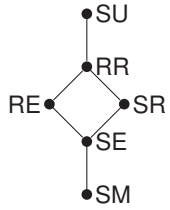

Figure 1: Program equivalences from the weakest in the bottom to the strongest on top. A missing link between X and Y indicates that $\equiv_{\mathsf{X}}$ is neither stronger nor weaker than $\equiv_{\mathsf{Y}}$.

Worth noting is also that any set of three-valued interpretations can be represented by a program using RE-models. This is not the case with SE-models since only sets of three-valued interpretations R such that $\langle I, J\rangle \in R$ implies $\langle J, J\rangle \in R$ have corresponding programs.

Proposition 10. *Let R be a set of three-valued interpretations. Then there exists a program P such that $[\![P]\!]^{\mathsf{RE}} = R$.*

Since RE-models are a refinement of SE-models that keep their essential properties while at the same time they are able to distinguish constraints from abolishing rules, we henceforth adopt them as the basis for defining syntax-independent rule update operators. We denote the set of sets of RE-models of rules inside a program P by $\langle\!\langle P\rangle\!\rangle^{\mathsf{RE}} = \{\,[\![\rho]\!]^{\mathsf{RE}} \mid \rho \in P\,\}$. We also introduce two additional notions of program equivalence: RE- and RR-equivalence that are analogous to SE- and SR-equivalence.

Definition 11 (Program Equivalence Using RE-Models). *Let P, Q be programs, $P^\tau = P \cup \{\,\tau\,\}$, $Q^\tau = Q \cup \{\,\tau\,\}$. We write*

$P \equiv_{\mathsf{RE}} Q$ *whenever* $[\![P]\!]^{\mathsf{RE}} = [\![Q]\!]^{\mathsf{RE}}$;
$P \equiv_{\mathsf{RR}} Q$ *whenever* $\langle\!\langle P^\tau\rangle\!\rangle^{\mathsf{RE}} = \langle\!\langle Q^\tau\rangle\!\rangle^{\mathsf{RE}}$.

Note that it follows directly from previous considerations that RR-equivalence is stronger than SR-equivalence and RE-equivalence is stronger than SE-equivalence. Figure 1 compares all six notions of equivalence in terms of strength (c.f. also Prop. 4). Note that RE-equivalence is neither stronger nor weaker than SR-equivalence: programs such as $\{\,p.,q.\,\}$ and $\{\,p., q \leftarrow p.\,\}$ are RE-equivalent but not SR-equivalent while programs such as $\{\,{\sim}p.\,\}$ and $\{\,\leftarrow p.\,\}$ are SR-equivalent but not RE-equivalent.

4 Exception-Driven Rule Update Operators

In this section we propose a generic framework for defining semantic rule update operators. We define instances of the framework and show that they enjoy a number of plausible properties, ranging from the respect for support and fact update to syntax-independence and other semantic properties.

As suggested above, a program is semantically characterised by the set of sets of RE-models of its rules. Our update framework is based on a simple yet novel idea of introducing additional interpretations – *exceptions* – to the sets of RE-models of rules in the original program. The formalisation of this idea is straight-forward: an exception-driven

update operator is characterised by an *exception function* ε that takes three inputs: the set of RE-models $[\![\rho]\!]^{\mathsf{RE}}$ of a rule $\rho \in P$ and the semantic characterisations $\langle\!\langle P \rangle\!\rangle^{\mathsf{RE}}$, $\langle\!\langle U \rangle\!\rangle^{\mathsf{RE}}$ of the original and updating programs. It then returns the three-valued interpretations that are to be introduced as exceptions to ρ, so the characterisation of the updated program contains the augmented set of RE-models,

$$[\![\rho]\!]^{\mathsf{RE}} \cup \varepsilon\left([\![\rho]\!]^{\mathsf{RE}}, \langle\!\langle P \rangle\!\rangle^{\mathsf{RE}}, \langle\!\langle U \rangle\!\rangle^{\mathsf{RE}}\right) \quad . \tag{2}$$

Hence, the semantic characterisation of P updated by U is

$$\left\{\, [\![\rho]\!]^{\mathsf{RE}} \cup \varepsilon\left([\![\rho]\!]^{\mathsf{RE}}, \langle\!\langle P \rangle\!\rangle^{\mathsf{RE}}, \langle\!\langle U \rangle\!\rangle^{\mathsf{RE}}\right) \;\middle|\; \rho \in P \,\right\} \cup \langle\!\langle U \rangle\!\rangle^{\mathsf{RE}} \; . \tag{3}$$

In other words, the set of RE-models of each rule ρ from P is augmented with the respective exceptions while the sets of RE-models of rules from U are kept untouched.

From the syntactic viewpoint, we want a rule update operator $\oplus$ to return a program $P \oplus U$ with the semantic characterisation (3). This brings us to the following issue: What if no rule exists whose set of RE-models is equal to (2)? In that case, no rule corresponds to the augmented set of RE-models of a rule $\rho \in P$, so the program $P \oplus U$ cannot be constructed. Moreover, such situations may occur quite frequently since a single rule has very limited expressivity. For instance, updating the fact $p.$ by the rule $\sim p \leftarrow q, r.$ may easily result in a set of RE-models representable by the program $\{\, p \leftarrow \sim q., p \leftarrow \sim r. \,\}$ but not representable by any single rule. To keep a firm link to operations on syntactic objects, we henceforth deal with this problem by allowing the inputs and output of rule update operators to be sets of rules *and programs*, which we dub *rule bases*. In other words, the result of updating a rule, i.e. introducing exceptions to it, may be a set of rules, so the result of updating a program may be a rule base. Technically, a rule base can capture any possible result of an exception-driven update due to Prop. 10.

