TPLP **23** (1): 57–110, 2023.

doi:10.1017/S1471068422000060 First published online 22 March 2022

*A Brief History of Updates of Answer-Set Programs**

JOÃO LEITE and MARTIN SLOTA
NOVA LINCS, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Lisbon, Portugal
(*e-mails:* jleite@fct.unl.pt, martin.slota@gmail.com)

submitted 1 August 2020; revised 26 December 2021; accepted 14 February 2022

Abstract

Over the last couple of decades, there has been a considerable effort devoted to the problem of updating logic programs under the stable model semantics (a.k.a. answer-set programs) or, in other words, the problem of characterising the result of bringing up-to-date a logic program when the world it describes changes. Whereas the state-of-the-art approaches are guided by the same basic intuitions and aspirations as belief updates in the context of classical logic, they build upon fundamentally different principles and methods, which have prevented a unifying framework that could embrace both belief and rule updates. In this paper, we will overview some of the main approaches and results related to answer-set programming updates, while pointing out some of the main challenges that research in this topic has faced.

KEYWORDS: belief update, belief change, logic programming, answer-set programming

1 Introduction

In this paper, we will take a historical journey through some of the main approaches proposed to deal with the problem of updating logic programs under the stable model semantics.

Knowledge-based systems must keep a representation of the world – often encoded in some logic-based language equipped with a formal semantics and reasoning mechanisms – which is then used for reasoning, for example, to automate decision making. Whereas languages that are based on classical logic – hence monotonic – like description logics, are often used, it has been known for several decades that non-monotonic features

*The authors would like to thank José Alferes, Martin Baláz, Federico Banti, Antonio Brogi, Martin Homola, Luís Moniz Pereira, Halina Przymusinska, Teodor C. Przymusinski, and Theresa Swift, with whom they worked on the topic of this paper over the years, as well as Ricardo Gonçalves and Matthias Knorr for valuable comments on an earlier draft of this paper. The authors would also like to thank the anonymous reviewers for their insightful comments and suggestions, which greatly helped us improve this paper. The authors were partially supported by Fundação para a Ciência e Tecnologia through projects FORGET (PTDC/CCI-INF/32219/2017) and RIVER (PTDC/CCI-COM/30952/2017), and strategic project NOVA LINCS (UIDB/04516/2020).

are important for common-sense reasoning, for example, to properly deal with default information, preferences, the frame problem, etc. Of the many existing languages for knowledge representation that exhibit non-monotonic features, logic programming under the stable model semantics (a.k.a. answer-set programming, or ASP, for short), introduced by Gelfond and Lifschitz (1988, 1991), is perhaps the biggest success story so far. Established some 30 years ago, ASP is similar in syntax to traditional logic programming, and has a simple and well-understood non-monotonic declarative semantics with known relationships with other logic-based formalisms such as default logic, autoepistemic logic, propositional and predicate logic, etc. (cf. the work of Lifschitz 2008 and references therein). Its rich expressive power allows to compactly represent all NP and coNP problems if non-disjunctive logic programs are used, while disjunctive logic programs capture the complexity class Σ_2^p and Π_2^p (Eiter *et al*. 1997). Additionally, the existence of efficient implementations such as *clasp* (Gebser *et al*. 2011) and *dlv* (Leone *et al*. 2006) has made it possible to use ASP in significant applications in diverse areas such as configuration, diagnosis and repair, planning, classification, scheduling, robotics, information integration, legal reasoning, computational biology and bioinformatics, e-medicine, and decision support systems (cf. the surveys by Erdem *et al*. 2016; Erdem and Patoglu 2018; Falkner *et al*. 2018, and references therein).

One of the more recent challenges for knowledge engineering and information management is to efficiently and plausibly deal with the incorporation of new, possibly conflicting knowledge and beliefs. There are several domains where this may be required. For example, knowledge-based systems that check for legal and regulatory compliance – such as a public procurement monitoring system – need to keep track of changing laws and regulations, and reason with them. Simply adding the new laws and regulations to the knowledge base containing the older ones would not work, since the newer may be in conflict with the older. It may also not be as simple as deleting the old conflicting ones, since the conflict may be contingent on particular cases. For example, some initial regulation may state that institutions are allowed to enter a contract without a public offer if the contract's value is below some fixed amount, and a later regulation may state that publicly funded foundations (a special kind of an institution) that have not filed their previous year's tax return, are not allowed to enter a contract without a public offer. The knowledge-based system would have to automatically deal with these cases, in a way similar to the legal principle of *lex posterior* used by judges and other legal practitioners, according to which a newer law repeals an earlier conflicting one. Other domains where dealing with dynamic, possibly conflicting, knowledge and beliefs is important include multi-agent systems, where agents need to change their knowledge and beliefs to properly reflect their observations, including incoming messages from other agents, and even new norms, possibly resulting in a change in behaviour; stream reasoning systems that learn/extract knowledge from streams of data, which may be in conflict with previously learnt knowledge; or even transfer learning, where what is learnt by one system in one domain serves as initial knowledge when that system is placed in a different domain, which is then changed as new knowledge is learnt in the new domain. Indeed, any knowledge-based system that maintains a knowledge base about a dynamic world, that is, a world that changes, needs to efficiently and plausibly deal with the incorporation of new, possibly conflicting knowledge.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The problems associated with the evolution of knowledge have been extensively studied over the years, in the context of classical logic. Most of this work was inspired by the seminal contribution of Alchourrón, Gärdenfors and Makinson (AGM) who proposed a set of desirable properties of belief change operators, now called *AGM postulates* (Alchourrón *et al.* 1985). Subsequently, *update* and *revision* have been distinguished as two very much related but ultimately different belief change operations (Keller and Winslett 1985; Winslett 1990; Katsuno and Mendelzon 1991). While revision deals with incorporating new *better* information about a *static* world, update takes place when changes occurring in a *dynamic* world are recorded. Katsuno and Mendelzon formulated a separate set of postulates for update, now known as *KM postulates*. For a comprehensive treatment of the subject and further references on belief change in classical logic, the reader is referred to the survey paper by Fermé and Hansson (2011).

Despite the large body of research on belief change in general, and on updates in particular, in the context of classical logic, the use of ASP for knowledge representation in dynamic domains, mainly due to its rule-based syntax and non-monotonic semantics, has called for a specific line of research on how to update an answer-set program, which constitutes the central topic of this paper.

To illustrate the problem at hand, consider an agent with knowledge represented by the following program $\mathcal{P}$ (where $\sim$ denotes default negation):

$$\mathsf{goHome} \leftarrow \sim\mathsf{money}. \tag{1}$$

$$\mathsf{goRestaurant} \leftarrow \mathsf{money}. \tag{2}$$

$$\mathsf{money}. \tag{3}$$

The only stable model of $\mathcal{P}$ is $I = \{\, \mathsf{money}, \mathsf{goRestaurant} \,\}$, capturing that the agent has money by rule (3), so according to rule (2) it plans to go to a restaurant. Suppose that the beliefs of the agent are to be updated by the program $\mathcal{U}$ with the following two rules:

$$\sim\mathsf{money} \leftarrow \mathsf{robbed}. \qquad\qquad \mathsf{robbed}.$$

What should the agent's beliefs be after the update of $\mathcal{P}$ by $\mathcal{U}$?

The central research question is then how to semantically characterise pairs or, more generally, sequences of ASP programs (a.k.a. *dynamic logic programs*) where each component represents an update of the preceding ones, and, if possible, produce an ASP program that encodes the result of the updates.

Going back to the previous example, when we update $\mathcal{P}$ by $\mathcal{U}$, the intuitively correct result is that robbed is true and money is false, because of the rules in the update $\mathcal{U}$, and that goRestaurant should now be false because its only justification, money, is no longer true. Furthermore, we would expect that goHome be now true, that is, the rule (1) should be triggered because money became false.

Over the years, many have tackled this issue, which has proved far more difficult and elusive than perhaps originally thought. Earlier approaches (Marek and Truszczynski 1994, 1998; Przymusinski and Turner 1995, 1997; Alferes and Pereira 1996) were based on literal inertia, following some of the basic principles inherited from the *possible models approach* by Winslett (1988) – an approach to updates in propositional logic based on minimising the set of atoms whose truth value changes when an interpretation is updated that satisfies the KM postulates. However, they were soon found to be inadequate, or at

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

least not sufficiently expressive to capture the result of updating an answer-set program by means of another answer-set program (Leite and Pereira 1998).

Since then, many different approaches were put forward. Though the state-of-the-art approaches are guided by the same basic intuitions and aspirations as belief change in classical logic, they build upon fundamentally different principles and methods. While many are based on the so-called *causal rejection principle* (Leite and Pereira 1998; 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 *et al*. 2007), or dependencies on defeasible assumptions (Šefránek 2011; Krümpelmann 2012).

One interesting feature about these developments is that the resulting operators often bear characteristics of *revision* rather than *update*, as viewed from the perspective of belief change in classical logic, often blurring the frontier between these two types of operations. We will review not only those approaches that aim to deal with updates, but also some operators that are closer to revision, but whose authors position their contribution as having similar goals, namely by comparing them with the operators specifically defined for updates. However, while evaluating the reviewed approaches, we will guide ourselves by the basic idea of what an update is, that is, an operation that deals with incorporating new information about a *dynamic* world, as opposed to a revision which deals with incorporating better knowledge about a *static* world. In our opinion, underlying this definition of update is the assumption that the information acquired at each point in time is correct at that moment, as opposed to the assumption adopted in revision, whereby the information acquired at each time point is not necessarily correct, just better than what we had before. One relevant consequence is that an empty or tautological update (which we take to represent that nothing changed) should have no effect, because our beliefs about the world were correct and nothing changed, while it is acceptable that a revision by an empty or tautological update may lead to some change, for example restoring consistency, because we take it that in revision, our beliefs about the world were not necessarily correct.

More recently, both the AGM and KM postulates were revisited, taking into account a monotonic characterisation of ASP – HT-models (Pearce 1997; Lifschitz *et al*. 2001) – as the basis for new classes of revision and update operators (Delgrande *et al*. 2013; Slota and Leite 2014). Despite their own merits and shortcomings, these approaches based on HT-models have opened up new avenues to investigate other belief change operations in ASP, such as forgetting (Wang *et al*. 2012, 2013, 2014; Delgrande and Wang 2015) (see the paper by Gonçalves *et al*. 2016 for a survey on forgetting in ASP, or another survey in this volume), and new insights into unifying classical logic and ASP-based updates (Slota and Leite 2012a,b; Slota *et al*. 2015).

In this paper, we will briefly revisit the main landmarks in the history of updating answer-set programs. We will follow a chronological approach that divides it into three main eras, although not without intersection – model updates, syntax-based updates, and semantics-based updates – that correspond to the three main sections of this paper. These sections are preceded by a brief section (pre-history) on belief change in classical logic, and followed by an outlook where some current and future issues are discussed. Before we begin the journey, we provide a brief background section with the usual preliminaries on

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

propositional logic, answer-set programming, and other basic concepts used throughout the paper.

Given the nature of this paper, we will often exercise some restraint on technical content in favour of better conveying the main intuitions and concepts underlying each approach and focusing on providing simple examples that bring forward their main differentiating features. Despite this compromise, this paper still conveys a novel and, we hope, significant scientific contribution, inasmuch as it is, to the best of our knowledge, the first encompassing and critical survey of the field, not only comparing different approaches but sometimes even providing intuitions and illustrative examples that were absent from the original papers.

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 $I \subseteq \mathcal{L}$. The set of all (two-valued) interpretations is denoted by $\mathcal{I}$. Each atom p is assigned one of two truth values in I: $I(\mathsf{p}) = \mathsf{T}$ if $\mathsf{p} \in I$ and $I(\mathsf{p}) = \mathsf{F}$ 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 $[\![\phi]\!] = [\![\psi]\!]$.

Answer-Set Programs. Answer-set programming (a.k.a. logic programming under the stable model semantics) has its roots in classical logic. However, answer-set programs diverge from classical semantics by adopting the closed world assumption and allowing for non-monotonic inferences. Here, we introduce the class of answer-set programs that allow for both disjunction and default negation in heads of rules.

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. Throughout this paper, we adopt a convention that double default negation is absorbed, so that $\sim\sim\mathsf{p}$ denotes the atom p. Given a set S of literals, we introduce the following notation: $S^+ = \{\, \mathsf{p} \in \mathcal{L} \mid \mathsf{p} \in S \,\}$, $S^- = \{\, \mathsf{p} \in \mathcal{L} \mid \sim\mathsf{p} \in S \,\}$, and $\sim S = \{\, \sim L \mid L \in S \,\}$.

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

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

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; an *integrity constraint* if its head is empty. A *program* is any set of rules. A program is *non-disjunctive* if all its rules are.

We define the class of *acyclic programs* using level mappings (Apt and Bezem 1991). A *level mapping* is a function ℓ that assigns a natural number to every atom, and is extended to default literals and sets of literals by putting $\ell(\sim L) = \ell(L)$ and $\ell(S) = \max\{\, \ell(L) \mid L \in S \,\}$. We say that a program $\mathcal{P}$ is *acyclic* if there exists a level mapping

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Table 1. *Satisfaction of literals, rules and programs*

$J \models l$ iff $l \in J$
$J \models {\sim}l$ iff $l \notin J$
$J \models S$ iff $J \models L$ for all $L \in S$
$J \models \pi$ iff $\exists L \in B(\pi) : J \not\models L$ or $\exists L \in H(\pi) : J \models L$
$J \models \mathcal{P}$ iff $J \models \pi$ for all $\pi \in \mathcal{P}$

ℓ such that for every rule $\pi \in \mathcal{P}$ it holds that $H(\pi) \neq \emptyset$ and $\ell(l_H) > \ell(l_B)$ for every $l_H \in H(\pi)$ and every $l_B \in B(\pi)$.

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

Satisfaction of programs is obtained by treating rules as classical implications. Table 1 defines satisfaction of literals l and ${\sim}l$, a set of literals S, a rule π and a program $\mathcal{P}$ in an interpretation $J \subseteq \mathcal{L}$. We say that J is a *C-model* (classical model) of a rule π if $J \models \pi$, and a *C-model* of a program $\mathcal{P}$ if $J \models \mathcal{P}$. The set of all C-models of a rule π is denoted by $[\![\pi]\!]_{\mathsf{C}}$ and for any program $\mathcal{P}$, $[\![\mathcal{P}]\!]_{\mathsf{C}} = \bigcap_{\pi \in \mathcal{P}} [\![\pi]\!]_{\mathsf{C}}$. A program $\mathcal{P}$ is *consistent* if $[\![\mathcal{P}]\!]_{\mathsf{C}} \neq \emptyset$, and *inconsistent* otherwise.

The stable and HT-models are defined in terms of reducts. Given a program $\mathcal{P}$ and an interpretation J, the *reduct of* $\mathcal{P}$ *w.r.t.* J is defined as

$$\mathcal{P}^J = \left\{ \langle H(\pi)^+, B(\pi)^+ \rangle \;\middle|\; \pi \in \mathcal{P} \wedge J \not\models \langle {\sim}H(\pi)^-, {\sim}B(\pi)^- \rangle \right\}.$$

An interpretation J is a *stable model* of a program $\mathcal{P}$ if J is a subset-minimal C-model of $\mathcal{P}^J$. The set of all stable models of $\mathcal{P}$ is denoted by $[\![\mathcal{P}]\!]_{\mathsf{SM}}$. A program $\mathcal{P}$ is *coherent* if $[\![\mathcal{P}]\!]_{\mathsf{SM}} \neq \emptyset$, and *incoherent* otherwise.

HT-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: $X(\mathsf{p}) = \mathsf{T}$ if $\mathsf{p} \in I$; $X(\mathsf{p}) = \mathsf{U}$ if $\mathsf{p} \in J \setminus I$; $X(\mathsf{p}) = \mathsf{F}$ if $\mathsf{p} \in \mathcal{L} \setminus J$. The set of all three-valued interpretations is denoted by $\mathfrak{X}$. A three-valued interpretation $\langle I, J \rangle$ is an *HT-model* of a rule π if $J \models \pi$ and $I \models \pi^J$. The set of all HT-models of a rule π is denoted by $[\![\pi]\!]_{\mathsf{HT}}$ and for any program $\mathcal{P}$, $[\![\mathcal{P}]\!]_{\mathsf{HT}} = \bigcap_{\pi \in \mathcal{P}} [\![\pi]\!]_{\mathsf{HT}}$. Note that J is a stable model of $\mathcal{P}$ if and only if $\langle J, J \rangle \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$ and for all $I \subsetneq J$, $\langle I, J \rangle \notin [\![\mathcal{P}]\!]_{\mathsf{HT}}$. Also, $J \in [\![\mathcal{P}]\!]_{\mathsf{C}}$ if and only if $\langle J, J \rangle \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$. We write $(I, J) \models \mathcal{P}$ if $(I, J) \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$. We say that $\mathcal{P}$ *is strongly equivalent to* $\mathcal{Q}$, denoted by $\mathcal{P} \equiv_{\mathsf{HT}} \mathcal{Q}$, if $[\![\mathcal{P}]\!]_{\mathsf{HT}} = [\![\mathcal{Q}]\!]_{\mathsf{HT}}$, and that $\mathcal{P}$ *strongly entails* $\mathcal{Q}$, denoted by $\mathcal{P} \models_{\mathsf{HT}} \mathcal{Q}$, if $[\![\mathcal{P}]\!]_{\mathsf{HT}} \subseteq [\![\mathcal{Q}]\!]_{\mathsf{HT}}$. A rule π is tautological if $[\![\pi]\!]_{\mathsf{HT}} = \mathfrak{X}$. It follows from the results of Inoue and Sakama (2004) and Cabalar *et al.* (2007) that a rule π is tautological if $B(\pi) \cap H(\pi) \neq \emptyset$ or $B(\pi)^+ \cap B(\pi)^- \neq \emptyset$.

The class of programs defined above can be extended to allow for a second form of negation, dubbed *strong negation*. In a nutshell, $\mathcal{L}$ is extended to also include, for each of its original atoms p, its (strong) negation $\neg\mathsf{p}$. Elements of this extended $\mathcal{L}$ are dubbed *objective literals* and we use the following notation to refer to complementary objective

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

literals: $\overline{\mathsf{p}} = \neg\mathsf{p}$ and $\overline{\neg\mathsf{p}} = \mathsf{p}$ for any atom p. Each atom p is interpreted separately of (though still consistently with) its strong negation $\neg\mathsf{p}$. Each interpretation naturally corresponds to a consistent subset of the extended set $\mathcal{L}$. More formally, an *(extended) interpretation* is a subset of $\mathcal{L}$ that does not contain both l and $\bar{l}$ for any objective literal l. Note that this is in contrast with the definition of answer-set by Gelfond and Lifschitz (1991), which allows for certain programs to have their semantics be characterised by the so-called *contradictory answer-set* $\mathcal{L}$.

To simplify notation, first-order atoms with variables are often used in program rules. Such rules should be seen as a shortcut corresponding to the set of rules obtained by replacing the variables with constants to form atoms in $\mathcal{L}$, in all possible ways.

Order Theory. Given a set $\mathcal{S}$, a *preorder over* $\mathcal{S}$ is a reflexive and transitive binary relation over $\mathcal{S}$; a *strict preorder over* $\mathcal{S}$ is an irreflexive and transitive binary relation over $\mathcal{S}$; a *partial order over* $\mathcal{S}$ is a preorder over $\mathcal{S}$ that is antisymmetric. Given a preorder $\leq$ over $\mathcal{S}$, we denote by $<$ the strict preorder induced by $\leq$, that is $s < t$ if and only if $s \leq t$ and not $t \leq s$. For any subset $\mathcal{T}$ of $\mathcal{S}$, the set of *minimal elements of* $\mathcal{T}$ *w.r.t.* $\leq$ is $\min(\mathcal{T}, \leq) = \{\, s \in \mathcal{T} \mid \neg\exists t \in \mathcal{T} : t < s \,\}$. A *preorder assignment over* $\mathcal{S}$ is any function ω that assigns a preorder $\leq^s_\omega$ over $\mathcal{S}$ to each $s \in \mathcal{S}$. A *partial order assignment over* $\mathcal{S}$ is any preorder assignment ω over $\mathcal{S}$ such that $\leq^s_\omega$ is a partial order over $\mathcal{S}$ for every $s \in \mathcal{S}$. A *total order assignment over* $\mathcal{S}$ is any preorder assignment ω over $\mathcal{S}$ such that $\leq^s_\omega$ is a total order over $\mathcal{S}$ for every $s \in \mathcal{S}$.

3 Pre-history – belief change in classical logic

An *update* is typically described as an operation that brings a knowledge base *up to date* when the *world described by it changes*, whereas a *revision* is typically described as an operation that deals with incorporating new *better* knowledge about a *world that did not change* (Keller and Winslett 1985; Winslett 1990; Katsuno and Mendelzon 1991). From a generic perspective, both forms of belief change operators – update and revision – were studied within the context of propositional logic.

Propositional belief change operators, either for update ($\diamond$) or for revision ($\circ$), take two propositional formulas, representing the original knowledge base and its update, as arguments, and return a formula representing the updated knowledge base. Any such operator $* \in \{\diamond, \circ\}$ is inductively generalised to finite sequences $\langle\phi_i\rangle_{i<n}$ of propositional formulas as follows: $*\langle\phi_0\rangle = \phi_0$ and $*\langle\phi_i\rangle_{i<n+1} = (*\langle\phi_i\rangle_{i<n}) * \phi_n$, $n > 0$. To further specify the desired properties of belief change operators, Katsuno and Mendelzon (1989, 1991) proposed two sets of postulates – one for revision and one for update. Following the original order of presentation, we will first briefly review the postulates for revision, and then those for update. Even though these postulate have been mostly absent from the literature in updates of answer-set programs, more so during the era of syntax-based updates, they took a more prominent role during the era of semantic-based updates, and will be revisited in Section 6.

We start with the following six postulates for a belief revision operator $\circ$ and formulas ϕ, ψ, μ, ν, proposed by Katsuno and Mendelzon (1989), which correspond to the AGM postulates for the case of propositional logic.

(BR1) $\phi \circ \mu \models \mu$.
(BR2) If $[\![\phi \wedge \mu]\!] \neq \emptyset$, then $\phi \circ \mu \equiv \phi \wedge \mu$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

(BR3) If $[\![\mu]\!] \neq \emptyset$, then $[\![\phi \circ \mu]\!] \neq \emptyset$.
(BR4) If $\phi \equiv \psi$ and $\mu \equiv \nu$, then $\phi \circ \mu \equiv \psi \circ \nu$.
(BR5) $(\phi \circ \mu) \wedge \nu \models \phi \circ (\mu \wedge \nu)$.
(BR6) If $[\![(\phi \circ \mu) \wedge \nu]\!] \neq \emptyset$, then $\phi \circ (\mu \wedge \nu) \models (\phi \circ \mu) \wedge \nu$.

Most of these postulates can be given a simple intuitive reading. For instance, (BR1) requires that information from the revision be retained in the revised belief base. This is also frequently referred to as the *principle of primacy of new information* (Dalal 1988). Postulate (BR2) requires that whenever the original formula and the revision are jointly consistent, the result corresponds to their conjunction. Postulate (BR3) requires that whenever the formula used for revision is satisfiable, then so should be the result of the revision. Postulate (BR4) encodes independence of syntax. Postulates (BR5) and (BR6) require that revision should be accomplished with minimal change.

The main idea behind these postulates is formally captured by the notion of a belief revision operator characterised by an order assignment.

Definition 1 (*Belief Revision Operator Characterised by an Order Assignment*)
Let $\circ$ be a belief revision operator and ω a preorder assignment over the set of all formulas. We say that $\circ$ is *characterised by* ω if for all formulae ϕ, μ,

$$[\![\phi \circ \mu]\!] = \min\left([\![\mu]\!], \leq_\omega^\phi\right).$$

A set of natural conditions on the assigned orders is captured by the following notion of a *faithful* order assignment.

Definition 2 (*Faithful Preorder Assignment Over Formulas*)
A preorder assignment ω over the set of all formulas is *faithful* if the following three conditions hold:

- If $I, J \in [\![\phi]\!]$, then $I \not<_\omega^\phi J$.
- If $I \in [\![\phi]\!]$ and $J \notin [\![\phi]\!]$, then $I <_\omega^\phi J$.
- If $[\![\phi]\!] = [\![\psi]\!]$, then $<_\omega^\phi = <_\omega^\psi$.

The representation theorem of Katsuno and Mendelzon (1989) states that operators characterised by faithful total preorder assignments over the set of all formulas are exactly those that satisfy the KM revision postulates.

Theorem 3 (*Katsuno and Mendelzon 1989*)
Let $\circ$ be a belief revision operator. Then the following conditions are equivalent:

- The operator $\circ$ satisfies conditions (BR1) – (BR6).
- The operator $\circ$ is characterised by a faithful total preorder assignment over the set of all formulas.

Whereas according to Katsuno and Mendelzon (1991), *revision* is used when we are obtaining new information about a static world, *updates* consists of bringing a knowledge base up to date when the world described by it changes. To characterise updates, Katsuno and Mendelzon (1991) proposed the following eight postulates for a belief update operator $\diamond$ and formulas ϕ, ψ, μ, ν:

(BU1) $\phi \diamond \mu \models \mu$.
(BU2) If $\phi \models \mu$, then $\phi \diamond \mu \equiv \phi$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

(BU3) If $[\![\phi]\!] \neq \emptyset$ and $[\![\mu]\!] \neq \emptyset$, then $[\![\phi \diamond \mu]\!] \neq \emptyset$.
(BU4) If $\phi \equiv \psi$ and $\mu \equiv \nu$, then $\phi \diamond \mu \equiv \psi \diamond \nu$.
(BU5) $(\phi \diamond \mu) \wedge \nu \models \phi \diamond (\mu \wedge \nu)$.
(BU6) If $\phi \diamond \mu \models \nu$ and $\phi \diamond \nu \models \mu$, then $\phi \diamond \mu \equiv \phi \diamond \nu$.
(BU7) If ϕ is complete, then $(\phi \diamond \mu) \wedge (\phi \diamond \nu) \models \phi \diamond (\mu \vee \nu)$.
(BU8) $(\phi \vee \psi) \diamond \mu \equiv (\phi \diamond \mu) \vee (\psi \diamond \mu)$.

Postulates (BU1)–(BU5) correspond to postulates (BR1)–(BR5). However, when ϕ is consistent, then (BU2) is weaker than (BR2). The property expressed by (BU8) is at the heart of belief updates: Alternative models of the original belief base ϕ or ψ are treated as possible real states of the modelled world. Each of these models is updated independently of the others to make it consistent with the update μ, obtaining a new set of interpretations – the models of the updated belief base. Based on this view of updates, Katsuno and Mendelzon (1991) proved an important representation theorem that makes it possible to constructively characterize and evaluate every operator $\diamond$ that satisfies postulates (BU1)–(BU8). The main idea, based on postulate (BU8), is formally captured by the notion of a belief update operator characterized by an order assignment.

Definition 4 (*Belief Update Operator Characterised by an Order Assignment*)
Let $\diamond$ be a belief update operator and ω a preorder assignment over $\mathfrak{I}$. We say that $\diamond$ is *characterised by* ω if for all formulae ϕ, μ,

$$[\![\phi \diamond \mu]\!] = \bigcup_{I \in [\![\phi]\!]} \min\left([\![\mu]\!], \leq_\omega^I\right).$$

A natural condition on the assigned orders is that every interpretation be the closest to itself, captured by the following notion of a *faithful* order assignment.

Definition 5 (*Faithful Order Assignment over Interpretations*)
A preorder assignment ω over $\mathfrak{I}$ is *faithful* if for every interpretation I the following condition is satisfied:

$$\text{For every } J \in \mathfrak{I} \text{ with } J \neq I \text{ it holds that } I <_\omega^I J.$$

The representation theorem of Katsuno and Mendelzon (1991) states that operators characterised by faithful order assignments over $\mathfrak{I}$ are exactly those that satisfy the KM update postulates.

Theorem 6 (*Katsuno and Mendelzon 1991*)
Let $\diamond$ be a belief update operator. Then the following conditions are equivalent:

- The operator $\diamond$ satisfies conditions (BU1)–(BU8).
- The operator $\diamond$ is characterised by a faithful preorder assignment over $\mathfrak{I}$.
- The operator $\diamond$ is characterised by a faithful partial order assignment over $\mathfrak{I}$.

Katsuno and Mendelzon's result provides a framework for belief update operators, each specified on the semantic level by strict preorders assigned to each propositional interpretation. The most influential instance of this framework is the *possible models approach* by Winslett (1988), based on minimising the set of atoms whose truth value changes when an interpretation is updated. Formally, for all interpretations I, J and K,

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

the strict preorder $<^I_{\mathsf{w}}$ is defined as follows: $J <^I_{\mathsf{w}} K$ if and only if $(J \div I) \subsetneq (K \div I)$, where $\div$ denotes set-theoretic symmetric difference. The operator $\diamond_{\mathsf{w}}$ by Winslett, unique up to equivalence of its inputs and output, thus satisfies the following equation:

$$[\![\phi \diamond_{\mathsf{w}} \mu]\!] = \bigcup_{I \in [\![\phi]\!]} \{\, J \in [\![\mu]\!] \mid \neg \exists K \in [\![\mu]\!] : (K \div I) \subsetneq (J \div I) \,\}.$$

Note that it follows from Theorem 6 that $\diamond_{\mathsf{w}}$ satisfies postulates (BU1)–(BU8).

4 The first era – model updates

The first authors to address the issue of updates and logic programs using the stable model semantics were Marek and Truszczynski (1994, 1998), although not as a means to update logic program. Instead, they used a rule-based language, and a semantics similar to the stable models semantics, to specify the updates of a database.

The underlying idea was that the rules would specify constraints that had to be satisfied by a database. Then, given an initial database and a set of rules, Marek and Truszczynski (1994, 1998) defined a semantics that assigns a set of databases that are the justified result of the update.

In a nutshell, a database DB' is considered to be a justified update of a database DB by a program U if DB', viewed as an interpretation, is a model of U, and no other database DB'', that is also a model of U, is closer to DB than DB'.

Procedurally, this notion of justified update corresponds to following the common-sense law of inertia, whereby *only* those elements that *need* to be changed due to the update specification are *actually changed*, the remaining staying the same, in line with the ideas proposed by Winslett (1988).

Representing databases as interpretations, update programs as logic programs, and with $\div$ denoting set-theoretic symmetric difference, as before, the set of interpretations resulting from updating I by $\mathcal{U}$ is given by

$$\{\, J \in [\![\mathcal{U}]\!]_{\mathsf{c}} \mid \neg \exists K \in [\![\mathcal{U}]\!]_{\mathsf{c}} : (K \div I) \subsetneq (J \div I) \,\}.$$

Example 7

Consider an initial interpretation $I = \{\, \mathsf{cold}, \mathsf{sun} \,\}$ and suppose we want to update it according to the following program $\mathcal{U}$:

$$\mathsf{rain}. \qquad \mathsf{clouds} \leftarrow \mathsf{rain}. \qquad \sim\mathsf{sun} \leftarrow \mathsf{clouds}.$$

The only justified update is the interpretation $J = \{\, \mathsf{cold}, \mathsf{clouds}, \mathsf{rain} \,\}$. Declaratively, $\mathcal{P}$ has two models $\{\, \mathsf{cold}, \mathsf{clouds}, \mathsf{rain} \,\}$ and $\{\, \mathsf{clouds}, \mathsf{rain} \,\}$, where the former is obviously closer to I than the latter. Procedurally, rain is true in J because of the fact in $\mathcal{U}$, clouds is true in J because of the second rule in $\mathcal{U}$, together with the fact that rain is now true, sun is false because of the third rule in the program, while cold is true in J, *by inertia*, because it was true in I and there is nothing in $\mathcal{U}$ forcing it to become false.

Subsequently, Przymusinski and Turner (1995, 1997) showed how the framework proposed by Marek and Truszczynski (1994, 1998) could be captured by logic programming under the stable models semantics, by encoding the initial database as a set of facts, the rules proposed by Marek and Truszczynski (1994, 1998) as rules of logic programming,

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

and by adding additional rules encoding the common-sense law of inertia, so that the stable models of the resulting logic program would correspond to the justified updates of the initial database.

It was Alferes and Pereira (1996) who first proposed to use a logic program to update another logic program, under the stable model semantics. Following the possible models approach proposed by Winslett (1988), a knowledge base DB' is considered to be the update of a knowledge base DB by U if each model of DB' is an update of a model of DB by U.

According to this approach, dubbed the *model update approach*, the problem of finding an update of a logic program $\mathcal{P}$ is reduced to the problem of individually finding updates of each of its stable models I. Each stable model would be updated following the ideas proposed by Marek and Truszczynski (1994, 1998), that is, following the common-sense law of inertia, or minimal change.

Just like Przymusinski and Turner (1995, 1997), Alferes and Pereira (1996) proposed an encoding of the problem of updating a program $\mathcal{P}$ by a program $\mathcal{U}$ into logic programming, producing another logic program, written in an extended language, whose stable models correspond to the updates of each of the stable models of $\mathcal{P}$ by $\mathcal{U}$.

According to Alferes and Pereira (1996), the update of a program $\mathcal{P}$ by a program $\mathcal{U}$ is characterized by the following set of interpretations:

$$\bigcup_{I \in [\![\mathcal{P}]\!]_{\mathsf{SM}}} \{\, J \in [\![\mathcal{U}]\!]_{\mathsf{C}} \mid \neg \exists K \in [\![\mathcal{U}]\!]_{\mathsf{C}} : (K \div I) \subsetneq (J \div I) \,\}\,.$$

As it turns out, when we take a closer look at this semantics, we soon realise that things do not behave exactly as one might expect.

Example 8
Consider the same example from the introduction, where an agent had its beliefs represented by the program $\mathcal{P}$:

$$\mathsf{goHome} \leftarrow {\sim}\mathsf{money}. \qquad \mathsf{goRestaurant} \leftarrow \mathsf{money}. \qquad \mathsf{money}.$$

which was then the subject of an update by $\mathcal{U}$ with the following two rules:

$${\sim}\mathsf{money} \leftarrow \mathsf{robbed}. \qquad \mathsf{robbed}.$$

The only stable model of $\mathcal{P}$ is $I = \{\, \mathsf{money}, \mathsf{goRestaurant} \,\}$. Program $\mathcal{U}$ has the following four models: $K_1 = \{\, \mathsf{robbed} \,\}$, $K_2 = \{\, \mathsf{robbed}, \mathsf{goRestaurant} \,\}$, $K_3 = \{\, \mathsf{robbed}, \mathsf{goHome} \,\}$, and $K_4 = \{\, \mathsf{robbed}, \mathsf{goRestaurant}, \mathsf{goHome} \,\}$. Of these four models, K_2 is the only one that is closest to I, hence it is the only one that characterizes the update of $\mathcal{P}$ by $\mathcal{U}$ according to Alferes and Pereira (1996).

Procedurally, if we update $\mathcal{P}$ by $\mathcal{U}$ following the fundamental ideas behind the possible models approach and the common sense law of inertia, the result must be characterized by the stable models of $\mathcal{P}$ after they are minimally changed to become consistent with $\mathcal{U}$. And in order to make I consistent with the rules in $\mathcal{U}$, one needs to modify the truth value of two atoms, robbed and money, arriving at the interpretation $K_2 = \{\, \mathsf{robbed}, \mathsf{goRestaurant} \,\}$. So, after the update, the agent has no money but still plans to go to a restaurant, different from the intuitively correct result where, after being robbed, the agent would not go to the restaurant, and instead go home, because he has no money.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The undesirable behaviour illustrated by the previous example was first observed by Leite and Pereira (1998), leading to the beginning of the second era.

5 The second era – syntax-based updates

The reason for the problem illustrated by the example at the end of the previous section is that modifications on the level of individual stable models, akin to model-based belief update operators, are unable to capture the essential relationships between literals encoded in rules. This was first argued by Leite and Pereira (1998), who took a closer look at Newton's first law, also known as the *law of inertia*, which states that *"every body remains at rest or moves with constant velocity in a straight line, unless it is compelled to change that state by an unbalanced force acting upon it"* (Newtono 1726).[1] In their discussion, Leite and Pereira pointed out that the common-sense interpretation of this law as *"things keep as they are unless some kind of force is applied to them"* is true, but does not exhaust its meaning. It is the result of all applied forces that governs the outcome. Take a body to which several forces are applied, and which is in a state of equilibrium due to those forces cancelling out. Later, one of those forces is removed and the body starts to move. The same kind of behaviour presents itself when updating programs. Before obtaining the truth value, by inertia, of those elements not directly affected by the update program, one should verify whether the truth of such elements is not indirectly affected by the updating of other elements or, in other words, whether there is still some rule that supports such truth.