Formally, a *rule base* is any set of rules and programs $\mathcal{R}$. We put $[\![\mathcal{R}]\!]^{\mathsf{C}} = \bigcap_{\Pi \in \mathcal{R}} [\![\Pi]\!]^{\mathsf{C}}$; $\langle\!\langle \mathcal{R} \rangle\!\rangle^{\mathsf{SE}} = \{\, [\![\Pi]\!]^{\mathsf{SE}} \mid \Pi \in \mathcal{R} \,\}$; $[\![\mathcal{R}]\!]^{\mathsf{SE}} = \bigcap \langle\!\langle \mathcal{R} \rangle\!\rangle^{\mathsf{SE}}$; $\langle\!\langle \mathcal{R} \rangle\!\rangle^{\mathsf{RE}} = \{\, [\![\Pi]\!]^{\mathsf{RE}} \mid \Pi \in \mathcal{R} \,\}$; $[\![\mathcal{R}]\!]^{\mathsf{RE}} = \bigcap \langle\!\langle \mathcal{R} \rangle\!\rangle^{\mathsf{RE}}$; $\mathcal{R}^J = \left\{\, \Pi^J \;\middle|\; \Pi \in \mathcal{R} \,\right\}$ for every interpretation J. We say that J is a *stable model of* $\mathcal{R}$ if J is a subset-minimal C-model of $\mathcal{R}^J$. The set of stable models of $\mathcal{R}$ is denoted by $[\![\mathcal{R}]\!]^{\mathsf{SM}}$. All notions of program equivalence are extended to rule bases by using the same definition. Note that a program is a special case of a rule base. Each element Π of a rule base, be it a rule or a program, represents an *atomic* piece of information. Exception-driven update operators view and manipulate Π only through its set of RE-models $[\![\Pi]\!]^{\mathsf{RE}}$. Due to this, we refer to all such elements Π as *rules*, even if formally they may actually be programs.

Having resolved this issue, we can proceed to the definition of an exception-driven rule update operator.

Definition 12 (Exception-Driven Rule Update Operator). *A rule update operator $\oplus$ is* exception-driven *if for some exception function ε, $\langle\!\langle \mathcal{R} \oplus \mathcal{U} \rangle\!\rangle^{\mathsf{RE}}$ is equal to*

$$\left\{\, [\![\Pi]\!]^{\mathsf{RE}} \cup \varepsilon\left([\![\Pi]\!]^{\mathsf{RE}}, \langle\!\langle \mathcal{R} \rangle\!\rangle^{\mathsf{RE}}, \langle\!\langle \mathcal{U} \rangle\!\rangle^{\mathsf{RE}}\right) \;\middle|\; \Pi \in \mathcal{R} \,\right\} \cup \langle\!\langle \mathcal{U} \rangle\!\rangle^{\mathsf{RE}} \tag{4}$$

for all rule bases $\mathcal{R}$, $\mathcal{U}$. In that case we also say that $\oplus$ is ε-driven.

Note that for each exception function ε there is a whole class of ε-driven rule update operators that differ in the syntactic representations of the sets of RE-models in (4).

Simple Exception Functions

Of particular interest to us is a constrained class of exception functions that requires less information to determine the resulting exceptions. Not only does it lead to simpler definitions and to modular, more efficient implementations, but the study of restricted classes of exception functions is also essential in order to understand their expressivity, i.e. the types of update operators they are able to capture. We focus on exception functions that produce exceptions based on conflicts between pairs of rules, one from the original and one from the updating program, while ignoring the context in which these rules are situated. More formally:

Definition 13 (Simple Exception Function). *An exception function ε is* simple *if for all $R \subseteq \mathcal{X}$ and $\mathcal{M}, \mathcal{N} \subseteq 2^{\mathcal{X}}$,*

$$\varepsilon(R, \mathcal{M}, \mathcal{N}) = \textstyle\bigcup_{S \in \mathcal{N}} \delta(R, S)$$

where $\delta : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is a local exception function. *If $\oplus$ is an ε-driven rule update operator, then we also say that $\oplus$ is δ-*driven*, that δ* generates *$\oplus$ and that $\oplus$ is* simple.

As we shall see, despite their local nature, particular simple exception functions generate rule update operators that satisfy all syntactic properties laid out in Sect. 2 and are closely related to the justified update semantics for DLPs.

Our goal is now to investigate concrete local exception functions δ that generate rule update operators with interesting properties. We take inspiration from the rule update semantics based on the causal rejection principle. However, the relevant concepts, such as that of a *conflict* or *rule rejection*, rely on rule syntax to which an exception function has no direct access. Our goal is thus to investigate similar concepts on the semantic level. In particular, we need to define conditions under which two sets of RE-models are in conflict. Similarly as with rule rejection in the syntactic case, we define these conflicts w.r.t. a two-valued interpretation. We first introduce two preparatory concepts.

We define a *truth value substitution* as follows: Given an interpretation J, an atom p and a truth value $\mathsf{V} \in \{\, \mathsf{T}, \mathsf{U}, \mathsf{F} \,\}$, by $J[\mathsf{V}/p]$ we denote the three-valued interpretation X such that $X(p) = \mathsf{V}$ and $X(q) = J(q)$ for all atoms $q \neq p$.

This enables us to introduce the main concept needed for defining a conflict between two sets of three-valued interpretations. Given a set of three-valued interpretations R, an atom p, a truth value V_0 and a two-valued interpretation J, we say that *R forces p to have the truth value V_0 w.r.t. J*, denoted by $R^J(p) = \mathsf{V}_0$, if

$$J[\mathsf{V}/p] \in R \text{ if and only if } \mathsf{V} = \mathsf{V}_0 \quad .$$

In other words, the three-valued interpretation $J[\mathsf{V}_0/p]$ must be the unique member of R that either coincides with J or differs from it only in the truth value of p. Note that $R^J(p)$ stays undefined in case no V_0 with the above property exists.

Two sets of three-valued interpretations R, S are in *conflict on atom p w.r.t. J*, denoted by $R \bowtie_p^J S$, if $R^J(p) \neq S^J(p)$. The following example illustrates all these concepts.

Example 14. *Consider rules $\rho = (p.)$, $\sigma = (\sim p \leftarrow \sim q.)$*

with the respective sets of RE-models[2]

$$R = \{\ \langle p,p\rangle, \langle p,pq\rangle, \langle pq,pq\rangle\ \}\ \ ,$$
$$S = \{\ \langle \emptyset,\emptyset\rangle, \langle \emptyset,q\rangle, \langle q,q\rangle, \langle \emptyset,pq\rangle, \langle p,pq\rangle, \langle q,pq\rangle, \langle pq,pq\rangle\ \}\ .$$

Intuitively, R forces p to T w.r.t. all interpretations and S forces p to F w.r.t. interpretations in which q is false. Formally it follows that $R^{\emptyset}(p) = \mathsf{T}$ because $\langle p,p\rangle$ belongs to R and neither $\langle \emptyset,p\rangle$ nor $\langle \emptyset,\emptyset\rangle$ belongs to R. Similarly it follows that $S^{\emptyset}(p) = \mathsf{F}$. Hence, $R \bowtie_p^{\emptyset} S$. Using similar arguments we can conclude that $R \bowtie_p^{p} S$. However, it does not hold that $R \bowtie_p^{pq} S$ because $S^{pq}(p)$ is undefined.

We are ready to introduce the local exception function δ_{a}.

Definition 15 (Local Exception Function δ_{a}). *The local exception function δ_{a} is for all $R, S \subseteq \mathcal{X}$ defined as*

$$\delta_{\mathsf{a}}(R,S) = \{\ \langle I,J\rangle \in \mathcal{X} \mid \exists p : R \bowtie_p^J S\ \}\ \ .$$

Thus, if there is a conflict on some atom w.r.t. J, the exceptions introduced by δ_{a} are of the form $\langle I,J\rangle$ where I is an arbitrary subset of J. This means that δ_{a} introduces as exceptions all three-valued interpretations that preserve false atoms from J while the atoms that are true in J may be either true or undefined. This is somewhat related to the definition of a stable model where the default assumptions (false atoms) are fixed while the necessary truth of the remaining atoms is checked against the rules of the program. The syntactic properties of δ_{a}-driven operators are as follows.

Theorem 16 (Syntactic Properties of δ_{a}). *Every δ_{a}-driven rule update operator respects support and fact update. Furthermore, it also respects causal rejection and acyclic justified update w.r.t. DLPs of length at most two.*

This means that δ_{a}-driven rule update operators enjoy a combination of desirable syntactic properties that operators based on SE-models do not (c.f. Thm. 2). However, these operators diverge from causal rejection, even on acyclic DLPs, when more than one update is performed.

Example 17. *Consider again the rules ρ, σ and their sets of RE-models R, S from Example 14 and some δ_{a}-driven rule update operator $\oplus$. Then $\langle\!\langle \{\ \rho\ \} \oplus \{\ \sigma\ \} \rangle\!\rangle^{\mathsf{RE}}$ will contain two elements: R' and S, where $R' = R \cup \delta_{\mathsf{a}}(R,S) = R \cup \{\ \langle \emptyset,\emptyset\rangle, \langle \emptyset,p\rangle\ \}$. An additional update by the fact $\{\ q.\ \}$ then leads to the characterisation $\langle\!\langle \bigoplus \langle \{\ \rho\ \}, \{\ \sigma\ \}, \{\ q.\ \}\rangle \rangle\!\rangle^{\mathsf{RE}}$ which contains three elements: R'', S and T where $R'' = R' \cup \{\ \langle \emptyset,q\rangle, \langle q,q\rangle\ \}$ and T is the set of RE-models of q.*

Furthermore, due to Prop. 9, the interpretation $J = \{\ q\ \}$ is a stable model of $\bigoplus\langle \{\ \rho\ \}, \{\ \sigma\ \}, \{\ q.\ \}\rangle$ because $\langle q,q\rangle$ belongs to all members of $\langle\!\langle \bigoplus \langle \{\ \rho\ \}, \{\ \sigma\ \}, \{\ q.\ \}\rangle \rangle\!\rangle^{\mathsf{RE}}$ and $\langle \emptyset,q\rangle$ does not. However, J violates causal rejection and it is not a justified update model of $\langle \{\ \rho\ \}, \{\ \sigma\ \}, \{\ q.\ \}\rangle$.