To rectify this problem, a number of approaches were proposed. Despite being based on fundamentally different principles and methods when compared to their model update counterparts, they all take into account the syntactic rule-based form of the programs involved.

This section provides an overview of existing rule update semantics, pointing at some of the technical as well as semantic differences between them, often relying on examples to show how these semantics are interrelated.

Rule update semantics typically deal only with ground non-disjunctive rules and some do not allow for default negation in their heads. While some of them follow the belief update tradition and construct an updated program given the original program and its update, others only assign a set of stable models to a pair or sequence of programs where each represents an update of the preceding ones. In order to compare these semantics, we adopt the latter, less restrictive point of view. The "input" of a rule update semantics is thus defined as follows:

Definition 9 (*Dynamic Logic Program*)
A *dynamic logic program* (DLP) is a finite sequence of ground non-disjunctive logic programs. Given a DLP $\boldsymbol{P}$, we denote by $\mathsf{all}(\boldsymbol{P})$ the set of all rules belonging to the programs in $\boldsymbol{P}$. We say that $\boldsymbol{P}$ is *acyclic* if $\mathsf{all}(\boldsymbol{P})$ is acyclic.

In order to avoid issues with rules that are repeated in multiple components of a DLP, we assume throughout this section that every rule is uniquely identified in all

[1] The original text of Newton is as follows: *"Corpus omne perseverare in statu suo quiescendi vel movendi uniformiter in directum, nisi quatenus illud a viribus impressis cogitur statum suum mutare."*.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

set-theoretic operations. This could be formalised by assigning a unique name to each rule and performing operations on names instead of on the rules themselves.

The set of stable models assigned to dynamic logic programs under a particular update semantics will be denoted as follows:

Definition 10 (*Rule Update Semantics*)
A *rule update semantics* S is characterised by a (partial) function $[\![\cdot]\!]_{\mathsf{S}}$ that assigns a set $[\![\boldsymbol{P}]\!]_{\mathsf{S}}$ of interpretations to a dynamic logic program $\boldsymbol{P}$. We call each member of $[\![\boldsymbol{P}]\!]_{\mathsf{S}}$ an S*-model of* $\boldsymbol{P}$.

Whenever an approach is defined through a rule update operator, such an operator is understood as a function that assigns a program to each pair of programs. A rule update operator $\oplus$ is extended to DLPs as follows: $\bigoplus\langle\mathcal{P}_0\rangle = \mathcal{P}_0$; $\bigoplus\langle\mathcal{P}_i\rangle_{i<n+1} = (\bigoplus\langle\mathcal{P}_i\rangle_{i<n}) \oplus \mathcal{P}_n$, $n > 0$. Note that such an operator naturally induces a rule update semantics $\mathsf{S}_{\oplus}$: given a DLP $\boldsymbol{P}$, the $\mathsf{S}_{\oplus}$-models of $\boldsymbol{P}$ are the stable models of $\bigoplus \boldsymbol{P}$. In the rest of this paper, we exercise a slight abuse of notation by referring to the operators and their associated update semantics interchangeably.

We first discuss a major group of semantics based on the *causal rejection principle* (Leite and Pereira 1998; Buccafurri *et al*. 1999; Alferes *et al*. 2000; Eiter *et al*. 2002; Alferes *et al*. 2005; Osorio and Cuevas 2007), followed by semantics based on *preferences* (Zhang 2006; Delgrande *et al*. 2007). We then proceed to discuss semantics that bear some characteristics of revision rather than update (Sakama and Inoue 2003; Osorio and Zepeda 2007; Delgrande 2010) and touch upon approaches that manipulate dependencies on default assumptions induced by rules (Šefránek 2011; Krümpelmann and Kern-Isberner 2010; Krümpelmann 2012). Towards the end of this section, we formulate some fundamental properties of rule update semantics.

The use and effect of integrity constraints within the semantics presented in this section has not received significant attention in the literature. Whereas most semantics are only defined for programs without integrity constraints, often pointing to the fact that one can replace the integrity constraint $\leftarrow B(\pi).$ with the rule $a_\pi \leftarrow {\sim}a_\pi, B(\pi).$, where a_π is a new atom, with the same effect, other semantics are defined for programs with integrity constraints, but without any specific provisions regarding their role. Since all these semantics that were defined for programs with integrity constraints are preserved under the above transformation that eliminates them, in this section we will restrict the definition of semantics to only consider DLPs without integrity constraints.

5.1 Causal rejection-based semantics

The *causal rejection principle* (Leite and Pereira 1998) forms the basis of a number of rule update semantics. Informally, it can be stated as follows:

> A rule should be *rejected* when it is directly contradicted by a more recent rule.

The common understanding of this principle has been to consider a direct contradiction between rules to mean a conflict between the heads of rules, that is, that the head of the rejecting rule is the negation of the head of the rejected one.

In the first proposals based on the causal rejection principle, only conflicts between *objective* literals in rule heads were considered, and default negation in rule heads was

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

not allowed (Leite and Pereira 1998; Eiter *et al.* 2002). Later, it was found that this approach has certain limitations, namely that some belief states, represented by stable models, become unreachable (Alferes *et al.* 2000; Leite 2003). For example, no update of the program $\mathcal{P} = \{\, \mathsf{p}. \,\}$ leads to a stable model where neither p nor $\neg\mathsf{p}$ is true. Default negation in rule heads was thus used to regain reachability of such states. For instance, the update $\mathcal{U} = \{\, \sim\mathsf{p}., \sim\neg\mathsf{p}. \,\}$ forces p to be unknown, regardless of its previous state. Hence, strong negation is used to express that an atom *becomes explicitly false*, while default negation allows for more fine-grained control: the atom only *ceases to be true*, but its truth value may not be unknown. The latter also makes it possible to move between any pair of epistemic states by means of updates, as illustrated by the following example from the book by Leite (2003):

Example 11 (*Railway crossing*)
Suppose that we use the following logic program to choose an action at a railway crossing:

$$\mathsf{cross} \leftarrow \neg\mathsf{train}. \qquad \mathsf{wait} \leftarrow \mathsf{train}. \qquad \mathsf{listen} \leftarrow \sim\mathsf{train}, \sim\neg\mathsf{train}.$$

The intuitive meaning of these rules is as follows: one should cross if there is evidence that no train is approaching; wait if there is evidence that a train is approaching; listen if there is no such evidence.

Consider a situation where a train is approaching, represented by the fact (train.). After this train has passed by, we want to update our knowledge to an epistemic state where we lack evidence with regard to the approach of a train. If this was accomplished by updating with the fact ($\neg$train.), we would cross the tracks at the subsequent state, risking being killed by another train that was approaching. Therefore, we need to express an update stating that all past evidence for an atom is to be removed. The proposal was to accomplish this by allowing default negation in heads of rules. In this scenario, the intended update could be expressed by the fact ($\sim$train.).

In the following, we thus present the semantics from Leite and Pereira (1998) and Eiter *et al.* (2002) in generalised forms that allow default negation in rule heads, but coincide with their original definitions on programs without such feature (Leite 2003).

The notion of conflicting rules plays an important role in all the semantics based on the causal rejection principle. When generalised logic programs are used, a conflict between rules occurs when the head literal l of one rule is the default or strong negation of the head literal of the other rule, $\sim l$ or $\bar{l}$, respectively. Following the proposal of Leite (2003), we consider the conflicts between a rule with an objective literal l in its head and a rule with the default negation of the same literal $\sim l$ in its head as primary conflicts, while conflicts between rules with complementary objective literals in their heads, l and $\bar{l}$, are handled by *expanding* DLPs. Expansion of a DLP corresponds to the following operation: whenever a DLP contains a rule with an objective literal l in its head, its expansion also contains a rule with the same body and the literal $\sim\bar{l}$ in its head. Formally:

Definition 12 (*Expanded Version of a DLP*)
Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP. The *expanded version of* $\boldsymbol{P}$ is the DLP $\boldsymbol{P}^{\mathsf{e}} = \langle \mathcal{P}_i^{\mathsf{e}} \rangle_{i<n}$ where for every $i < n$,

$$\mathcal{P}_i^{\mathsf{e}} = \mathcal{P}_i \cup \{ \sim\bar{l} \leftarrow B(\pi). | \pi \in \mathcal{P}_i \wedge H(\pi) = \{\, l \,\} \wedge l \in \mathcal{L} \}.$$

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The additional rules in the expanded version capture the coherence principle: when an objective literal l is derived, its complement $\bar{l}$ cannot be concurrently true and thus $\sim\bar{l}$ must be true. In this way, every conflict between complementary objective literals directly translates into a conflict between an objective literal and its default negation. By ensuring that we always use expanded versions of DLPs, we can adopt the following definition of a conflict between a pair of rules.

Definition 13 (*Conflicting rules*)
We say that rules π, σ *are in conflict*, denoted by $\pi \bowtie \sigma$, if and only if

$$H(\pi) \neq \emptyset \quad \text{and} \quad H(\pi) = \sim H(\sigma).$$

5.1.1 The JU-semantics and the AS-semantics

The historically first rule update semantics in answer-set programming is the *justified update semantics*, or *JU-semantics* for short (Leite and Pereira 1998), with the idea to define a set of *rejected rules*, which depends on a stable model candidate, and then verify that the candidate is indeed a stable model of the remaining rules.

Definition 14 (*JU-Semantics Leite and Pereira 1998*)
Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. We define the set $\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J)$ of rejected rules as

$$\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J) = \{\pi \in \mathcal{P}_i | \exists j\, \exists \sigma : i < j < n \wedge \sigma \in \mathcal{P}_j \wedge \pi \bowtie \sigma \wedge J \models B(\sigma)\}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{JU}}$ of *JU-models of a DLP* $\boldsymbol{P}$ consists of all stable models J of the program

$$\mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}^{\mathsf{e}}, J).$$

Under the JU-semantics, a rule π is rejected if and only if a more recent rule σ is in conflict with π and the body of σ is satisfied in the stable model candidate J. Note that the latter condition is essential – without it, rules might get rejected simply because a more recent rule σ has a conflicting head, without a guarantee that σ will actually be activated.

Before we illustrate the JU-semantics with an example, we first present a related semantics, which prevents rejected rules from rejecting other rules. It is dubbed the *update answer-set semantics*, or *AS-semantics* for short (Eiter *et al.* 2002):

Definition 15 (*AS-Semantics Eiter et al. 2002*)
Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. We define the set of rejected rules $\mathsf{rej}_{\mathsf{AS}}(\boldsymbol{P}, J)$ as[2]

$$\mathsf{rej}_{\mathsf{AS}}(\boldsymbol{P}, J) = \{\pi \in \mathcal{P}_i | \exists j\, \exists \sigma : i < j < n \wedge \sigma \in \mathcal{P}_j \setminus \mathsf{rej}_{\mathsf{AS}}(\boldsymbol{P}, J) \wedge \pi \bowtie \sigma \wedge J \models B(\sigma)\}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{AS}}$ of *AS-models of a DLP* $\boldsymbol{P}$ consists of all stable models J of the program

$$\mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_{\mathsf{AS}}(\boldsymbol{P}^{\mathsf{e}}, J).$$

[2] Note that although the definition is recursive, the defined set is unique. This is because we assume that every rule is uniquely identified and to determine whether a rule from $\mathcal{P}_i$ is rejected, the recursion only refers to rejected rules from programs $\mathcal{P}_j$ with j strictly greater than i. One can thus first find the rejected rules in $\mathcal{P}_{n-1}$ (always $\emptyset$ by the definition), then those in $\mathcal{P}_{n-2}$ and so on until $\mathcal{P}_0$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The definitions of the JU- and AS-semantics are fairly straightforward and reflect intuitions about rule updates better than an approach based on a belief update construction, such as the one by Alferes and Pereira (1996). As an illustration, let us look at the result of these semantics when applied to the example in the introduction:

Example 16
Consider again the program $\mathcal{P}$ from the example in the introduction which contains the rules

$$\mathsf{goHome} \leftarrow \sim\mathsf{money}. \qquad \mathsf{goRestaurant} \leftarrow \mathsf{money}. \qquad \mathsf{money}.$$

and its update $\mathcal{U}$ with the rules

$$\sim\mathsf{money} \leftarrow \mathsf{robbed}. \qquad \mathsf{robbed}.$$

Following the discussion in the introduction, the expected stable model of the DLP $\langle\mathcal{P},\mathcal{U}\rangle$ is $J = \{\,\mathsf{robbed}, \mathsf{goHome}\,\}$. Also, $\mathsf{rej}_{\mathsf{JU}}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}}, J) = \mathsf{rej}_{\mathsf{AS}}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}}, J) = \{\,\mathsf{money}.\,\}$, and J is indeed a stable model of the remaining rules in $\mathsf{all}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}})$. Furthermore, J is the only interpretation with these properties, so

$$[\![\langle\mathcal{P},\mathcal{U}\rangle]\!]_{\mathsf{JU}} = [\![\langle\mathcal{P},\mathcal{U}\rangle]\!]_{\mathsf{AS}} = \{\,J\,\}.$$

Example 17
To illustrate the expansion mechanism and its interplay with the rejection mechanism, consider again the program $\mathcal{P}$ from the example in the introduction which contains the rules

$$\mathsf{goHome} \leftarrow \sim\mathsf{money}. \qquad \mathsf{goRestaurant} \leftarrow \mathsf{money}. \qquad \mathsf{money}.$$

but now its update $\mathcal{U}$ is modified by replacing $\sim\mathsf{money}$ with $\neg\mathsf{money}$:

$$\neg\mathsf{money} \leftarrow \mathsf{robbed}. \qquad \mathsf{robbed}.$$

The expanded version of both programs are:

$$\begin{array}{lccc}
\mathcal{P}^{\mathsf{e}} & \mathsf{goHome} \leftarrow \sim\mathsf{money}. & \mathsf{goRestaurant} \leftarrow \mathsf{money}. & \mathsf{money}. \\
 & \sim\neg\mathsf{goHome} \leftarrow \sim\mathsf{money}. & \sim\neg\mathsf{goRestaurant} \leftarrow \mathsf{money}. & \sim\neg\mathsf{money}. \\
\mathcal{U}^{\mathsf{e}} & \neg\mathsf{money} \leftarrow \mathsf{robbed}. & \mathsf{robbed}. & \\
 & \sim\mathsf{money} \leftarrow \mathsf{robbed}. & \sim\neg\mathsf{robbed}. &
\end{array}$$

The expected stable model of the DLP $\langle\mathcal{P},\mathcal{U}\rangle$ is $J = \{\,\mathsf{robbed}, \neg\mathsf{money}, \mathsf{goHome}\,\}$. Also, $\mathsf{rej}_{\mathsf{JU}}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}}, J) = \mathsf{rej}_{\mathsf{AS}}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}}, J) = \{\,\mathsf{money}., \sim\neg\mathsf{money}.\,\}$, and J is indeed a stable model of the remaining rules in $\mathsf{all}(\langle\mathcal{P},\mathcal{U}\rangle^{\mathsf{e}})$. Furthermore, J is the only interpretation with these properties, so

$$[\![\langle\mathcal{P},\mathcal{U}\rangle]\!]_{\mathsf{JU}} = [\![\langle\mathcal{P},\mathcal{U}\rangle]\!]_{\mathsf{AS}} = \{\,J\,\}.$$

Nevertheless, problematic examples which are not handled correctly by these semantics have also been identified (Leite 2003). Many of them involve tautological updates, the intuition being that a tautological rule (i.e., a rule whose head literal also belongs to its body) cannot indicate a change in the modelled world because it is always true. It thus follows that a tautological update should not affect the stable models of the original program. Interestingly, immunity to tautological updates is a desirable property

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

of belief updates in classical logic, being a direct consequence of postulate (BU2). The following example illustrates a misbehaviour of the AS-semantics

Example 18
Consider the DLP $\boldsymbol{P}_1 = \langle \{ \mathsf{p}. \}, \{ \neg \mathsf{p}. \}, \{ \mathsf{p} \leftarrow \mathsf{p}. \} \rangle$. Under the AS-semantics, we obtain $[\![\boldsymbol{P}_1]\!]_{\mathsf{AS}} = \{ \{ \neg \mathsf{p} \}, \{ \mathsf{p} \} \}$, where the expected result is $\{ \{ \neg \mathsf{p} \} \}$.

In the previous example, the JU-semantics provides an adequate solution: since it allows rejected rules to reject, the initial rule is always rejected and $[\![\boldsymbol{P}_1]\!]_{\mathsf{JU}} = \{ \{ \neg \mathsf{p} \} \}$ as expected. Unfortunately, there are also numerous DLPs to which the JU-semantics assigns unwanted models, as illustrated b the following example.

Example 19
Consider the DLP $\boldsymbol{P}_2 = \langle \{ \mathsf{p}. \}, \{ \sim \mathsf{p} \leftarrow \sim \mathsf{p}. \} \rangle$. Under the JU- and AS-semantics, we obtain $[\![\boldsymbol{P}_2]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}_2]\!]_{\mathsf{AS}} = \{ \emptyset, \{ \mathsf{p} \} \}$, where the expected result is $\{ \{ \mathsf{p} \} \}$.

The unwanted stable model $\emptyset$ arises because rejecting p. causes the default assumption $\sim$p to be "reinstated", that is, p to be assumed false *by default* – despite p being initially asserted as a fact.

5.1.2 The DS-semantics

The problem illustrated with Example 19 is addressed in the *dynamic stable model semantics*, or *DS-semantics* for short (Alferes *et al.* 2000), by constraining the set of atoms that can be assumed false by default.

Definition 20 (*DS-Semantics Alferes et al. 2000*)
Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. The set of rejected rules $\mathsf{rej}_{\mathsf{DS}}(\boldsymbol{P}, J)$ is identical to the set $\mathsf{rej}_{\mathsf{JU}}(\boldsymbol{P}, J)$ and we define the set of default assumptions $\mathsf{def}(\boldsymbol{P}, J)$ as

$$\mathsf{def}(\boldsymbol{P}, J) = \{ \sim l | l \in \mathcal{L} \wedge \neg \exists \pi \in \mathsf{all}(\boldsymbol{P}) : H(\pi) = \{ l \} \wedge J \models B(\pi) \}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{DS}}$ of *DS-models of a DLP* $\boldsymbol{P}$ consists of all interpretations J such that

$$J' = \mathsf{least}([\mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_{\mathsf{DS}}(\boldsymbol{P}^{\mathsf{e}}, J)] \cup \mathsf{def}(\boldsymbol{P}^{\mathsf{e}}, J)),$$

where $J' = J \cup \sim(\mathcal{L} \setminus J)$ and $\mathsf{least}(\cdot)$ denotes the least model of the argument program with all literals treated as atoms.

Note that it follows from the definition of a (regular) stable model that J is a JU-stable model of a DLP $\boldsymbol{P}$ if and only if

$$J' = \mathsf{least}([\mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_{\mathsf{DS}}(\boldsymbol{P}^{\mathsf{e}}, J)] \cup \sim(\mathcal{L} \setminus J)).$$

Hence, the difference between the JU- and DS-semantics is only in the set of default assumptions that can be adopted to construct the model. In particular, if a rule that derives an objective literal l is present in $\mathsf{all}(\boldsymbol{P})$, then $\sim l$ is not among the default assumptions in the DS-semantics although it could be used as a default assumption in the JU-semantics. In other words, according to the DS-semantics, once some objective literal l can be derived by some (*older*) rule, we can no longer assume $\sim l$ by default. The DS-semantics thus resolves problems with examples such as the previous one encoded by $\boldsymbol{P}_2$, that is, it

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

holds that $[\![\boldsymbol{P}_2]\!]_{\mathsf{DS}} = \{\{\mathsf{p}\}\}$. But even the DS-semantics exhibits problematic behaviour when tautological updates are involved, as illustrated by the following example.

Example 21
Consider the DLP $\boldsymbol{P}_3 = \langle\{\mathsf{p}., \neg\mathsf{p}.\}, \{\mathsf{p} \leftarrow \mathsf{p}.\}\rangle$. Under the DS-semantics, just as the case with the JU- and the AS-semantics, we obtain $[\![\boldsymbol{P}_3]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}_3]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}_3]\!]_{\mathsf{DS}} = \{\{\mathsf{p}\}\}$, where the expected result is $\{\}$.

The expected result here is that no stable model should be assigned to $\boldsymbol{P}_3$ because initially it has none and the tautological update should not change anything about that situation. Whereas one might wonder why not simply achieve immunity to tautologies by preprocessing programs and removing them, it is important to note that the problem runs deeper. Immunity to tautologies is a simple, easy to understand manifestation of a deeper problem concerning updates that should be considered tautological, even though there are no tautological rules. These tautological updates are characterized by the existence of cycles involving more than one rule, which could not be dealt with by simply removing tautologies, not even sets of cyclic rules within a single program.

5.1.3 The RD-semantics

The trouble with tautological and some other types of irrelevant updates has been discussed and finally resolved by Alferes *et al.* (2005) who defined the so-called *refined extension principle* – a principle encoding the desirable immunity to tautological, cyclic and other irrelevant updates – as well as a rule update semantics satisfying the principle. The definition of this semantics is very similar to the DS-semantics, the only difference being that in the set of rejected rules, $i \leq j$ is required instead of $i < j$, which seems to be a technical trick with little correspondence to any intuition. The semantics is called the *refined dynamic stable model semantics*, or *RD-semantics* for short:

Definition 22 (*RD-Semantics Alferes et al. 2005*)
Let $\boldsymbol{P} = \langle\mathcal{P}_i\rangle_{i<n}$ be a DLP and J an interpretation. We define the set of rejected rules $\mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}, J)$ as

$$\mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}, J) = \{\pi \in \mathcal{P}_i | \exists j\, \exists \sigma : i \leq j < n \wedge \sigma \in \mathcal{P}_j \wedge \pi \bowtie \sigma \wedge J \models B(\sigma)\}.$$

The set $[\![\boldsymbol{P}]\!]_{\mathsf{RD}}$ of *RD-models of a DLP* $\boldsymbol{P}$ consists of all interpretations J such that

$$J' = \mathsf{least}([\mathsf{all}(\boldsymbol{P}^{\mathbf{e}}) \setminus \mathsf{rej}_{\mathsf{RD}}(\boldsymbol{P}^{\mathbf{e}}, J)] \cup \mathsf{def}(\boldsymbol{P}^{\mathbf{e}}, J)),$$

where J' and $\mathsf{least}(\cdot)$ are as before.

Due to satisfying the refined extension principle, the RD-semantics is completely immune to tautological updates. For instance, in case of the previous example encoded by $\boldsymbol{P}_3$ we obtain $[\![\boldsymbol{P}_3]\!]_{\mathsf{RD}} = \emptyset$. As we shall see, a vast majority of rule update semantics, even those that have been developed much later and are not based on causal rejection, are not immune to tautological updates.

Banti *et al.* (2005) present an alternative, equivalent, characterisation for the RD-semantics, based on level mappings, which is perhaps better in helping understand the difference w.r.t. previous semantics than the trick used in the previous definition of the set of rejected rules where $i \leq j$ is required instead of $i < j$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Theorem 23 (RD-Semantics, Alternative Characterisation Banti et al. 2005)
Let $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ be a DLP and J an interpretation. Given a level mapping ℓ, let the set $\mathsf{rej}_\ell(\boldsymbol{P}, J)$ of rejected rules be defined as follows:

$$\mathsf{rej}_\ell(\boldsymbol{P}, J) = \{\pi \in \mathcal{P}_i | \exists j\, \exists \sigma : i < j < n \wedge \sigma \in \mathcal{P}_j \wedge \pi \bowtie \sigma \wedge J \models B(\sigma) \wedge \ell(H(\sigma)) > \ell(B(\sigma))\}.$$

Then, $J \in [\![\boldsymbol{P}]\!]_{\mathsf{RD}}$ iff there exists a level mapping ℓ such that:

1. J is a C-model of $\mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_\ell(\boldsymbol{P}^{\mathsf{e}}, J)$, and
2. $\forall \mathsf{p} \in J, \exists \sigma \in \mathsf{all}(\boldsymbol{P}^{\mathsf{e}}) \setminus \mathsf{rej}_\ell(\boldsymbol{P}^{\mathsf{e}}, J)$ such that $H(\sigma) = \mathsf{p} \wedge, \ell(H(\sigma)) > \ell(B(\sigma)) \wedge J \models B(\sigma)$.

This characterisation borrows from the work of Hitzler and Wendt (2005) on uniform characterisations of different semantics for logic programs in terms of level mappings. In particular, this characterisation is based on the notion of well-supported models (Fages 1994), an alternative view of stable models that characterises them as C-models with the additional requirement that there exists some level mapping such all atoms in the model are supported by some rule whose head is that atom and the level of the that atom is greater than the level of the rule's body. When extended to the case of DLPs, besides being used to decide whether some objective literal should be true, rules are also used to reject other rules. Hence, the concept of well-supportedness is also adopted to the rejection mechanism, and rules can only reject other rules with the additional constraint that the level of their heads be greater than the level of their bodies.

5.1.4 Relationship between semantics based on causal rejection

The rule update semantics introduced above are strongly related to one another. The above considerations show that undesired stable models of the AS-, JU- and DS-semantics were eliminated by enlarging the set of rejected rules or by shrinking the set of default assumptions. The following theorem shows that no additional stable models were added in the process:

Theorem 24 (Leite 2003; Alferes et al. 2005)
Let $\boldsymbol{P}$ be a DLP. Then,

$$[\![\boldsymbol{P}]\!]_{\mathsf{AS}} \supseteq [\![\boldsymbol{P}]\!]_{\mathsf{JU}} \supseteq [\![\boldsymbol{P}]\!]_{\mathsf{DS}} \supseteq [\![\boldsymbol{P}]\!]_{\mathsf{RD}}.$$

Moreover, for each inclusion above there exists a DLP for which the inclusion is strict.

Furthermore, all of these semantics coincide when only acyclic DLPs are considered, showing that the differences in the definitions of rejected rules and default assumptions are only relevant in the presence of cyclic dependencies between literals.

Theorem 25 (Homola 2004)
Let $\boldsymbol{P}$ be an acyclic DLP. Then,

$$[\![\boldsymbol{P}]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}]\!]_{\mathsf{DS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}}.$$

The relation between these semantics and other formalisms has also been studied. It has been shown by Eiter *et al.* (2002) that the AS-semantics coincides with the non-disjunctive case of the semantics for inheritance programs by Buccafurri *et al.* (1999).

One of the open issues with these semantics is that one cannot easily *condense* a DLP to a single logic program that could be used *instead* of the DLP to perform further updates. The first obstacle is that the stable models of a DLP may be non-minimal, as illustrated by the following example.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Example 26
Consider the DLP $\boldsymbol{P}_4 = \langle \{ \mathsf{p}., \mathsf{q} \leftarrow \mathsf{p}. \}, \{ \sim\mathsf{p} \leftarrow \sim\mathsf{q}. \} \rangle$. According to all semantics introduced so far, we obtain $[\![\boldsymbol{P}_4]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}_4]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}_4]\!]_{\mathsf{DS}} = [\![\boldsymbol{P}_4]\!]_{\mathsf{RD}} = \{ \emptyset, \{ \mathsf{p}, \mathsf{q} \} \}$.

Since stable models of non-disjunctive programs are subset-minimal, no such program can have the set of stable models $[\![\boldsymbol{P}_4]\!]_{\mathsf{RD}}$. Condensing to a disjunctive program is also problematic because rule update semantics are constrained to non-disjunctive programs only, so after a condensation one would not be able to perform any further updates.

Nevertheless, the original definition of the four discussed S-semantics (JU-, AS-, DS- and RD-semantics) was accompanied by a translation of a DLP to a single non-disjunctive program *over an extended language* whose stable models correspond one-to-one to the S-models assigned to the DLP under the respective rule update semantics. Due to the language extension, the new program cannot simply be updated directly as a substitute for the original DLP, but may serve as a way to study the computational properties of the rule update semantics and as a way to implement it using existing answer-set solvers.

In the literature, we can also find the semantics proposed by Osorio and Cuevas (2007) which can be seen as simpler substitutes for the AS-semantics. Unlike the semantics discussed above, they are not defined declaratively, instead they are specified directly by translating the initial program and its update to a single program *over the same language*. The first translation essentially weakens the rules from the original program by making them defeasible, that is, a rule $l \leftarrow B(\pi)$ is transformed into the rule $l \leftarrow B(\pi), \sim\bar{l}$. The authors have shown that the resulting semantics is equivalent to the AS-semantics if a single update is performed and the updating program contains a tautology $l \leftarrow l$ for every objective literal l. Due to its simplicity, because it only deals with a single update in a way that cannot immediately be extended to additional updates, this semantics is *not* sensitive to the addition and removal of tautologies, but it adopts the problematic behaviour of the AS-semantics even when the tautologies are *removed* from the updating program. For example, when considering the DLPs

$$\boldsymbol{P}_5 = \langle \{ \mathsf{p}., \neg\mathsf{p}. \}, \emptyset \rangle \text{ and}$$
$$\boldsymbol{P}'_5 = \langle \{ \mathsf{p}., \neg\mathsf{p}. \}, \{ \mathsf{p} \leftarrow \mathsf{p}., \neg\mathsf{p} \leftarrow \neg\mathsf{p}. \} \rangle$$

the AS-semantics correctly assigns no AS-model to $\boldsymbol{P}_5$ although its sensitivity to tautological updates causes $\boldsymbol{P}'_5$ to have two AS-models: $\{ \mathsf{p} \}$ and $\{ \neg\mathsf{p} \}$. The first semantics suggested by Osorio and Cuevas (2007) assigns these two interpretations, $\{ \mathsf{p} \}$ and $\{ \neg\mathsf{p} \}$, to both $\boldsymbol{P}_5$ and $\boldsymbol{P}'_5$, so it exhibits problematic behaviour even on DLPs that were correctly handled by the AS-semantics.

The second translation is more involved as it produces a program that may not be expressible by a non-disjunctive program. The resulting update semantics is shown to coincide with the AS-semantics in case only a single update is performed. Note that since the AS-semantics coincides with the JU-semantics on DLPs of length two,[3] the

[3] The difference between the AS-semantics and JU-semantics is that according to JU you can have rejected rules rejecting other rules, while according to AS rejected rules cannot reject other rules. According to both semantics, rules of the first program of a DLP cannot reject and rules of the last program of a DLP cannot be rejected. It follows that in a DLP of length two, according to the JU-semantics, there cannot be a rule simultaneously rejecting and being rejected, so it follows that AS and JU coincide for DLPs of length two.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

above mentioned relationships between semantics by Osorio and Cuevas (2007) and the AS-semantics also hold for the JU-semantics. Their behaviour on DLPs of length three or more has not been studied.

5.2 Preference-based semantics

A smaller group of rule update semantics relies on syntactic transformations and semantics for *prioritised logic programs*. These semantics do not consider default negation in heads of rules.

Formally, a prioritised logic program is a pair $(\mathcal{P}, \prec)$ where $\mathcal{P}$ is a program and $\prec$ is a strict partial order over $\mathcal{P}$. The intuitive meaning of $\prec$ is that if $\pi \prec \sigma$, then σ is more preferred than π. There exist a number of different semantics for prioritised logic programs (Brewka and Eiter 1999; Delgrande *et al.* 2003; Schaub and Wang 2003; Zhang 2003). Their goal is to plausibly use the preference relation $\prec$ to choose the preferred stable models among the stable models of $\mathcal{P}$, or to constrain the rules of $\mathcal{P}$ used to determine the stable models.

5.2.1 The PRZ-semantics

One rule update semantics of this type was defined by Zhang (2006) and relies on the semantics for prioritised logic programs proposed by Zhang (2003).[4] Generally speaking, according to Zhang (2003), the semantics is assigned to a prioritised logic program $(\mathcal{P}, \prec)$ by pruning away less preferred rules, obtaining an ordinary logic program $\mathcal{P}^{\prec} \subseteq \mathcal{P}$ called a *reduct*. Formally, $\mathcal{P}^{\prec}$ is a reduct of $(\mathcal{P}, \prec)$ if there exists a sequence of sets $\mathcal{P}_i$ $(i = 0, 1, \ldots)$ such that

$$\mathcal{P}_0 = \mathcal{P}, \qquad \mathcal{P}_{i+1} = \mathcal{P}_i - \mathcal{R}_i, \qquad \mathcal{P}^{\prec} = \bigcap_{i=0}^{\infty} \mathcal{P}_i,$$

where

$$\mathcal{R}_i = \{\pi \in \mathcal{P}_i \mid \exists \pi' \in \mathcal{P}_i, \text{ such that } \forall \pi'' \in \mathcal{R}_i, \pi'' < \pi' \wedge \pi'' \lhd (\mathcal{P}_i - \mathcal{R}_i) \text{ and}$$
$$\nexists \mathcal{R}'_i \subseteq \mathcal{P}_i \text{ such that } \exists \pi'' \in \mathcal{R}_i, \forall \pi' \in \mathcal{R}'_i, \pi' < \pi'' \wedge \pi' \lhd (\mathcal{P}_i - \mathcal{R}'_i)\},$$

and $\pi \lhd \mathcal{Q}$ denotes that rule π is defeated by program $\mathcal{Q}$, which is true if there exists some objective literal $l \in I \in [\![\mathcal{Q}]\!]_{\mathsf{SM}}$ such that $l \in B(\pi)^-$.

A prioritised logic program may have zero or more reducts and the preferred stable models are all the stable models of all the reducts. For a detailed discussion of reducts and their properties the reader can refer to the paper by Zhang (2003).

Subsequently, the update semantics defined by Zhang (2006) performs an update of a program $\mathcal{P}$ by a program $\mathcal{U}$ by executing the following steps:

1. Take some stable model $J_{\mathcal{P}}$ of $\mathcal{P}$.
2. Determine the set $\mathit{Update}(J_{\mathcal{P}}, \mathcal{U})$ of interpretations resulting from updating $J_{\mathcal{P}}$ by $\mathcal{U}$, in a way similar to to the approach of Marek and Truszczynski (1994, 1998), given by:

[4] Note that the preference relation in these papers is reversed w.r.t. the one we use here, that is, $\pi \prec \sigma$ means in the sense of Zhang (2003, 2006) that π is more preferred than σ.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

$$Update(J_{\mathcal{P}},\mathcal{U}) = \{\, I \in [\![\mathcal{U}]\!]_{\mathsf{C}} \mid \neg\exists J \in [\![\mathcal{U}]\!]_{\mathsf{C}} : (J \div J_{\mathcal{P}}) \subsetneq (I \div J_{\mathcal{P}}) \,\},$$

where $\div$ denotes set-theoretic symmetric difference. Choose any interpretation from $Update(J_{\mathcal{P}},\mathcal{U})$ and denote it by $J_{\langle\mathcal{P},\mathcal{U}\rangle}$.

3. Extract a maximal subset $\mathcal{P}'$ of $\mathcal{P}$ that is coherent with $J_{\langle\mathcal{P},\mathcal{U}\rangle}$, that is, such that there exists a stable model of $\mathcal{P}' \cup \{\, l. \mid l \in J_{\langle\mathcal{P},\mathcal{U}\rangle} \,\}$.
4. The set $(\mathcal{P}' \cup \mathcal{U}, \mathcal{P}' \times \mathcal{U})^{<}$ of reducts of the prioritised logic program $(\mathcal{P}' \cup \mathcal{U}, \mathcal{P}' \times \mathcal{U})$ is the result of updating $\mathcal{P}$ by $\mathcal{U}$.

As explained by Zhang (2006), the intuition behind the first two steps is that simply taking a maximal subset of $\mathcal{P}$ coherent with $\mathcal{U}$ is too crude an operation because it does not take into account the source of a conflict.

Example 27 (*Intuition For Steps 1. and 2. Zhang 2006*)
Consider the programs

$$\mathcal{P}: \quad \mathsf{p}. \qquad \mathsf{q} \leftarrow \mathsf{r}. \qquad \text{and} \qquad \mathcal{U}: \quad \mathsf{r} \leftarrow \mathsf{p}. \qquad \neg\mathsf{q} \leftarrow \mathsf{r}.$$

Since $\mathcal{P}\cup\mathcal{U}$ is incoherent, some part of $\mathcal{P}$ needs to be eliminated to regain coherence. There are two maximal subsets of $\mathcal{P}$ that are coherent with $\mathcal{U}$: $\{\, \mathsf{p}. \,\}$ and $\{\, \mathsf{q} \leftarrow \mathsf{r}. \,\}$. However, intuition suggests that the former set is preferable since the direct conflict between rules $(\mathsf{q} \leftarrow \mathsf{r}.)$ and $(\neg\mathsf{q} \leftarrow \mathsf{r}.)$ provides a justification for eliminating the rule $(\mathsf{q} \leftarrow \mathsf{r}.)$ and thus keeping the fact $(\mathsf{p}.)$.