This shortcoming of δ_{a} can be overcome as follows:

Definition 18 (Local Exception Functions δ_{b}, δ_{c}). *The local exception functions δ_{b}, δ_{c} are for all $R, S \subseteq \mathcal{X}$ defined as*

$$\delta_{\mathsf{b}}(R,S) = \{\ \langle I,K\rangle \in \mathcal{X} \mid \exists J\, \exists p : R \bowtie_p^J S \wedge I \subseteq J \subseteq K \wedge (p \in K \setminus I \Longrightarrow K = J)\ \}\ \ ,$$
$$\delta_{\mathsf{c}}(R,S) = \mathcal{X} \text{ if } R = S\text{; otherwise } \delta_{\mathsf{c}}(R,S) = \delta_{\mathsf{b}}(R,S)\ \ .$$

The functions δ_{b} and δ_{c} introduce more exceptions than δ_{a}. A conflict on p w.r.t. J leads to the introduction of interpretations in which atoms either maintain the truth value they had in J, or they become undefined. They must also satisfy an extra condition: when p becomes undefined, no other atom may pass from false to undefined. Interestingly, this leads to operators that satisfy all syntactic properties.

Theorem 19 (Syntactic Properties of δ_{b} and δ_{c}). *Let $\oplus$ be a δ_{b}- or δ_{c}-driven rule update operator. Then $\oplus$ respects support, fact update, causal rejection, acyclic justified update.*

The difference between δ_{b} and δ_{c} is in that δ_{c} additionally "wipes out" rules from the original program that are repeated in the update by introducing all interpretations as exceptions to them, rendering them tautological.

Example 20. *Consider again the rules ρ, σ and their sets of RE-models R, S from Example 14. Furthermore, let $P = \{\ \rho\ \}$ and $U = \{\ \rho, \sigma\ \}$.*

If $\oplus$ is δ_{b}-driven, then $\langle\!\langle P \oplus U \rangle\!\rangle^{\mathsf{RE}}$ contains three elements: R, S and $R \cup \delta_{\mathsf{b}}(R,S) = R \cup \{\ \langle \emptyset,\emptyset\rangle, \langle \emptyset,p\rangle, \langle \emptyset,q\rangle, \langle \emptyset,pq\rangle\ \}$. Equivalently, $P \oplus U$ contains rules that are RE-equivalent to the rules ρ, σ and $p \leftarrow q$. Notice that the latter rule is a weakened version of the rule ρ.

On the other hand, if $\oplus$ is δ_{c}-driven, then $\langle\!\langle P \oplus U \rangle\!\rangle^{\mathsf{RE}}$ contains the sets R, S and $R \cup \delta_{\mathsf{c}}(R,R) \cup \delta_{\mathsf{c}}(R,S) = \mathcal{X}$. Equivalently, $P \oplus U$ contains rules that are RE-equivalent to the rules ρ, σ and to the canonical tautology τ.

The differences between δ_{b}- and δ_{c}-driven rule update operators will become more pronounced when we examine their semantic properties.

Before that, however, it is worth noting that δ_{b}- and δ_{c}-driven operators are very closely related to the justified update semantics, even on programs with cycles. They diverge from it only on rules with an appearance of the same atom in both the head and body. Formally, we say a rule is a *local cycle* if $(H(\rho)^+ \cup H(\rho)^-) \cap (B(\rho)^+ \cup B(\rho)^-) \neq \emptyset$.

Theorem 21. *Let $\mathcal{P}$ be a DLP, J an interpretation and $\oplus$ a δ_{b}- or δ_{c}-driven rule update operator. Then,*

- $[\![\bigoplus \mathcal{P}]\!]^{\mathsf{SM}} \subseteq [\![\mathcal{P}]\!]^{\mathsf{JU}}$ *and*
- *if* $\mathsf{all}(\mathcal{P})$ *contains no local cycles, then* $[\![\mathcal{P}]\!]^{\mathsf{JU}} \subseteq [\![\bigoplus \mathcal{P}]\!]^{\mathsf{SM}}$.

This means that up to the marginal case of local cycles, δ_{b} and δ_{c} can be seen as semantic characterisations of the justified update semantics: they lead to stable models that, typically, coincide with justified update models. This tight relationship also sheds new light on the problem of *state condensing* where the goal is to transform a DLP into a single program over the same alphabet that would behave just as the original DLP when further updates are performed. While this cannot be done if the result must be a non-disjunctive program (Leite 2003), it follows from Thm. 21 that a rule base is sufficiently expressive. It stays an open question whether a disjunctive program would suffice or not.

[2] We sometimes omit the usual set notation when we write interpretations. For example, instead of $\{\ p,q\ \}$ we write pq.

Corollary 22 (State Condensing into a Rule Base). *Let $\mathcal{P} = \langle P_i \rangle_{i<n}$ be a DLP such that* $\mathsf{all}(\mathcal{P})$ *contains no local cycles, $\oplus$ be an δ_b- or δ_c-driven rule update operator and $j < n$. Then there exists a rule base $\mathcal{R}$ such that $[\![\mathcal{P}]\!]^{\mathsf{JU}} = [\![\bigoplus \mathcal{P}']\!]^{\mathsf{SM}}$ where $\mathcal{P}' = \langle \mathcal{R}, P_{j+1}, \ldots, P_{n-1} \rangle$.*

Semantic Properties

We proceed by examining further properties of rule update operators – of those generated by simple exception functions in general, and of the δ_a-, δ_b- and δ_c-driven ones in particular. The properties we consider in this section are *semantic* in that they put conditions on the *models* of a result of an update and do not need to refer to the syntax of the original and updating programs. Our results are summarised in Table 1; in the following we explain and discuss them.