The approach taken, then, is to first consider a stable model of $\mathcal{P}$ and update it by $\mathcal{U}$, obtaining a new interpretation $J_{\langle\mathcal{P},\mathcal{U}\rangle}$ that reflects the new information in $\mathcal{U}$. Afterwards, a maximal set of rules from $\mathcal{P}$ coherent with $J_{\langle\mathcal{P},\mathcal{U}\rangle}$ is used to form a prioritised logic program that prefers rules from $\mathcal{U}$ over rules from $\mathcal{P}$. The reducts of this program form the result of the update.

Due to the possibility of having multiple reducts as possible results of the update, it is not completely clear how updates can be iterated. Do we choose one reduct and commit to it? Which one do we choose, then? Or do we simply consider all of the reducts and all possible evolutions? Due to these unresolved issues, we formally define this semantics only for DLPs of length two. We call it *preference-based Zhang's semantics*, or *PRZ-semantics* for short.

Definition 28 (*PRZ-Semantics Zhang 2006*)
Let $\boldsymbol{P} = \langle\mathcal{P},\mathcal{U}\rangle$ be a DLP without default negation in heads of rules. The set $[\![\boldsymbol{P}]\!]_{\mathsf{PRZ}}$ of *PRZ-models of* $\boldsymbol{P}$ is the union of sets of stable models of all reducts obtained by performing the steps 1–4 above.

One distinguishing feature of the PRZ-semantics is that by relying on a stable model of $\mathcal{P}$ for conflict resolution, it is unable to detect "latent" conflicts between rules that have not been "triggered" in the initial stable model or its update. This is illustrated in the following example:[5]

[5] This example does not apply to an earlier version of the PRZ-semantics by Zhang and Foo (1998). This is because the maximal subset $\mathcal{P}'$ of $\mathcal{P}$ chosen for constructing the prioritised logic program is required to be coherent with both $J_{\langle\mathcal{P},\mathcal{U}\rangle}$ and $\mathcal{U}$, not only with $J_{\langle\mathcal{P},\mathcal{U}\rangle}$ as by Zhang (2006).

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Example 29 (*Undetected Latent Conflicts in the PRZ-semantics*)
Consider the programs

$$\mathcal{P}: \quad \mathsf{p} \leftarrow \mathsf{r}. \quad \mathsf{q} \leftarrow \mathsf{r}. \qquad \text{and} \qquad \mathcal{U}: \quad \mathsf{r}. \quad \neg\mathsf{p} \leftarrow \mathsf{q}.$$

and let $\boldsymbol{P} = \langle \mathcal{P}, \mathcal{U} \rangle$. The single stable model of $\mathcal{P}$ is $J_{\mathcal{P}} = \emptyset$ and its update by $\mathcal{U}$ results in the interpretation $J_{\langle \mathcal{P},\mathcal{U} \rangle} = \{ \mathsf{r} \}$ which is coherent with $\mathcal{P}$. The resulting prioritised logic program $(\mathcal{P} \cup \mathcal{U}, \mathcal{P} \times \mathcal{U})$ has only one reduct, $\mathcal{P} \cup \mathcal{U}$, that has no stable model. In other words, $[\![\boldsymbol{P}]\!]_{\mathsf{PRZ}} = \emptyset$ and the conflict between $\mathcal{P}$ and $\mathcal{U}$ remained unresolved. Note also that $[\![\boldsymbol{P}]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}]\!]_{\mathsf{DS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}} = \{ \{ \neg\mathsf{p}, \mathsf{q}, \mathsf{r} \} \}$.

The PRZ-semantics is also sensitive to tautological updates:

Example 30 (*Tautological Updates in the PRZ-semantics*)
Consider the programs

$$\mathcal{P}: \quad \mathsf{p} \leftarrow {\sim}\neg\mathsf{p}. \quad \neg\mathsf{p} \leftarrow {\sim}\mathsf{p}. \qquad \text{and} \qquad \mathcal{U}: \quad \mathsf{p} \leftarrow \mathsf{p}.$$

Both stable models $\{ \mathsf{p} \}$ and $\{ \neg\mathsf{p} \}$ of $\mathcal{P}$ remain unchanged after an update by $\mathcal{U}$ and thus both rules of $\mathcal{P}$ are retained in the resulting prioritised logic program $(\mathcal{P} \cup \mathcal{U}, \mathcal{P} \times \mathcal{U})$. Its only reduct, however, is the program $\{ \mathsf{p} \leftarrow {\sim}\neg\mathsf{p}., \mathsf{p} \leftarrow \mathsf{p}. \}$, which has a single stable model $\{ \mathsf{p} \}$. The tautological update has thus discarded one of the stable models of $\mathcal{P}$.

5.2.2 The PRD_i-, PRW_i, and PRB_i-semantics

Preference-based rule update semantics were also considered by Delgrande *et al.* (2007), utilising the semantics for prioritised logic programs examined by Schaub and Wang (2003).[6] Instead of defining how a prioritised logic program $(\mathcal{P}, \prec)$ can be characterized in terms of reducts, as done by Zhang (2003), Schaub and Wang (2003) specify conditions that a stable model of $\mathcal{P}$ must satisfy in order to be a *preferred stable model of* $(\mathcal{P}, \prec)$. They use three such conditions, defined in the literature on programs with preferences, dubbed *D-preference*, *W-preference* and *B-preference*, which yield an increasing number of preferred stable models. For further details about these preference strategies the reader can refer to the paper by Schaub and Wang (2003) and the references therein.

Unlike in the approach by Zhang (2006), the methodology chosen by Delgrande *et al.* (2007) for performing rule updates is based on relatively simple transformations into a prioritised logic program. In order to define these transformations, we first need to introduce the following notation for arbitrary programs $\mathcal{P}$ and $\mathcal{U}$:

$$\begin{aligned} \mathcal{P}^d &= \left\{ l \leftarrow B(\pi), {\sim}\bar{l}. \;\middle|\; (l \leftarrow B(\pi).) \in \mathcal{P} \right\}, \\ C(\mathcal{P},\mathcal{U}) &= \{(\pi,\sigma) | \exists l \in \mathcal{L} : \pi \in \mathcal{P} \wedge \sigma \in \mathcal{U} \wedge H(\pi) = \{ l \} \wedge H(\sigma) = \{ \bar{l} \}\}, \\ c(\mathcal{P},\mathcal{U}) &= \{ \pi, \sigma \mid (\pi,\sigma) \in C(\mathcal{P},\mathcal{U}) \}. \end{aligned}$$

Intuitively, $\mathcal{P}^d$ denotes a program obtained from $\mathcal{P}$ by making all its rules defeasible, analogically to the semantics based on weakenings by Osorio and Cuevas (2007). The set

[6] Prioritised logic programs are called *ordered logic programs* by Schaub and Wang (2003) and by Delgrande *et al.* (2007).

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

$C(\mathcal{P},\mathcal{U})$ contains pairs of rules from $\mathcal{P}$ and $\mathcal{U}$ with conflicting heads and $c(\mathcal{P},\mathcal{U})$ contains rules from $\mathcal{P}$ and $\mathcal{U}$ involved in such conflicts.

Delgrande *et al.* (2007) proposed three different operators for updating a program $\mathcal{P}$ by a program $\mathcal{U}$, each of which outputs a different prioritised logic program:

$$\mathcal{P} *_0 \mathcal{U} = \left(\mathcal{P}^d \cup \mathcal{U}^d, \mathcal{P}^d \times \mathcal{U}^d\right),$$
$$\mathcal{P} *_1 \mathcal{U} = \left(\mathcal{P}^d \cup \mathcal{U}^d, C(\mathcal{P}^d,\mathcal{U}^d)\right),$$
$$\mathcal{P} *_2 \mathcal{U} = \left(c(\mathcal{P},\mathcal{U})^d \cup ((\mathcal{P} \cup \mathcal{U}) \setminus c(\mathcal{P},\mathcal{U})), C(\mathcal{P}^d,\mathcal{U}^d)\right).$$

Informally, $*_0$ makes all rules from $\mathcal{P}$ and $\mathcal{U}$ defeasible and gives preference to every rule from $\mathcal{U}$ over any rule from $\mathcal{P}$. The operator $*_1$ produces a more cautious preference relation, only preferring rules from $\mathcal{U}$ over rules from $\mathcal{P}$ with conflicting heads. In addition, the operator $*_2$ refrains from making defeasible rules that are not involved in any conflict.

It is argued by Delgrande *et al.* (2007) that these operators can be naturally generalised to account for arbitrary (finite) sequences of programs as follows:

$$*(\langle\mathcal{P}_i\rangle_{i<n}) = \begin{cases} \mathcal{P}_0 * \mathcal{P}_1 & \text{if } n = 2, \\ *(\langle\mathcal{P}_i\rangle_{i<n-1}) * \mathcal{P}_{n-1} & \text{if } n > 2. \end{cases}$$

This definition is slightly incomplete since the result of operators $*_0$, $*_1$, and $*_2$ is not an ordinary logic program but a prioritised one. The question then arises as to what happens with the priority relation of an intermediate result, say $\mathcal{P}_0 * \mathcal{P}_1$, when it is further updated by $\mathcal{P}_2$. In the following, we assume that the preference relations are merged and measures are taken to ensure that the merged relation remains a strict partial order, that is, transitivity is enforced after the merge. We can now define the update semantics by Delgrande *et al.* (2007) for arbitrary DLPs. We call them the *PRX$_i$-semantics* with X representing the preference strategy (i.e., X is one of D, W, or B), and i denoting the particular operator used for forming the prioritised logic program (i.e., $i \in \{0,1,2\}$).

Definition 31 (*PRX$_i$-Semantics Delgrande et al. 2007*)
Let $\boldsymbol{P}$ be a DLP without default negation in heads of rules, X be one of D, W, or B, and $i \in \{0,1,2\}$. The set $[\![\boldsymbol{P}]\!]_{\mathsf{PRX}_i}$ of *PRX$_i$-models of* $\boldsymbol{P}$ is the set of preferred stable models of the prioritised logic program $*_i(\boldsymbol{P})$ under the preference strategy X.

The overall properties of these rule update semantics depend on the chosen operator ($*_0$, $*_1$, or $*_2$) and on the chosen preference strategy (D-, W-, or B-preference). Nevertheless, as Delgrande *et al.* (2007) illustrated by examples, all PRX$_i$-semantics are sensitive to tautological updates. In addition, the following example shows an interesting behaviour that distinguishes these semantics from the previously discussed ones:

Example 32 (*Default Assumptions vs. Facts Delgrande et al. 2007*)
Consider the programs

$$\mathcal{P}: \quad \neg\mathsf{p}. \qquad \text{and} \qquad \mathcal{U}: \quad \mathsf{p} \leftarrow {\sim}\neg\mathsf{p}.$$

and let $\boldsymbol{P} = \langle\mathcal{P},\mathcal{U}\rangle$. For any operator $*_i$ and preference strategy X, $[\![\boldsymbol{P}]\!]_{\mathsf{PRX}_i} = \{\{\mathsf{p}\}\}$. This indicates that the default assumption in the updating program is given preference over the fact in the initial program. If we interpret p as $\mathsf{man}(\mathit{mary})$, then this example shows that if initially $\mathsf{man}(\mathit{mary})$ is known to be false and later we learn that

$$\mathsf{man}(\mathbf{x}) \leftarrow {\sim}\neg\mathsf{man}(\mathbf{x}).,$$

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

meaning that by default all individuals are men, then this immediately changes our knowledge about $\mathsf{man}(mary)$: we now know that $\mathsf{man}(mary)$ is true!

It seems more natural to give preference to initial facts over default assumptions in more recent rules. Note that $[\![\boldsymbol{P}]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}]\!]_{\mathsf{JU}} = \{\{\mathsf{p}\},\{\neg\mathsf{p}\}\}$ and $[\![\boldsymbol{P}]\!]_{\mathsf{DS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}} = [\![\boldsymbol{P}]\!]_{\mathsf{PRZ}} = \{\{\neg\mathsf{p}\}\}$, that is, the causal rejection semantics with unrestricted set of default assumptions allow both $\{\mathsf{p}\}$ and $\{\neg\mathsf{p}\}$ to be stable models of $\boldsymbol{P}$ while the "fixed" versions of these semantics together with Zhang's preference-based semantics actually prefer the initial fact over the default assumption.

5.3 Other approaches

5.3.1 The RVS-semantics

Sakama and Inoue (2003) have proposed a rule update semantics that is clearly based on ideas from belief revision, similarly as formula-based belief update operators. In particular, they define that a program $\mathcal{P}' \cup \mathcal{U}$ achieves the update of $\mathcal{P}$ by $\mathcal{U}$ if $\mathcal{P}'$ is a maximal subset of $\mathcal{P}$ such that $\mathcal{P}' \cup \mathcal{U}$ is coherent, that is, it has a stable model.

As with the PRZ-semantics, we define the semantics of Sakama and Inoue (2003) only for DLPs of length two because it is not clear how one should deal with multiple results of an update. We call the resulting semantics the *RVS-semantics*:

Definition 33 (*RVS-Semantics Sakama and Inoue 2003*)
Let $\boldsymbol{P} = \langle \mathcal{P}, \mathcal{U} \rangle$ be a DLP. The set $[\![\boldsymbol{P}]\!]_{\mathsf{RVS}}$ of *RVS-models of* $\boldsymbol{P}$ is the union of sets of stable models of all programs $\mathcal{P}' \cup \mathcal{U}$ where $\mathcal{P}'$ is a maximal subset of $\mathcal{P}$ such that $\mathcal{P}' \cup \mathcal{U}$ is coherent.

Similarly to the PRZ-semantics, as discussed in Example 27, the approach adopted by the RVS-semantics pays no attention to the source of conflicts – any solution of a conflict is as good as any other as long as only a minimal set of rules is eliminated. Another consequence is that conflicts are removed at any cost, even if there is no plausible way to explain why the update should restore coherence. This has been criticised by Leite (2003), who argued that every conflict has several causes and each type of conflict should be dealt with accordingly. One consequence of this is that an empty or tautological update may restore coherence (and consistency) of an initial program. If we compare this to belief change principles and operators, such a behaviour is typical of *revision* but is not desirable for *updates*. Garcia *et al.* (2019) introduced a family of revision operators for logic programs similar in spirit to the RVS-semantics, except that they allow both for the addition and/or removal of rules from a logic program to achieve coherence. As with the RVS-semantics, these operators are sensitive to empty or tautological updates.

5.3.2 The RVD-semantics

Similar ideas form the basis of the rule *revision* semantics proposed by Delgrande (2010). Note that since the distinction between program update and revision, as these terms are used in the literature, is somewhat blurry, in the following we also refer to this semantics as an *update semantics*. Informally, the stable model of a sequence of programs is constructed by first keeping all rules from the last program and committing to a minimal set of default literals used to derive one of its stable models. Subsequently, a

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

maximal coherent subset of the previous program is added and further commitments are made. This process is iterated until the first program of the sequence is processed, as illustrated by the following example.

Example 34
Consider the DLP

$$\boldsymbol{P} = \langle \{ \mathsf{p}. \}, \{ \mathsf{q}. \}, \{ \mathsf{r} \leftarrow {\sim}\mathsf{p}., \mathsf{r} \leftarrow {\sim}\mathsf{q}. \} \rangle.$$

We start with the last program of the sequence which has the stable model $\{ \mathsf{r} \}$. This stable model can be derived either using the literal ${\sim}\mathsf{p}$ or ${\sim}\mathsf{q}$ and we need to choose one of these and commit to it. If we pick the former, the overall set of literals we commit to at this stage is $\{ {\sim}\mathsf{p}, \mathsf{r} \}$. We then proceed to the second program and realise that it is coherent with our commitments as well as the rules from the last program. We thus add q to our set of commitments and the fact $(\mathsf{q}.)$ to the set of rules that we are going to keep. Proceeding to the first program of the sequence, the rule within it is inconsistent with our commitment to ${\sim}\mathsf{p}$, so the rule needs to be discarded. The set of objective literals we committed to until now, namely $\{ \mathsf{q}, \mathsf{r} \}$, forms one stable model of $\boldsymbol{P}$. Note that if we initially commit to ${\sim}\mathsf{q}$, we obtain the stable model $\{ \mathsf{p}, \mathsf{r} \}$.

To formalise this construction, Delgrande (2010) uses three-valued interpretations, defined as pairs of disjoint interpretations $J = (J^+, J^-)$, which we dub *three-valued d-interpretations* to distinguish from the three-valued interpretations used elsewhere in this paper, based on which a modified notion of reduct of a program $\mathcal{P}$ without default negation in the head is defined as

$$\mathcal{P}^J = \{ \langle H(\pi)^+ \leftarrow B(\pi)^+ \cup {\sim}B(\pi)^- \setminus J^- \rangle | \pi \in \mathcal{P} \wedge J^+ \cap B(\pi)^- = \emptyset \}.$$

This reduct is used to define a special notion of three-valued answer-sets of a program $\mathcal{P}$ which are those three-valued d-interpretations $J = (J^+, J^-)$ such that $\mathsf{least}(\mathcal{P}^{J^+}) = \mathsf{least}(\mathcal{P}^J) = J^+$ and for any $I = (J^+, I^-)$ such that $I^- \subset J^-$ we have that $\mathsf{least}(P^I) \neq J^+$, where $\mathsf{least}(\cdot)$ is as before. Additionally, Delgrande (2010) defines a concept of canonical program corresponding to a three-valued d-interpretation $J = (J^+, J^-)$ as

$$Pgm(J) = \{ \mathsf{p}. \mid \mathsf{p} \in J^+ \} \cup \{ \leftarrow \mathsf{p}. \mid \mathsf{p} \in J^- \}.$$

Then, given a DLP without default negation in heads of rules $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n+1}$, an interpretation J is an *r-answer-set of* $\boldsymbol{P}$ iff there is a sequence $\langle (\mathcal{P}_i^r, J_i) \rangle_{i<n+1}$ such that

1. $\mathcal{P}_n^r = \mathcal{P}_n$ and J_n is a three-valued answer-set of $\mathcal{P}_n$;
2. for $i < n$, $\mathcal{P}_i^r$ is a maximal set of rules of $\mathcal{P}_i$ consistent with $\mathcal{P}_{i+1}^r \cup Pgm(J_{i+1})$ (or $\mathcal{L}$ if $\mathcal{P}_{i+1}^r \cup Pgm(J_{i+1})$ is inconsistent) and J_i is a three-valued answer-set of $\mathcal{P}_i^r$;
3. $J = J_1^+$.