The properties in the upper part of the table were introduced in (Eiter et al. 2002; Alferes et al. 2005; Delgrande, Schaub, and Tompits 2007). All of them are formalised for non-disjunctive programs P, Q, U, V and a rule update operator $\oplus$. Each can be seen as a *meta-property* that is instantiated once we adopt a particular notion of program equivalence. Therefore, each row of Table 1 has six cells that stand for particular instantiations of the property. This provides a more complete picture of how simple rule update operators, properties and program equivalence are interrelated.

Unless stated otherwise (in a footnote), each tick (✓) signifies that the property in question holds for *all* simple rule update operators. A missing tick signifies that the property does not hold in general for simple rule update operators and, in particular, there are δ_a-, δ_b- and δ_c-driven operators for which it is violated. A tick is smaller if it is a direct consequence of a preceding larger tick in the same row and of the interrelations between the notions of program equivalence (c.f. Fig. 1).

The lower part of Table 1 contains a straightforward reformulation of the first six KM postulates for programs. We omit the last two postulates as they require a definition of program disjunction and it is not clear how an approapriate one can be obtained. Furthermore, (PU7) has been heavily criticised in the literature as being mainly a means to achieve formal results instead of an intuitive principle (Herzig and Rifi 1999). And though (PU8) reflects basic intuition behind belief update – that of updating each model independently of the others – we believe that such point of view is hardly transferable to knowledge represented using rules because a single model, be it a stable, C-, SE- or RE-model, fails to encode the interdependencies between atoms expressed in rules that are necessary for properties such as support.

Some KM postulates also require a notion of entailment.

Definition 23 (Program Entailment). *Let $\mathcal{R}, \mathcal{S}$ be rule bases, $\mathcal{R}^\tau = \mathcal{R} \cup \{\, \tau \,\}$, $\mathcal{S}^\tau = \mathcal{S} \cup \{\, \tau \,\}$. We write*

$\mathcal{R} \models_{\mathsf{SE}} \mathcal{S}$ *whenever* $[\![\mathcal{R}]\!]^{\mathsf{SE}} \subseteq [\![\mathcal{S}]\!]^{\mathsf{SE}}$;
$\mathcal{R} \models_{\mathsf{RE}} \mathcal{S}$ *whenever* $[\![\mathcal{R}]\!]^{\mathsf{RE}} \subseteq [\![\mathcal{S}]\!]^{\mathsf{RE}}$;
$\mathcal{R} \models_{\mathsf{SR}} \mathcal{S}$ *whenever* $\langle\!\langle \mathcal{R}^\tau \rangle\!\rangle^{\mathsf{SE}} \supseteq \langle\!\langle \mathcal{S}^\tau \rangle\!\rangle^{\mathsf{SE}}$;
$\mathcal{R} \models_{\mathsf{RR}} \mathcal{S}$ *whenever* $\langle\!\langle \mathcal{R}^\tau \rangle\!\rangle^{\mathsf{RE}} \supseteq \langle\!\langle \mathcal{S}^\tau \rangle\!\rangle^{\mathsf{RE}}$;
$\mathcal{R} \models_{\mathsf{SU}} \mathcal{S}$ *whenever* $[\![\mathcal{S} \setminus \mathcal{R}]\!]^{\mathsf{SE}} = \mathcal{X}$.

In terms of strength, the defined notions of entailment maintain all the relationships depicted in Fig. 1. They are also in line with the associated notions of equivalence:

Proposition 24. *For X one of SE, RE, SR, RR, SU and rule bases $\mathcal{R}$ and $\mathcal{S}$, $\mathcal{R} \equiv_\mathsf{X} \mathcal{S}$ if and only if $\mathcal{R} \models_\mathsf{X} \mathcal{S}$ and $\mathcal{S} \models_\mathsf{X} \mathcal{R}$.*

Note that we refrain from defining SM-entailment, mainly due to the fact that stable models are non-monotonic and the usage of entailment in KM postulates is clearly a monotonic one. For instance, (PU1) requires that $\mathcal{R} \oplus \mathcal{U} \models \mathcal{U}$, though there is no reason for $\mathcal{R} \oplus \mathcal{U}$ to have less or the same stable models as $\mathcal{U}$. Therefore, KM postulates that refer to entailment have the SM column marked as "n/a".

At a first glance, it is obvious that none of the semantic properties is satisfied under SU-equivalence. This is because the conditions placed on a rule update operator by an exception function are at the semantic level, while SU-equivalence effectively compares programs syntactically. For instance, an exception-driven operator $\oplus$, for any exception function ε, may behave as follows: $\emptyset \oplus \{\, \sim p \leftarrow p. \,\} = \{\, \leftarrow p. \,\}$. This is because the rules before and after update are RE-equivalent. However, due to the fact that the programs $\{\, \sim p \leftarrow p. \,\}$ and $\{\, \leftarrow p. \,\}$ are considered different under SU-equivalence, $\oplus$ cannot satisfy Initialisation w.r.t. SU-equivalence. The situation with all other properties is analogous.

A closer look at Table 1 reveals that many properties are satisfied "by construction", regardless of which simple rule update operator we consider and of which notion of equivalence we pick. It also turns out that most simple rule update operators, including δ_a-, δ_b- and δ_c-driven ones, naturally satisfy Tautology and Immunity to Tautologies, properties that are generally acknowledged as very desirable although most existing rule update semantics fail to comply with them.

The properties of Idempotence, Absorption and Augmentation are the only ones that reveal differences amongst δ_a, δ_b and δ_c. They are not satisfied by δ_a- and δ_b-driven operators under SR- and RR-equivalence. The reason for this is that when a program is updated by a program that includes the original program, exceptions may still be introduced to some rules, resulting in weakened versions of the original rules . Since such rules are not part of the original program, the programs before and after update are considered different under SR- and RR-equivalence. As illustrated in Example 20, this problem is dodged in δ_c by completely eliminating original rules that also appear in the update.

Nevertheless, this also seems to indicate that SR- and RR-equivalence are slightly too strong for characterising updates because programs such as such as $\{\, p. \,\}$ and $\{\, p., p \leftarrow q. \,\}$ are not considered equivalent even though we expect the same behaviour from them when they are updated. We speculated in (Slota and Leite 2011) that this could be solved by adopting a weaker equivalence: *SMR-equivalence*. However, it turns out that SMR-equivalence is too weak because programs such as $\{\, \sim q. \,\}$ and $\{\, \sim q., p \leftarrow q. \,\}$ are SMR-equivalent although, when updated by $\{\, q. \,\}$, different results are expected for each of them. The same applies to a counterpart of SMR-equivalence based on RE-models.

Moreover, δ_a-driven operators fail to satisfy Absorption

Table 1: Semantic properties of simple rule update operators.

Property	Formalisation	Type of $\equiv$, $\models$ and $[\![\cdot]\!]$					
		SU	RR	SR	RE	SE	SM
Initialisation	$\emptyset \oplus U \equiv U$.		✓	✓	✓	✓	✓
Disjointness	If P, Q are over disjoint alphabets, then $(P \cup Q) \oplus U \equiv (P \oplus U) \cup (Q \oplus U)$.		✓	✓	✓	✓	✓
Non-interference	If U, V are over disjoint alphabets, then $(P \oplus U) \oplus V \equiv (P \oplus V) \oplus U$.		✓[abc]	✓[abc]	✓[abc]	✓[abc]	✓[abc]
Tautology	If U is tautological, then $P \oplus U \equiv P$.		✓[⋆]	✓[⋆]	✓[⋆]	✓[⋆]	✓[⋆]
Immunity to Tautologies	If Q and V are tautological, then $(P \cup Q) \oplus (U \cup V) \equiv P \oplus U$.		✓[⋆]	✓[⋆]	✓[⋆]	✓[⋆]	✓[⋆]
Idempotence	$P \oplus P \equiv P$.		✓[c]	✓[c]	✓	✓	✓
Absorption	$(P \oplus U) \oplus U \equiv P \oplus U$.		✓[c]	✓[c]	✓[bc]	✓[bc]	✓[bc]
Augmentation	If $U \subseteq V$, then $(P \oplus U) \oplus V \equiv P \oplus V$.		✓[c]	✓[c]	✓[bc]	✓[bc]	✓[bc]
Associativity	$P \oplus (U \oplus V) \equiv (P \oplus U) \oplus V$.						
(PU1)	$P \oplus U \models U$		✓	✓	✓	✓	n/a
(PU2.$\top$)	$P \oplus \emptyset \equiv P$		✓	✓	✓	✓	✓
(PU2.1)	$P \cup U \models P \oplus U$				✓	✓	n/a
(PU2.2)	$(P \cup U) \oplus U \models P$						n/a
(PU3)	If $[\![P]\!] \neq \emptyset$ and $[\![U]\!] \neq \emptyset$, then $[\![P \oplus U]\!] \neq \emptyset$	n/a	n/a	n/a			
(PU4)	If $P \equiv Q$ and $U \equiv V$, then $P \oplus U \equiv Q \oplus V$		✓[⋆]				
(PU5)	$(P \oplus U) \cup V \models P \oplus (U \cup V)$				✓	✓	n/a
(PU6)	If $P \oplus U \models V$ and $P \oplus V \models U$, then $P \oplus U \equiv P \oplus V$						n/a

[a] Holds if $\oplus$ is generated by δ_{a}.
[b] Holds if $\oplus$ is generated by δ_{b}.
[c] Holds if $\oplus$ is generated by δ_{c}.
[⋆] Holds if $\oplus$ is generated by a local exception function δ such that $\delta(R, \mathcal{X}) \subseteq R$ for all $R \subseteq \mathcal{X}$. This is satisfied by δ_{a}, δ_{b}, δ_{c}.

and Augmentation. Along with Thm. 16, this seems to indicate that δ_{a} does not correctly handle iterated updates.

Associativity is one of the few properties that is not satisfied by any of the defined classes of operators. This is closely related to the question of whether *rejected rules are allowed to reject*, studied in literature on causal rejection-based update semantics (Leite 2003). Associativity can be seen as postulating that an update operator must behave the same way regardless of whether rejected rules are allowed to reject or not. However, many semantics generate unwanted models when rejected rules are not allowed to reject earlier rules.

Turning to KM postulates, (PU2.1) is not satisfied under SR- and RR-equivalence for the same reasons, described above, that prevent δ_{a}- and δ_{b}-driven operators from satisfying Idempotence. The situation with (PU5) is the same since it implies (PU2.1) in the presence of (PU2.$\top$).

Postulate (PU2.2) requires that $\{ p., {\sim}p. \} \oplus \{ {\sim}p. \} \models p$ which, in the presence of (PU1), amounts to postulating that one can never recover from an inconsistent state, contrary to most rule update semantics which do allow for recovery from such states. The case of (PU6) is the same since it implies (PU2.2) in the presence of (PU1) and (PU2.$\top$).