We refer to this semantics as the *RVD-semantics*:

Definition 35 (*RVD-Semantics Delgrande 2010*)
Let $\boldsymbol{P}$ be a DLP without default negation in heads of rules. The set $[\![\boldsymbol{P}]\!]_{\mathsf{RVD}}$ of *RVD-models of* $\boldsymbol{P}$ is the set of r-answer-sets of $\boldsymbol{P}$.

Similarly as the RVS-semantics, the RVD-semantics resolves conflicts at any cost, a consequence of which is that empty and tautological updates restore coherence and consistency. Furthermore, it exhibits the same behaviour as the PRX_i-semantics in

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Example 32, that is, it prefers to satisfy default assumptions in further programs to satisfying earlier facts. It actually goes even further than the PRX_i-semantics, as illustrated in the following example:

Example 36 (*Default Assumptions vs. Facts in the RVD-semantics*)
Consider the programs

$$\mathcal{P}: \quad \mathsf{p}. \qquad \text{and} \qquad \mathcal{U}: \quad \mathsf{q} \leftarrow {\sim}\mathsf{p}.$$

and let $\boldsymbol{P} = \langle \mathcal{P}, \mathcal{U} \rangle$. We obtain $[\![\boldsymbol{P}]\!]_{\mathsf{RVD}} = \{\{\mathsf{q}\}\}$, as opposed to $[\![\boldsymbol{P}]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}]\!]_{\mathsf{DS}} = [\![\boldsymbol{P}]\!]_{\mathsf{RD}} = [\![\boldsymbol{P}]\!]_{\mathsf{PRZ}} = [\![\boldsymbol{P}]\!]_{\mathsf{PRX}_i} = [\![\boldsymbol{P}]\!]_{\mathsf{RVS}} = \{\{\mathsf{p}\}\}$. This indicates that the default assumptions in the updating program are given preference over facts from the initial program even more aggressively than in case of the PRX_i-semantics. If we interpret p as $\mathsf{dog}(bo)$ and q as $\neg\mathsf{canBark}(bo)$, then this example shows that if initially $\mathsf{dog}(bo)$ is known to be true and later we learn that

$$\neg\mathsf{canBark}(\mathbf{x}) \leftarrow {\sim}\mathsf{dog}(\mathbf{x}).,$$

meaning that, by default, individuals that are not dogs cannot bark, then this immediately modifies our knowledge about *bo*: we no longer know whether $\mathsf{dog}(bo)$ is true or not and, in addition, we conclude that $\mathsf{canBark}(bo)$ is (explicitly) false. Poor *bo*...

A methodology based on maximal subsets of the initial program coherent with its update was also used by Osorio and Zepeda (2007) for updating programs under the *pstable model semantics*. The idea is used indirectly by augmenting the bodies of original rules with additional literals and using an abductive framework to minimise the set of rules "disabled" by falsifying the added literal. We do not further consider this semantics because it diverges from the standard notion of a stable model and uses pstable models instead.

Finally, there also exist approaches based on a semantic framework that directly encodes literal dependencies induced by rules, and performs changes on the dependencies instead of on the rules themselves. The advantage over dealing with rules is that the dependency relation is monotonic, so AGM postulates and operators can be applied to it directly (Krümpelmann and Kern-Isberner 2010; Krümpelmann 2012). In the work of Šefránek (2006, 2011), the dependency framework is used for specifying *irrelevant updates*, an instance of which are tautological updates, and designing update semantics immune to such irrelevant updates.

5.4 Fundamental properties

As demonstrated above, rule update semantics are based on a number of different approaches and constructions and provide different results even on very simple examples.

In this section, we indicate and examine some fundamental properties of rule update semantics. We call them *syntactic* because they have been discussed in the context of syntax-based semantics for rule updates and, with only two exceptions, their formulation requires that we refer to the syntax of the respective DLP. The first three properties, as well as the last one, will be satisfied by all rule update semantics that we formally introduced above. The remaining four will only be satisfied by a subset of the semantics, serving as entry points for comparing them.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Table 2. *Applicability of rule update semantics*

Semantics	Applicability
AS, JU, DS, RD	Arbitrary DLPs without integrity constraints
PRX_i, RVD	DLPs without default negation in heads of rules and without integrity constraints
PRZ, RVS	DLPs of length two without default negation in heads of rules and without integrity constraints

Recall that the distinct semantics have been defined for different classes of DLPs, with the assumption that none contained integrity constraints. The properties defined below do not have that assumption. When we say that a semantics S satisfies a particular property, we constrain ourselves only to DLPs in the scope of the definition of S. The classes of DLPs to which the introduced semantics are applicable is summarized in Table 2.

The reader can find a systematic account of the proofs of the theorems in this subsection in the PhD thesis by Slota (2012), either by presenting the proof or pointing to the relevant paper where the result was first proved.

The first fundamental property captures the fact that rule update semantics produce only *supported* models. In a static setting, support (Apt *et al*. 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 straightforward.

Definition 37 (*Support*)
Let S be a rule update semantics, $\mathcal{P}$ a program, l an objective literal and J an interpretation. We say that

- $\mathcal{P}$ *supports* l *in* J if for some rule $\pi \in \mathcal{P}$, $l \in H(\pi)$, and $J \models B(\pi)$;
- $\mathcal{P}$ *supports* J if every objective literal $l \in J$ is supported by $\mathcal{P}$ in J;
- S *respects support* if for every DLP $\boldsymbol{P}$ to which S is applicable and every S-model J of $\boldsymbol{P}$, $\mathsf{all}(\boldsymbol{P})$ supports J.

In other words, a rule update semantics S respects support if every objective literal l that is true in an S-model of a DLP is the head of some rule of that DLP whose body is true in the same model. Such a rule then provides a *justification* for l.

A consequence of support is that the rule update semantics satisfies *language conservation*, defined as follows:

Definition 38 (*Language Conservation for Rule Updates*)
Let S be a rule update semantics. We say that S *conserves the language* if for every set $\mathcal{L}$ of propositional variables, every DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ to which S is applicable, and every S-model J of $\boldsymbol{P}$, if $\mathsf{pr}(\mathcal{P}_i) \subseteq \mathcal{L}$ for all $i < n$, then $\mathsf{pr}(J) \subseteq \mathcal{L}$. Where $\mathsf{pr}(\mathcal{P}_i)$ (resp. $\mathsf{pr}(J)$) denotes the set of all propositional variables appearing in $\mathcal{P}_i$ (resp. J).

Informally, if both the initial and updating theories represent knowledge about propositional variables from the set $\mathcal{L}$, then the updated theory should not introduce knowledge about propositional variables that do not belong to $\mathcal{L}$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Though support and language conservation are basic requirements, and certainly too weak to be sufficient for a "good" rule update semantics, they seem to be intuitive from the logic programming perspective. And, indeed, they are satisfied by all rule update semantics that we introduced previously.

Theorem 39 (*Respect for Support and Language Conservation*)
Let X be one of D, W, B and $i \in \{0, 1, 2\}$. The rule update semantics AS, JU, DS, RD, PRZ, PRX_i, RVS, and RVD respect support and conserve the language.

The third fundamental property for rule update semantics expresses the usual expectation regarding how facts should be updated by newer facts.

Definition 40 (*Fact Update*)
Let S be a rule update semantics. We say that S *respects fact update* if for every finite sequence $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ of consistent sets of facts to which S is applicable, the unique S-model of $\boldsymbol{P}$ is the interpretation

$$\{l \in \mathcal{L} | \exists j < n : (l.) \in \mathcal{P}_j \wedge (\forall i : j < i < n \Longrightarrow \{\, \bar{l}., \sim l. \,\} \cap \mathcal{P}_i = \emptyset)\}.$$

Fact update enforces literal inertia, which forms the basis for belief update operators such as the one by Winslett, but only for the case when both the initial program and its updates are consistent sets of facts. Similarly as before, all rule update semantics adhere to this property.

Theorem 41 (*Respect for Fact Update*)
Let X be one of D, W, B and $i \in \{0, 1, 2\}$. The rule update semantics AS, JU, DS, RD, PRZ, PRX_i, RVS, and RVD respect fact update.

The fourth and fifth syntactic properties are fundamental for all semantics based on causal rejection. The first of them is the causal rejection principle itself.

Definition 42 (*Causal Rejection*)
Let S be a rule update semantics. We say that S *respects causal rejection* if for every DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ to which S is applicable, every S-model J of $\boldsymbol{P}$, all $i < n$, and all rules $\pi \in \mathcal{P}_i$,

$$J \not\models \pi \quad \text{implies} \quad \exists j \, \exists \sigma : i < j < n \wedge \sigma \in \mathcal{P}_j^{\mathsf{e}} \wedge \pi \bowtie \sigma \wedge J \models B(\sigma).$$

This principle requires a *cause* for every violated rule in the form of a more recent rule with a conflicting head and a satisfied body. It is hard-wired in the definitions of sets of rejected rules of the four rule update semantics that are based on it.

Theorem 43 (*Respect for Causal Rejection*)
The rule update semantics AS, JU, DS, and RD respect causal rejection.

Example 44
The following examples illustrate why each of the rule update semantics PRZ, PRX_i, RVS, and RVD does not respect causal rejection.

$\boldsymbol{P}_1 = \langle \{\, \mathsf{p} \leftarrow \mathsf{q}., \neg\mathsf{p} \leftarrow \mathsf{q}. \,\}, \{\, \mathsf{q}. \,\} \rangle$, where $[\![\boldsymbol{P}_1]\!]_{\mathsf{PRZ}} = \{\, \{\, \neg\mathsf{p}, \mathsf{q} \,\}, \{\, \mathsf{p}, \mathsf{q} \,\} \,\}$;
$\boldsymbol{P}_2 = \langle \{\, \mathsf{p}., \neg\mathsf{p}. \,\}, \{\, \mathsf{q} \leftarrow \mathsf{p}. \,\} \rangle$, where $[\![\boldsymbol{P}_2]\!]_{\mathsf{PRB}_0} = [\![\boldsymbol{P}_2]\!]_{\mathsf{PRB}_1} = \{\, \{\, \neg\mathsf{p} \,\}, \{\, \mathsf{p}, \mathsf{q} \,\} \,\}$;

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

$\boldsymbol{P}_3 = \langle \{ \mathsf{p}., \neg \mathsf{p}. \}, \{ \mathsf{q} \leftarrow \mathsf{p}. \} \rangle$, where $[\![\boldsymbol{P}_3]\!]_{\mathsf{PRD}_0} = [\![\boldsymbol{P}_3]\!]_{\mathsf{PRD}_1} = [\![\boldsymbol{P}_3]\!]_{\mathsf{PRW}_0} = [\![\boldsymbol{P}_3]\!]_{\mathsf{PRW}_1} = \{ \{ \neg \mathsf{p} \} \}$;

$\boldsymbol{P}_4 = \langle \{ \mathsf{p}., \neg \mathsf{p}. \}, \{ \mathsf{p} \leftarrow \mathsf{q}., \neg \mathsf{p} \leftarrow \mathsf{q}. \} \rangle$, where $[\![\boldsymbol{P}_4]\!]_{\mathsf{PRB}_2} = [\![\boldsymbol{P}_4]\!]_{\mathsf{PRD}_2} = [\![\boldsymbol{P}_4]\!]_{\mathsf{PRW}_2} = \{ \{ \neg \mathsf{p} \}, \{ \mathsf{p} \} \}$;

$\boldsymbol{P}_5 = \langle \{ \mathsf{p} \leftarrow {\sim}\mathsf{p}. \}, \{ \} \rangle$, where $[\![\boldsymbol{P}_5]\!]_{\mathsf{RVS}} = \{ \{ \} \}$; and

$\boldsymbol{P}_6 = \langle \{ \mathsf{p}. \}, \{ \mathsf{q} \leftarrow {\sim}\mathsf{p}. \} \rangle$, where $[\![\boldsymbol{P}_6]\!]_{\mathsf{RVD}} = \{ \{ \mathsf{q} \} \}$.

The sixth 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.

Definition 45 (*Acyclic Justified Update*)
Let S be a rule update semantics. We say that S *respects acyclic justified update* if for every acyclic DLP $\boldsymbol{P}$ to which S is applicable, the set of S-models of $\boldsymbol{P}$ coincides with $[\![\boldsymbol{P}]\!]_{\mathsf{JU}}$.

Theorem 46 (*Respect for Acyclic Justified Update*)
The rule update semantics AS, JU, DS, and RD respect acyclic justified update.

The next property has been extensively discussed in the literature, and has been at the heart of the motivation for developing some of the variants of semantics based on causal rejection. It requires that the semantics be immune to tautological updates (i.e., an update composed only of rules whose head literal also belongs to its body), the intuition being that such update cannot indicate any change in the modelled world because it is always true. Immunity to tautological updates is a desirable property of belief updates in classical logic, being a direct consequence of postulate (BU2).

Definition 47 (*Immunity to Tautological Updates*)
Let S be a rule update semantics, π a tautological rule, and $\boldsymbol{P}^{\pi}$ the DLP obtained from DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ by appending $\mathcal{P}_n = \{\pi\}$. We say that S *respects immunity to tautological updates* if for every DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ and every tautological rule π such that S is applicable to $\boldsymbol{P}^{\pi}$, $[\![\boldsymbol{P}]\!]_{\mathsf{S}} = [\![\boldsymbol{P}^{\pi}]\!]_{\mathsf{S}}$.

As it turns out, only the RD-semantics is immune to tautological updates.

Theorem 48 (*Immunity to Tautologies*)
The rule update semantics RD respects immunity to tautological updates.

The remaining semantics are not immune to tautologies for one of the following three reasons:

1. the tautology can be used to reject other rules, as is the case with most semantics based on causal rejection, namely AS, JU, and DS, such as for example

$\boldsymbol{P}_1 = \langle \{ \mathsf{p}., \neg \mathsf{p}. \}, \{ \mathsf{p} \leftarrow \mathsf{p}. \} \rangle$, where $[\![\boldsymbol{P}_1]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}_1]\!]_{\mathsf{AS}} = [\![\boldsymbol{P}_1]\!]_{\mathsf{DS}} = \{ \{ \mathsf{p} \} \}$;

$\boldsymbol{P}_2 = \langle \{ \mathsf{p}. \}, \{ \neg \mathsf{p}. \}, \{ \mathsf{p} \leftarrow \mathsf{p}. \} \rangle$, where $[\![\boldsymbol{P}_2]\!]_{\mathsf{AS}} = \{ \{ \neg \mathsf{p} \} \{ \mathsf{p} \} \}$;

$\boldsymbol{P}_3 = \langle \{ \mathsf{p}. \}, \{ {\sim}\mathsf{p} \leftarrow {\sim}\mathsf{p}. \} \rangle$, where $[\![\boldsymbol{P}_3]\!]_{\mathsf{JU}} = [\![\boldsymbol{P}_3]\!]_{\mathsf{AS}} = \{ \emptyset, \{ \mathsf{p} \} \}$;

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

2. the tautology can be used as a reason not to prefer other rules, as is the case with semantics based on preferences, namely PRZ and PRX_i (for X be one of D, W, B and $i \in \{0, 1, 2\}$), such as for example

$$\boldsymbol{P}_4 = \langle \{\, \mathsf{p} \leftarrow {\sim}\neg\mathsf{p}.,\ \neg\mathsf{p} \leftarrow {\sim}\mathsf{p}.\,\}, \{\, \mathsf{p} \leftarrow \mathsf{p}.\,\}\rangle, \text{ where } [\![\boldsymbol{P}_4]\!]_{\mathsf{PRZ}} = \{\{\,\mathsf{p}\,\}\};$$

$$\boldsymbol{P}_5 = \langle \{\, \mathsf{p} \leftarrow {\sim}\neg\mathsf{p}.,\ \neg\mathsf{p}.\,\}, \{\, \mathsf{p} \leftarrow \mathsf{p}.\,\}\rangle, \text{ where } [\![\boldsymbol{P}_5]\!]_{\mathsf{PRX_i}} = \{\{\,\mathsf{p}\,\}\};$$

3. conflicts are resolved at any cost, as is the case with the RVS and RVD semantics, such as for example

$$\boldsymbol{P}_6 = \langle \{\, \mathsf{p} \leftarrow {\sim}\mathsf{p}.\,\}, \{\, \mathsf{q} \leftarrow \mathsf{q}.\,\}\rangle, \text{ where } [\![\boldsymbol{P}_6]\!]_{\mathsf{RVS}} = [\![\boldsymbol{P}_6]\!]_{\mathsf{RVD}} = \{\emptyset\};$$

$$\boldsymbol{P}_7 = \langle \{\, \mathsf{p}.,\ \neg\mathsf{p}.\,\}, \{\, \mathsf{q} \leftarrow \mathsf{q}.\,\}\rangle, \text{ where } [\![\boldsymbol{P}_7]\!]_{\mathsf{RVS}} = [\![\boldsymbol{P}_7]\!]_{\mathsf{RVD}} = \{\{\,\mathsf{p}\,\}, \{\,\neg\mathsf{p}\,\}\}.$$

The next two properties are no longer syntactical as the ones considered so far, but we mention them here since they have been often discussed together with the semantics considered in this section.

The first one can be seen as a weaker version of immunity to tautological updates. It imposes that semantics be immune to empty updates, and is obeyed by a larger set of semantics.

Definition 49 (*Immunity to Empty Updates*)
Let S be a rule update semantics and $\boldsymbol{P}^{\emptyset}$ the DLP obtained from DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ by appending $\mathcal{P}_n = \emptyset$. We say that *S respects immunity to empty updates* if for every DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i<n}$ such that S is applicable to $\boldsymbol{P}^{\emptyset}$, $[\![\boldsymbol{P}]\!]_{\mathsf{S}} = [\![\boldsymbol{P}^{\emptyset}]\!]_{\mathsf{S}}$.

Theorem 50 (*Immunity to Empty Updates*)
Let X be one of D, W, or B. The rule update semantics AS, JU, DS, RD, PRZ, and PRX_2 respect immunity to empty updates.

There has been a considerable amount of discussion regarding whether immunity to empty updates should be considered a desirable property. At the heart of such discussion is the fact that most counter-examples used to show why some semantics does not obey this property involve incoherent/contradictory programs whose coherence/consistency is restored through an empty update. Two arguments are usually used to support the adequacy of this property, and why coherence/consistency should not be regained no matter what.

The first argument is conceptual, and related to the difference between *revision* and *update*. Since revision is about incorporating *better* knowledge about some world that did not change, there is an implicit assumption that the initial knowledge was not *perfect* (*complete*, *correct*,...), which can somehow be used to justify the removal of inconsistencies/incoherences, even if better explicit knowledge to incorporate is not available, as indicated by an empty program. On the contrary, an update is about incorporating *new* knowledge about a world that changed. The knowledge about the *old* world might be a bad representation of the *new* world, but there is no underlying assumption that it was an imperfect representation of the *old* world, that is, it could simply be that the *old* world being represented is simply incoherent/inconsistent. And, if nothing changed in that incoherent/inconsistent world, as indicated by the empty program, we should still take it to remain incoherent/inconsistent.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The second argument is more pragmatic, and tied to the answer-set programming methodology for problem solving (Marek and Truszczyński 1999; Niemelä 1999; Lifschitz 1999), where stable models correspond to solutions to the problem, and the lack of existing stable models simply means that the problem has no solutions. As a concrete example, suppose that a logic program encodes the well-known n-queens problem, and includes some facts encoding the queens that have already been placed on the chessboard. Stable models will correspond to possible solutions to the n-queens problem, indicating where queens should be further placed, given the ones already on the board. To enforce this, the program would have several rules, acting as integrity constraints, encoding that no two queens should be placed in the same column, line or diagonal. Suppose that the initial program encodes a situation where two queens have already been placed in the same column, hence without stable models. It seems clear that an update by an empty program, encoding that no further queens have been added or removed, should not change the fact that those two queens are still attacking each other, and no solution can exist with them on the board, that is, the program should still have no stable models after the update.

The last property mentioned here – primacy of new information (Dalal 1988) – is at the heart of every update operator, independently of the base formalism for which it is defined. It corresponds to postulate (BR1) in belief updates, and conveys the fact that the models produced by a rule update semantics should conform to the new information, captured by the following definition:

Definition 51 (*Primacy of New Information*)
Let S be a rule update semantics. We say that S *respects primacy of new information* if for every DLP $\boldsymbol{P} = \langle \mathcal{P}_i \rangle_{i \leq n}$ to which S is applicable and every S-model J of $\boldsymbol{P}$, $J \models \mathcal{P}_n$.

Primacy of new information is satisfied by all rule update semantics that we considered so far.

Theorem 52 (*Respect for Primacy of New Information*)
Let X be one of D, W, or B and $i \in \{0, 1, 2\}$. The rule update semantics AS, JU, DS, RD, PRZ, PRX_i, RVS, and RVD respect primacy of new information.

6 The third era – semantics-based updates

Though useful in practical scenarios (Alferes *et al*. 2003; Saias and Quaresma 2004; Siska 2006; Ilic *et al*. 2008; Slota *et al*. 2011), it turned out that most of the syntax-based semantics exhibit some undesirable behaviour. For example, except for the semantics proposed by Alferes *et al*. (2005), a tautological update may influence the result under all of these semantics, a behaviour that is highly undesirable when considering knowledge updates. But, more importantly, the common feature of all of these semantics is that they make heavy use of the syntactic structure of programs and rules, while lacking some semantic characterisation that would allow, for example, some adequate notion of equivalence under updates. This is a well known problem associated with these semantics (Eiter *et al*. 2002; Leite 2003; Slota 2012), which has prevented a more thorough analysis and understanding of their semantic properties.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

6.1 Operators based on HT-models

Recently, AGM revision was reformulated in the context of logic programming in a manner analogous to belief revision in classical propositional logic, and specific revision operators for logic programs were investigated by Delgrande *et al.* (2008, 2013) and by Osorio and Cuevas (2007). Central to this novel approach are *HT-models*, based on the logic of here-and-there (Heyting 1930; Pearce 1997), which provide a monotonic semantic characterisation of logic programs that is strictly more expressive than the answer-set semantics in the sense that it is possible to determine the answer-sets of a program from its set of HT-models, but not vice-versa that is, there are programs with the same set of answer-sets but different sets of HT-models. Furthermore, two programs have the same set of HT-models if and only if they are strongly equivalent (Lifschitz *et al.* 2001), which means that programs $\mathcal{P}, \mathcal{Q}$ with the same set of HT-models can be modularly replaced by one another, even in the presence of additional rules, without affecting the resulting answer-sets.

Indeed, these investigations constitute an important breakthrough in the research of answer-set program evolution. They change the focus from the syntactic representation of a program, where not all rules and literal occurrences are necessarily relevant to its meaning as a whole, to its semantic content, that is, to the information that the program is intended to represent.

Subsequently, Slota and Leite (2010, 2014) followed a similar path, but to tackle the problem of answer-set program *updates*, instead of *revision* as tackled by Delgrande *et al.* (2008, 2013). The studied operators are semantic in their very nature and in line with KM postulates for updates, in contrast with the traditional syntax-based approaches to rule updates described in the previous section. Others have followed this path of defining change operators for logic programs characterised by their HT-models. For example, Zhuang *et al.* (2016) introduced a revision operator that adds and/or removes rules from a logic program while minimising the symmetric difference between HT-models. Binnewies *et al.* (2018) introduced partial meet and ensconcement constructions for logic program belief change, which allowed them to define revision and contraction operators.

In this subsection, we first briefly review the revision approach of Delgrande *et al.* (2008, 2013) and the update approach of Slota and Leite (2010, 2014). Then, we discuss a serious drawback which extends to all rule update operators that characterise logic programs through their HT-models. It turns out that both these revision and the update operators are incompatible with the properties of *support* and *fact update*, which are at the core of rule updates (cf. Theorems 39 and 41). This is a very important finding as it guides the research on rule revision and updates away from the semantic approach materialised in AGM and KM postulates or, alternatively, to the development of semantic characterisations of programs, richer than HT-models, that are appropriate for describing their dynamic behaviour.

Before we proceed, and in order to reformulate some of the postulates for programs under the HT-models semantics, we assume some given program conjunction and disjunction operators $\dot{\wedge}$, $\dot{\vee}$, where each assigns, to each pair of programs, a program whose set of HT-models is the intersection and union, respectively, of the sets of HT-models of argument programs. The program conjunction operator may simply return the union of argument programs – it is the same as the *expansion operator* defined by Delgrande *et al.*

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

(2008, 2013) – while the program disjunction operator can be defined by translating the argument programs into the logic of here-and-there (Heyting 1930; Łukasiewicz 1941; Pearce 1997), taking their disjunction, and transforming the resulting formula back into a logic program (using results by Cabalar and Ferraris 2007).

In this section, we no longer restrict programs and DLPs to those without integrity constraints.

6.1.1 Revision based on HT-models

For a rule revision operator $\otimes$ and programs $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{U}$, $\mathcal{V}$, acording to Delgrande *et al.* (2008, 2013), the KM postulates for belief revision are adapted to answer-set program revision using *HT-models* as follows:

$(\mathsf{PR1})_{\mathsf{HT}}$ $\mathcal{P} \otimes \mathcal{U} \models_{\mathsf{HT}} \mathcal{U}$.
$(\mathsf{PR2})_{\mathsf{HT}}$ If $[\![\mathcal{P} \dot{\wedge} \mathcal{Q}]\!]_{\mathsf{HT}} \neq \emptyset$, then $\mathcal{P} \otimes \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{P} \dot{\wedge} \mathcal{Q}$.
$(\mathsf{PR3})_{\mathsf{HT}}$ If $[\![\mathcal{U}]\!]_{\mathsf{HT}} \neq \emptyset$, then $[\![\mathcal{P} \otimes \mathcal{U}]\!]_{\mathsf{HT}} \neq \emptyset$.
$(\mathsf{PR4})_{\mathsf{HT}}$ If $\mathcal{P} \equiv_{\mathsf{HT}} \mathcal{Q}$ and $\mathcal{U} \equiv_{\mathsf{HT}} \mathcal{V}$, then $\mathcal{P} \otimes \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{Q} \otimes \mathcal{V}$.
$(\mathsf{PR5})_{\mathsf{HT}}$ $(\mathcal{P} \otimes \mathcal{U}) \dot{\wedge} \mathcal{V} \models_{\mathsf{HT}} \mathcal{P} \otimes (\mathcal{U} \dot{\wedge} \mathcal{V})$.
$(\mathsf{PR6})_{\mathsf{HT}}$ If $[\![(\mathcal{P} \otimes \mathcal{U}) \dot{\wedge} \mathcal{V}]\!]_{\mathsf{HT}} \neq \emptyset$ then $\mathcal{P} \otimes (\mathcal{U} \dot{\wedge} \mathcal{V}) \models_{\mathsf{HT}} (\mathcal{P} \otimes \mathcal{U}) \dot{\wedge} \mathcal{V}$.

As discussed by Delgrande *et al.* (2008, 2013), no $\otimes$ can be immune to tautologies or empty updates – whenever $\mathcal{P}$ is unsatisfiable and $\mathcal{U}$ is tautological or empty, immunity to tautologies and immunity to empty updates conflicts with $(\mathsf{PR3})_{\mathsf{HT}}$. As also discussed by Delgrande *et al.* (2008, 2013), immunity to tautologies and empty updates could have been adopted instead of $(\mathsf{PR3})_{\mathsf{HT}}$.

Analogically to belief revision, a constructive characterisation of rule revision operators satisfying conditions $(\mathsf{PR1})_{\mathsf{HT}}$–$(\mathsf{PR6})_{\mathsf{HT}}$ is based on an order assignment. Since the set of HT-models of any program $\mathcal{P}$ must be well-defined, that is, $(I, J) \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$ implies that $(J, J) \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$, not every order assignment characterises a rule revision operator. Slota and Leite (2010, 2014) additionally define *well-defined order assignments* as those that do.

Definition 53 (*Rule Revision Operator Characterised by an Order Assignment*)
Let $\otimes$ be a rule revision operator and ω a preorder assignment over the set of all logic programs. We say that $\otimes$ is *characterised by* ω if for all programs $\mathcal{P}$, $\mathcal{U}$,

$$[\![\mathcal{P} \otimes \mathcal{U}]\!]_{\mathsf{HT}} = \min\left([\![\mathcal{U}]\!]_{\mathsf{HT}}, \leq_{\omega}^{\mathcal{P}}\right).$$

A preorder assignment over over the set of all programs is *well-defined* if some rule update operator is characterised by it.

Similarly as with belief revision, the following definition captures a set of conditions on the assigned orders.

Definition 54 (*Faithful Order Assignment Over Programs*)
A preorder assignment ω over the set of all programs is *faithful* if the following three conditions hold, for all programs $\mathcal{P}$, $\mathcal{Q}$:

- If $I, J \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$, then $I \not<_{\omega}^{\mathcal{P}} J$;
- If $I \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$ and $J \notin [\![\mathcal{P}]\!]_{\mathsf{HT}}$, then $I <_{\omega}^{\mathcal{P}} J$;
- If $[\![\mathcal{P}]\!]_{\mathsf{HT}} = [\![\mathcal{Q}]\!]_{\mathsf{HT}}$, then $<_{\omega}^{\mathcal{P}} = <_{\omega}^{\mathcal{Q}}$.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

The representation theorem of Delgrande *et al.* (2013) states that operators characterised by faithful order assignments over the set of all programs are exactly those that satisfy $(\mathsf{PR1})_{\mathsf{HT}}$–$(\mathsf{PR6})_{\mathsf{HT}}$.

Theorem 55 (*Delgrande et al. 2013*)
Let $\otimes$ be a rule revision operator. Then the following conditions are equivalent:

- The operator $\otimes$ satisfies conditions $(\mathsf{PR1})_{\mathsf{HT}}$–$(\mathsf{PR6})_{\mathsf{HT}}$.
- The operator $\otimes$ is characterised by a faithful total preorder assignment over the set of all programs.

Schwind and Inoue (2016) provided an alternative constructive characterisation of logic programs revision operators in terms of preorders over interpretations. Delgrande *et al.* (2008, 2013) present two different revision operators, one analogous to the so-called set-containment based revision proposed by Satoh (1988), and the other analogous to the so-called cardinality-based revision proposed by Dalal (1988). Here, we recap the latter one, because, unlike the former, it obeys all six postulates $(\mathsf{PR1})_{\mathsf{HT}}$–$(\mathsf{PR6})_{\mathsf{HT}}$.

The operator is based on a notion of *closeness* that is given in terms of cardinality:

Definition 56
Let $\mathcal{M}$ and $\mathcal{N}$ be sets of either interpretations or three-valued interpretations. Then,

$$\sigma_{||}(\mathcal{M},\mathcal{N}) = \{X \in \mathcal{M} | \exists Y \in \mathcal{N} \text{ such that } \forall X' \in \mathcal{M}, \forall Y' \in \mathcal{N}, |X' \div Y'| \not< |X \div Y|\},$$

where $\div$ denotes the symmetric difference, extended for three-valued interpretations $\langle I, J\rangle$ and $\langle K, L\rangle$ as follows:

$$\langle I, J\rangle \div \langle K, L\rangle = \langle I \div K, J \div L\rangle,$$

and

$$|\langle I, J\rangle| \leq |\langle K, L\rangle| \text{ iff } |J| \leq |L| \text{ and if } |J| = |L| \text{ then } |I| \leq |K|;$$

$$|\langle I, J\rangle| < |\langle K, L\rangle| \text{ iff } |\langle I, J\rangle| \leq |\langle K, L\rangle| \text{ and } |\langle K, L\rangle| \not\leq |\langle I, J\rangle|.$$

The operator is then defined as follows:

Definition 57 (*Cardinality-based Program Revision Delgrande et al. 2008, 2013*)
Let $\mathcal{P}$ and $\mathcal{U}$ be two logic programs. The cardinality-based revision operator $\otimes_c$ is defined, up to equivalence of its input and output, as a logic program such that:

$$[\![\mathcal{P} \otimes_c \mathcal{Q}]\!]_{\mathsf{HT}} = [\![\mathcal{Q}]\!]_{\mathsf{HT}}, \qquad \text{if } [\![\mathcal{P}]\!]_{\mathsf{HT}} = \emptyset;$$

$$[\![\mathcal{P} \otimes_c \mathcal{Q}]\!]_{\mathsf{HT}} = \{\langle I, J\rangle | J \in \sigma_{||}([\![\mathcal{P}]\!]_{\mathsf{C}}, [\![\mathcal{Q}]\!]_{\mathsf{C}}), I \subseteq J, \text{ and if } I \subset J \text{ then } \langle I, J\rangle \in \sigma_{||}([\![\mathcal{P}]\!]_{\mathsf{HT}}, [\![\mathcal{Q}]\!]_{\mathsf{HT}})\}, \qquad \text{otherwise.}$$

Theorem 58
The cardinality based operator $\otimes_c$ satisfies $(\mathsf{PR1})_{\mathsf{HT}}$–$(\mathsf{PR6})_{\mathsf{HT}}$.

6.1.2 Updates based on HT-models

To adapt the KM postulates for updates to answer-set program updates, Slota and Leite (2010, 2014) substitute the notion of a *complete formula* used in (BU7) with the notion of a *basic program*, that is, a program that either it has a unique HT-model (J, J), or a

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

pair of HT-models (I, J) and (J, J). In the former case, the program exactly determines the truth values of all atoms – the atoms in J are true and the remaining atoms are false. In the latter case, the program makes atoms in I true, the atoms in $J \setminus I$ may either be undefined or true, as long as they all have the same truth value, and the remaining atoms are false. The latter case needs to be allowed in order to make the new postulate applicable to three-valued interpretations (I, J) with $I \subsetneq J$ because no program has the single HT-model (I, J).

Using *HT-models*, the adaptation of the KM postulates for updates to answer-set program updates, which, for a rule update operator $\oplus$ and programs $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{U}$, $\mathcal{V}$ become:

$(\mathsf{PU1})_{\mathsf{HT}}$ $\mathcal{P} \oplus \mathcal{U} \models_{\mathsf{HT}} \mathcal{U}$.
$(\mathsf{PU2})_{\mathsf{HT}}$ If $\mathcal{P} \models_{\mathsf{HT}} \mathcal{U}$, then $\mathcal{P} \oplus \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{P}$.
$(\mathsf{PU3})_{\mathsf{HT}}$ If $[\![\mathcal{P}]\!]_{\mathsf{HT}} \neq \emptyset$ and $[\![\mathcal{U}]\!]_{\mathsf{HT}} \neq \emptyset$, then $[\![\mathcal{P} \oplus \mathcal{U}]\!]_{\mathsf{HT}} \neq \emptyset$.
$(\mathsf{PU4})_{\mathsf{HT}}$ If $\mathcal{P} \equiv_{\mathsf{HT}} \mathcal{Q}$ and $\mathcal{U} \equiv_{\mathsf{HT}} \mathcal{V}$, then $\mathcal{P} \oplus \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{Q} \oplus \mathcal{V}$.
$(\mathsf{PU5})_{\mathsf{HT}}$ $(\mathcal{P} \oplus \mathcal{U}) \mathbin{\dot{\wedge}} \mathcal{V} \models_{\mathsf{HT}} \mathcal{P} \oplus (\mathcal{U} \mathbin{\dot{\wedge}} \mathcal{V})$.
$(\mathsf{PU6})_{\mathsf{HT}}$ If $\mathcal{P} \oplus \mathcal{U} \models_{\mathsf{HT}} \mathcal{V}$ and $\mathcal{P} \oplus \mathcal{V} \models_{\mathsf{HT}} \mathcal{U}$, then $\mathcal{P} \oplus \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{P} \oplus \mathcal{V}$.
$(\mathsf{PU7})_{\mathsf{HT}}$ If $\mathcal{P}$ is basic, then $(\mathcal{P} \oplus \mathcal{U}) \mathbin{\dot{\wedge}} (\mathcal{P} \oplus \mathcal{V}) \models_{\mathsf{HT}} \mathcal{P} \oplus (\mathcal{U} \mathbin{\dot{\vee}} \mathcal{V})$.
$(\mathsf{PU8})_{\mathsf{HT}}$ $(\mathcal{P} \mathbin{\dot{\vee}} \mathcal{Q}) \oplus \mathcal{U} \equiv_{\mathsf{HT}} (\mathcal{P} \oplus \mathcal{U}) \mathbin{\dot{\vee}} (\mathcal{Q} \oplus \mathcal{U})$.