Postulate (PU3) relies on a function that returns the set of models of a rule base. Thus, $[\![\cdot]\!]^{\mathsf{SM}}$, $[\![\cdot]\!]^{\mathsf{SE}}$ and $[\![\cdot]\!]^{\mathsf{RE}}$ can be used for this purpose while the other columns in the corresponding row in Table 1 make little sense, so they are marked as "n/a". Furthermore, this postulate is not satisfied by any of the defined classes of exception-driven operators. It is also one of the principles that most existing approaches to rule update chronically fail to satisfy. In order to satisfy it, a context-aware exception function would have to be used because conflicts may arise between more than two rules that are otherwise pairwise consistent. For instance, when updating $\{ p. \}$ by $\{ q \leftarrow p., {\sim}q \leftarrow p. \}$, one would somehow need to detect the joint conflict between these three rules. This is however impossible with a simple exception function because it only considers conflicts between pairs of rules, one from the original program and one from the update.

Finally, (PU4) is the postulate that requires update operators to be syntax-independent. As motivated in the introduction, the failure to satisfy it under SM-, SE- and RE-equivalence is inevitable if properties such as support and fact update are to be respected. On the other hand, due to the semantic underpinning of simple rule update op-

erators, (PU4) is satisfied by most of them, including all δ_{a}-, δ_{b}- and δ_{c}-driven ones, under RR-equivalence. It might be interesting to look for constrained classes of exception functions that satisfy syntax-independence w.r.t. SR-equivalence. Such functions, however, will not be able to satisfy the causal rejection principle anymore because of the inability of SE-models to distinguish abolishing rules.

5 Concluding Remarks

We defined a new monotonic characterisation of rules, the RE-models, and introduced a generic method for specifying semantic rule update operators in which a logic program is viewed as the *set of sets of RE-models of its rules* and updates are performed by introducing additional interpretations to the sets of RE-models of rules in the original program. This framework allowed us to define concrete update operators that enjoy an interesting combination of syntactic as well as semantic properties that had never been reconciled before. These findings are essential to better understand the interrelations between belief update and syntax-based rule update semantics and so serve as stepping stones to addressing updates of Hybrid Knowledge Bases.

Our investigation directly points to challenges that need to be tackled next. First, semantic characterisations of additional rule update semantics need to be investigated. This poses a number of challenges due to the need to detect non-tautological irrelevant updates (Alferes et al. 2005; Šefránek 2006; 2011). For instance, simple functions examined in this paper cannot distinguish an update of $\{\, p. \,\}$ by $U = \{\, \sim p \leftarrow \sim q., \sim q \leftarrow \sim p \,\}$, where it is plausible to introduce the exception $\langle \emptyset, \emptyset \rangle$, from an update of $\{\, p., q. \,\}$ by U, where such an exception should not be introduced due to the cyclic dependency of justifications to reject $p.$ and $q.$ In such situations, context-aware functions need to be used. In addition, such functions have the potential of satisfying properties such as (PU3) and (Associativity).

Another challenge is to find further logical characterisations of rules, namely a notion of program equivalence that is weaker than RR-equivalence but stronger than RE-equivalence so that both (PU4) and properties such as (PU2.1) can be achieved under a single notion of program equivalence.

Finally, computational properties of different classes of exception-driven update operators should be investigated.

Acknowledgements

M. Slota was partially supported by FCT Scholarship SFRH/BD/38214/2007 and FCT Project ERRO PTDC/EIA-CCO/121823/2010. J. Leite was partially supported by FCT Project ASPEN PTDC/EIA-CCO/110921/2009.

References

Alchourrón, C. E.; Gärdenfors, P.; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic* 50(2):510–530.

Alferes, J. J.; Leite, J. A.; Pereira, L. M.; Przymusinska, H.; and Przymusinski, T. C. 2000. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming* 45(1-3):43–70.

Alferes, J. J.; Banti, F.; Brogi, A.; and Leite, J. A. 2005. The refined extension principle for semantics of dynamic logic programming. *Studia Logica* 79(1):7–32.

Apt, K. R., and Bezem, M. 1991. Acyclic programs. *New Generation Computing* 9(3/4):335–364.

Apt, K. R.; Blair, H. A.; and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufmann. 89–148.

Boutilier, C. 1995. Generalized update: Belief change in dynamic settings. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1550–1556. Montral, Québec, Canada: Morgan Kaufmann.

Brewka, G., and Hertzberg, J. 1993. How to do things with worlds: On formalizing actions and plans. *Journal of Logic and Computation* 3(5):517–532.

Cabalar, P.; Pearce, D.; and Valverde, A. 2007. Minimal logic programs. In Dahl, V., and Niemelä, I., eds., *Proceedings of the 23rd International Conference on Logic Programming (ICLP 2007)*, volume 4670 of *Lecture Notes in Computer Science*, 104–118. Porto, Portugal: Springer.

de Bruijn, J.; Pearce, D.; Polleres, A.; and Valverde, A. 2010. A semantical framework for hybrid knowledge bases. *Journal of Knowledge and Information Systems* 25(1):81–104.

de Bruijn, J.; Eiter, T.; Polleres, A.; and Tompits, H. 2011. Embedding nonground logic programs into autoepistemic logic for knowledge-base combination. *ACM Transactions on Computational Logic (TOCL)* 12(3):20.

Delgrande, J. P.; Schaub, T.; Tompits, H.; and Woltran, S. 2008. Belief revision of logic programs under answer set semantics. In Brewka, G., and Lang, J., eds., *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR 2008)*, 411–421. Sydney, Australia: AAAI Press.

Delgrande, J. P.; Schaub, T.; and Tompits, H. 2007. A preference-based framework for updating logic programs. In Baral, C.; Brewka, G.; and Schlipf, J. S., eds., *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, volume 4483 of *Lecture Notes in Computer Science*, 71–83. Tempe, AZ, USA: Springer.

Dix, J. 1995. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae* 22(3):257–288.

Doherty, P.; Lukaszewicz, W.; and Madalinska-Bugaj, E. 1998. The PMA and relativizing minimal change for action update. In Cohn, A. G.; Schubert, L. K.; and Shapiro, S. C., eds., *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, 258–269. Trento, Italy: Morgan Kaufmann.

Eiter, T.; Fink, M.; Sabbatini, G.; and Tompits, H. 2002. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming (TPLP)* 2(6):721–777.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In Kowalski, R. A., and Bowen, K. A., eds., *Proceedings of the 5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, 1070–1080. Seattle, Washington: MIT Press.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9(3-4):365–385.

Herzig, A., and Rifi, O. 1999. Propositional belief base update and minimal change. *Artificial Intelligence* 115(1):107–138.

Heyting, A. 1930. Die formalen Regeln der intuitionistischen Logik. *Sitzungsberichte der Preussischen Akademie der Wissenschaften* 42–56. Reprint in *Logik-Texte: Kommentierte Auswahl zur Geschichte der Modernen Logik*, Akademie-Verlag, 1986.

Homola, M. 2004. Dynamic logic programming: Various semantics are equal on acyclic programs. In Leite, J. A., and Torroni, P., eds., *Proceedings of the 5th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, volume 3487 of *Lecture Notes in Computer Science*, 78–95. Lisbon, Portugal: Springer.

Inoue, K., and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming* 35(1):39–78.

Inoue, K., and Sakama, C. 2004. Equivalence of logic programs under updates. In Alferes, J. J., and Leite, J. A., eds., *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *Lecture Notes in Computer Science*, 174–186. Lisbon, Portugal: Springer.

Janhunen, T. 2001. On the effect of default negation on the expressiveness of disjunctive rules. In Eiter, T.; Faber, W.; and Truszczynski, M., eds., *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *Lecture Notes in Computer Science*, 93–106. Vienna, Austria: Springer.

Katsuno, H., and Mendelzon, A. O. 1991. On the difference between updating a knowledge base and revising it. In Allen, J. F.; Fikes, R.; and Sandewall, E., eds., *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, 387–394. Cambridge, MA, USA: Morgan Kaufmann Publishers.

Knorr, M.; Alferes, J. J.; and Hitzler:, P. 2011. Local closed world reasoning with description logics under the well-founded semantics. *Artificial Intelligence* 175(9-10):1528–1554.

Krümpelmann, P. 2012. Dependency semantics for sequences of extended logic programs. *Logic Journal of the IGPL*. To appear.

Leite, J. A., and Pereira, L. M. 1997. Generalizing updates: From models to programs. In Dix, J.; Pereira, L. M.; and Przymusinski, T. C., eds., *Proceedings of the 3rd International Workshop on Logic Programming and Knowledge Representation (LPKR '97)*, volume 1471 of *Lecture Notes in Computer Science*, 224–246. Port Jefferson, New York, USA: Springer.

Leite, J. A. 2003. *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover*. IOS Press.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic (TOCL)* 2(4):526–541.

Marek, V. W., and Truszczynski, M. 1998. Revision programming. *Theoretical Computer Science* 190(2):241–277.

Motik, B., and Rosati, R. 2010. Reconciling description logics and rules. *Journal of the ACM* 57(5):93–154.

Osorio, M., and Cuevas, V. 2007. Updates in answer set programming: An approach based on basic structural properties. *Theory and Practice of Logic Programming* 7(4):451–479.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In Dix, J.; Pereira, L. M.; and Przymusinski, T. C., eds., *Proceedings of the 6th Workshop on Non-Monotonic Extensions of Logic Programming (NMELP '96)*, volume 1216 of *Lecture Notes in Computer Science*, 57–70. Bad Honnef, Germany: Springer.

Sakama, C., and Inoue, K. 2003. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming (TPLP)* 3(6):671–713.

Šefránek, J. 2006. Irrelevant updates and nonmonotonic assumptions. In Fisher, M.; van der Hoek, W.; Konev, B.; and Lisitsa, A., eds., *Proceedings of the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006)*, volume 4160 of *Lecture Notes in Computer Science*, 426–438. Liverpool, UK: Springer.

Šefránek, J. 2011. Static and dynamic semantics. In *Special Session of the 10th Mexican International Conference on Artificial Intelligence (MICAI 2011)*. To appear.

Slota, M., and Leite, J. 2010. On semantic update operators for answer-set programs. In Coelho, H.; Studer, R.; and Wooldridge, M., eds., *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, 957–962. Lisbon, Portugal: IOS Press.

Slota, M., and Leite, J. 2011. Back and forth between rules and SE-models. In Delgrande, J. P., and Faber, W., eds., *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-11)*, volume 6645 of *Lecture Notes in Computer Science*, 174–186. Vancouver, Canada: Springer.

Turner, H. 2003. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming (TPLP)* 3(4-5):609–622.

Winslett, M. 1990. *Updating Logical Databases*. New York, USA: Cambridge University Press.

Zhang, Y., and Foo, N. Y. 2005. A unified framework for representing logic program updates. In Veloso, M. M., and Kambhampati, S., eds., *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI 2005)*, 707–713. Pittsburgh, Pennsylvania, USA: AAAI Press / The MIT Press.

Zhang, Y. 2006. Logic program-based updates. *ACM Transactions on Computational Logic* 7(3):421–472.