Analogically to belief updates, a constructive characterisation of rule update operators satisfying conditions $(\mathsf{PU1})_{\mathsf{HT}}$–$(\mathsf{PU8})_{\mathsf{HT}}$ is based on an order assignment, but this time over the set of all three-valued interpretations $\mathfrak{X}$. Since the set of HT-models of any program $\mathcal{P}$ must be well-defined that is, $(I, J) \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$ implies that $(J, J) \in [\![\mathcal{P}]\!]_{\mathsf{HT}}$, not every order assignment characterises a rule update operator. Slota and Leite (2010, 2014) additionally define *well-defined order assignments* as those that do.

Definition 59 (*Rule Update Operator Characterised by an Order Assignment*)
Let $\oplus$ be a rule update operator and ω a preorder assignment over $\mathfrak{X}$. We say that $\oplus$ is *characterised by* ω if for all programs $\mathcal{P}, \mathcal{U}$,

$$[\![\mathcal{P} \oplus \mathcal{U}]\!]_{\mathsf{HT}} = \bigcup_{X \in [\![\mathcal{P}]\!]_{\mathsf{HT}}} \min\left([\![\mathcal{U}]\!]_{\mathsf{HT}}, \leq_\omega^X\right).$$

A preorder assignment over $\mathfrak{X}$ is *well-defined* if some rule update operator is characterised by it.

Similarly as with belief updates, order assignments are required to be faithful, that is to consider each three-valued interpretation the closest to itself.

Definition 60 (*Faithful Order Assignment Over Three-valued Interpretations*)
A preorder assignment ω over $\mathfrak{X}$ is *faithful* if for every three-valued interpretation X the following condition is satisfied:

$$\text{For every } Y \in \mathfrak{X} \text{ with } Y \neq X \text{ it holds that } X <_\omega^X Y.$$

Interestingly, faithful assignments characterise the same class of operators as the larger class of semi-faithful assignments, defined as follows:

Definition 61 (*Semi-Faithful Order Assignment*)
A preorder assignment ω over $\mathfrak{X}$ is *semi-faithful* if for every three-valued interpretation $X = (I, J)$ the following conditions are satisfied (where $X^* = (J, J)$):

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

For every $Y \in \mathfrak{X}$ with $Y \neq X$ and $Y \neq X^*$, either $X <_\omega^X Y$ or $X^* <_\omega^X Y$.

If $X^* \leq_\omega^X X$, then $X \leq_\omega^X X^*$.

Preorder assignments need to satisfy one further condition, related to the well-definedness of sets of SE-models of every program. It can naturally be seen as the semantic counterpart of $(\mathsf{PU7})_{\mathsf{HT}}$.

Definition 62 (*Organised Preorder Assignment*)
A preorder assignment ω is *organised* if for all three-valued interpretations X, Y and all well-defined sets of three-valued interpretations $\mathcal{M}, \mathcal{N}$ the following condition is satisfied:

If $Y \in \min(\mathcal{M}, \leq_\omega^X) \cup \min(\mathcal{M}, \leq_\omega^{X^*})$ and $Y \in \min(\mathcal{N}, \leq_\omega^X) \cup \min(\mathcal{N}, \leq_\omega^{X^*})$,

then $Y \in \min(\mathcal{M} \cup \mathcal{N}, \leq_\omega^X) \cup \min(\mathcal{M} \cup \mathcal{N}, \leq_\omega^{X^*})$.

Just as with updates in classical logic, the following representation theorem provides a constructive characterisation of rule update operators satisfying the postulates, making it possible to define and evaluate any operator satisfying the postulates using an intuitive construction.

Theorem 63 (*Representation theorem for rule updates Slota and Leite 2014*)
Let $\oplus$ be a rule update operator. The following conditions are equivalent:

1. The operator $\oplus$ satisfies conditions $(\mathsf{PU1})_{\mathsf{HT}}$–$(\mathsf{PU8})_{\mathsf{HT}}$.
2. The operator $\oplus$ is characterised by a semi-faithful and organised preorder assignment.
3. The operator $\oplus$ is characterised by a faithful and organised partial order assignment.

One of the benefits of dealing with rule updates on the semantic level is that semantic properties that are rather difficult to show for syntax-based update operators are much easier to analyse and prove. As we have seen, one of the most widespread and counterintuitive side-effects of syntax-based rule update semantics is that they are sensitive to tautological updates. In case of the semantic update operators characterised in the previous theorem, such a behaviour is impossible given that the operators satisfy $(\mathsf{PU2})_{\mathsf{HT}}$ and $(\mathsf{PU4})_{\mathsf{HT}}$.

Slota and Leite (2014) additionally defined a concrete update operator that can be seen as a counterpart to the belief update operator by Winslett (1988), which we do not present here.

6.1.3 Problems with revision and update operators based on HT-models

The most important contribution by Slota and Leite (2010, 2014) is not the adaptation of the postulates to answer-set program updates, the representation theorem, nor even the concrete update operator they defined, but rather the uncovering of a serious drawback that extends to all rule update and revision operators based on KM postulates and on HT-models. In particular, it turns out that these operators are incompatible with the properties of *support* and *fact update* which are at the core of logic programming and their updates.

The following theorem shows that every rule update operator satisfying $(\mathsf{PU4})_{\mathsf{HT}}$ violates either *support* or *fact update*, while its proof illustrates why.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Theorem 64 (*Slota and Leite 2014*)
A rule update operator that satisfies $(\mathsf{PU4})_{\mathsf{HT}}$ either does not respect *support* or it does not respect *fact update*.

Proof
Let $\oplus$ be a rule update operator that satisfies $(\mathsf{PU4})_{\mathsf{HT}}$ and $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{U}$ the following programs:

$$\mathcal{P}: \quad \mathsf{p}. \quad \mathsf{q}. \qquad \mathcal{Q}: \quad \mathsf{p} \leftarrow \mathsf{q}. \quad \mathsf{q}. \qquad \mathcal{U}: \quad {\sim}\mathsf{q}.$$

Since $\mathcal{P}$ is strongly equivalent to $\mathcal{Q}$, by $(\mathsf{PU4})_{\mathsf{HT}}$ we obtain that $\mathcal{P} \oplus \mathcal{U}$ is strongly equivalent to $\mathcal{Q} \oplus \mathcal{U}$. Consequently, $\mathcal{P} \oplus \mathcal{U}$ has the same answer-sets as $\mathcal{Q} \oplus \mathcal{U}$. It only remains to observe that if $\oplus$ respects fact update, then $\mathcal{P} \oplus \mathcal{U}$ has the unique answer-set $\{\mathsf{p}\}$. But then $\{\mathsf{p}\}$ is an answer-set of $\mathcal{Q} \oplus \mathcal{U}$ in which p is unsupported by $\mathcal{Q} \cup \mathcal{U}$. Hence $\oplus$ does not respect support. □

So, any answer-set program update operator based on HT-models and the KM approach to belief update, as materialised in the fundamental principle $(\mathsf{PU4})_{\mathsf{HT}}$, cannot respect two basic and desirable properties: *support* and *fact update*. This is a major drawback of such operators, severely diminishing their applicability.

Moreover, the principle $(\mathsf{PU4})_{\mathsf{HT}}$ is also adopted as $(\mathsf{PR4})_{\mathsf{HT}}$ for *revision* of answer-set programs based on HT-models by Delgrande *et al.* (2013). This means that Theorem 64 extends to semantic program revision operators, such as those defined by Delgrande *et al.* (2013): whenever support and fact update are expected to be satisfied by a rule revision operator, it cannot be defined by purely manipulating the sets of HT-models of the underlying programs.

One question that suggests itself is whether a weaker version of the principle $(\mathsf{PU4})_{\mathsf{HT}}$ can be combined with properties such as support and fact update. Its two immediate weakenings, analogous to the weakenings of (BU4) by Herzig and Rifi (1999), are as follows:

$(\mathsf{PU4.1})_{\mathsf{HT}}$ If $\mathcal{P} \equiv_{\mathsf{HT}} \mathcal{Q}$, then $\mathcal{P} \oplus \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{Q} \oplus \mathcal{U}$.
$(\mathsf{PU4.2})_{\mathsf{HT}}$ If $\mathcal{U} \equiv_{\mathsf{HT}} \mathcal{V}$, then $\mathcal{P} \oplus \mathcal{U} \equiv_{\mathsf{HT}} \mathcal{P} \oplus \mathcal{V}$.

In case of $(\mathsf{PU4.1})_{\mathsf{HT}}$, it is easy to see that the proof of Theorem 64 applies in the same way as with $(\mathsf{PU4})_{\mathsf{HT}}$, so $(\mathsf{PU4.1})_{\mathsf{HT}}$ is likewise incompatible with support and fact update.

On the other hand, principle $(\mathsf{PU4.2})_{\mathsf{HT}}$, also referred to as *weak independence of syntax* (WIS) (Osorio and Cuevas 2007), does not suffer from such severe limitations. It is, nevertheless, violated by syntax-based rule update semantics that assign a special meaning to occurrences of default literals in heads of rules, as illustrated in the following example:

Example 65
Let the programs $\mathcal{P}$, $\mathcal{U}$ and $\mathcal{V}$ be as follows:

$$\mathcal{P}: \quad \mathsf{p}. \quad \mathsf{q}. \qquad \mathcal{U}: \quad {\sim}\mathsf{p} \leftarrow \mathsf{q}. \qquad \mathcal{V}: \quad {\sim}\mathsf{q} \leftarrow \mathsf{p}.$$

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Since $\mathcal{U}$ is strongly equivalent to $\mathcal{V}$, $(\mathsf{PU4.2})_{\mathsf{HT}}$ requires that $\mathcal{P} \oplus \mathcal{U}$ be strongly equivalent to $\mathcal{P} \oplus \mathcal{V}$. This is in contrast with the JU-, DS-, and RD-semantics where a default literal $\sim\mathsf{p}$ in the head of a rule indicates that whenever the body of the rule is satisfied, there is a *reason for* p *to cease being true*. A consequence of this is that an update of $\mathcal{P}$ by $\mathcal{U}$ results in the single answer-set $\{\mathsf{q}\}$ while an update by $\mathcal{V}$ leads to the single answer-set $\{\mathsf{p}\}$.

Thus, when considering the principle $(\mathsf{PU4.2})_{\mathsf{HT}}$, benefits of the declarativeness that it brings with it need to be weighed against the loss of control over the results of updates by rules with default literals in their heads.

6.2 Operators based on RE-models

The problems identified by Slota and Leite (2010, 2014) could be mitigated if, instead of HT-models, a richer semantic characterisation of logic programs was used. Such a characterisation would have to be able to distinguish between programs such as $\mathcal{P} = \{\mathsf{p}., \mathsf{q}.\}$ and $\mathcal{Q} = \{\mathsf{p} \leftarrow \mathsf{q}., \mathsf{q}.\}$ because they are expected to behave differently when subject to evolution. And it would have to distinguish between the rule $\sim\mathsf{p} \leftarrow \mathsf{q}.$ and the rule $\sim\mathsf{q} \leftarrow \mathsf{p}.$, inasmuch as they are expected to have different behaviour, for example, when used to update the program $\mathcal{P} = \{\mathsf{p}., \mathsf{q}.\}$.

This is precisely the approach taken by Slota and Leite (2012a), who defined a new monotonic characterisation of rules, dubbed *robust equivalence models*, or RE-models for short, which is expressive enough to distinguish between the so-called abolishing rules $\sim\mathsf{p} \leftarrow \mathsf{q}.$, $\sim\mathsf{q} \leftarrow \mathsf{p}.$, and $\leftarrow \mathsf{p}, \mathsf{q}$. Then, they introduced a generic method for specifying semantic rule update operators in which

1. a logic program is viewed as the *set of sets of RE-models of its rules*,[7] hence acknowledging rules as the atomic pieces of knowledge while, at the same time, abstracting away from unimportant differences between their syntactic forms, focusing on their semantic content;
2. updates are performed by introducing additional interpretations (*exceptions*) to the sets of RE-models of rules in the original program.

Instances of such generic framework were shown to obey properties such as *support*, *fact update*, and *causal rejection*, thus far only obeyed by syntactic approaches. Furthermore, they have a semantic characterisation that ensures several *semantic* properties such as replacement of equivalents captured by (BU4), adapted to employ this novel semantic characterisation using RE-models instead of HT-models. One such instance is also shown to provide a semantic characterisation of the JU-Semantics for DLPs without local cycles.

The RE-models and associated notions of equivalence are defined as follows.

Definition 66 (*RE-models, RE-equivalence, and RR-equivalence*)
A three-valued interpretation $\langle I, J\rangle$ is an *RE-model* of a rule π if $I \models \pi^J$. The set of all RE-models of a rule π is denoted by $[\![\pi]\!]_{\mathsf{RE}}$. Rules π, σ are *RE-equivalent* whenever

[7] This view is closely related to the view taken by *base revision operators* (Gärdenfors 1992; Hansson 1993).

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

$[\![\pi]\!]_{\mathsf{RE}} = [\![\sigma]\!]_{\mathsf{RE}}$. Programs $\mathcal{P}$ and $\mathcal{Q}$ are *RE-equivalent*, denoted by $\mathcal{P} \equiv_{\mathsf{RE}} \mathcal{Q}$, whenever $[\![\mathcal{P}]\!]_{\mathsf{RE}} = [\![\mathcal{Q}]\!]_{\mathsf{RE}}$. The set of sets of RE-models of rules inside a program $\mathcal{P}$ is denoted by $\langle\!\langle \mathcal{P} \rangle\!\rangle_{\mathsf{RE}} = \{ [\![\pi]\!]_{\mathsf{RE}} \mid \pi \in \mathcal{P} \}$, and is used as the basis for the notion of *robust rule equivalence*, or *RR-equivalence* for short. Programs $\mathcal{P}$ and $\mathcal{Q}$ and *RE-equivalent*, denoted by $\mathcal{P} \equiv_{\mathsf{RR}} \mathcal{Q}$, whenever $\langle\!\langle \mathcal{P}^\tau \rangle\!\rangle_{\mathsf{RE}} = \langle\!\langle \mathcal{Q}^\tau \rangle\!\rangle_{\mathsf{RE}}$, where, given a program $\mathcal{P}$, $\mathcal{P}^\tau = \mathcal{P} \cup \{ \tau \}$, where τ is the *canonical tautology*, that is, the rule $\mathsf{p}_\tau \leftarrow \mathsf{p}_\tau$ given a fixed atom p_τ from $\mathcal{L}$.

Interestingly, the affinity between HT-models and stable models is fully retained by RE-models: an interpretation J is a stable model of a program $\mathcal{P}$ if and only if $\langle J, J \rangle \in [\![\mathcal{P}]\!]_{\mathsf{RE}}$ and for all $I \subsetneq J$, $\langle I, J \rangle \notin [\![\mathcal{P}]\!]_{\mathsf{RE}}$. Furthermore, unlike with HT-models, any set of three-valued interpretations can be represented by a program using RE-models (Slota and Leite 2012a; Slota 2012).

The formalisation of the idea of viewing updates as introducing additional interpretations – *exceptions* – to the sets of RE-models of rules in the original program is straightforward: an exception-driven update operator is characterised by an *exception function* ε that takes three inputs: the set of RE-models $[\![\pi]\!]_{\mathsf{RE}}$ of a rule $\pi \in \mathcal{P}$ and the semantic characterisations, $\langle\!\langle \mathcal{P} \rangle\!\rangle_{\mathsf{RE}}$ and $\langle\!\langle \mathcal{U} \rangle\!\rangle_{\mathsf{RE}}$, of the original and updating programs. It then returns the three-valued interpretations that are to be introduced as exceptions to π, so the characterisation of the updated program contains the augmented set of RE-models,

$$[\![\pi]\!]_{\mathsf{RE}} \cup \varepsilon\left([\![\pi]\!]_{\mathsf{RE}}, \langle\!\langle \mathcal{P} \rangle\!\rangle_{\mathsf{RE}}, \langle\!\langle \mathcal{U} \rangle\!\rangle_{\mathsf{RE}}\right). \tag{4}$$

A rule update operator $\oplus$ is *exception-driven* if for some exception function ε, $\langle\!\langle \mathcal{P} \oplus \mathcal{U} \rangle\!\rangle_{\mathsf{RE}}$ is equal to

$$\{ [\![\pi]\!]_{\mathsf{RE}} \cup \varepsilon\left([\![\pi]\!]_{\mathsf{RE}}, \langle\!\langle \mathcal{P} \rangle\!\rangle_{\mathsf{RE}}, \langle\!\langle \mathcal{U} \rangle\!\rangle_{\mathsf{RE}}\right) \mid \pi \in \mathcal{P} \} \cup \langle\!\langle \mathcal{U} \rangle\!\rangle_{\mathsf{RE}}, \tag{5}$$

for all programs $\mathcal{P}$ and $\mathcal{U}$. In that case we also say that $\oplus$ is *ε-driven*. In words, the set of RE-models of each rule π from $\mathcal{P}$ is augmented with the respective exceptions while the sets of RE-models of rules from $\mathcal{U}$ are kept untouched. Note that since a set of three-valued interpretations may have different syntactic representations as a program using RE-models, 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 (5).[8]

Slota and Leite (2012a) further investigated a constrained class of exception functions, which they dubbed *simple exception functions*, characterised by the fact that they produce (local) 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, that is, the other rules in the programs. Formally, an exception function ε is *simple* if for all $\mathcal{M} \subseteq \mathfrak{X}$ and $\mathcal{S}, \mathcal{T} \subseteq 2^{\mathfrak{X}}$,

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

where $\delta : 2^{\mathfrak{X}} \times 2^{\mathfrak{X}} \to 2^{\mathfrak{X}}$ is a *local exception function*.

Despite their local nature, particular simple exception functions generate rule update operators that satisfy the syntactic properties of rule update semantics discussed

[8] To deal with sets of RE-models defined in (5) which do not correspond to a single rule, but rather to a set of rules, Slota and Leite (2012a) consider the so-called *rule-bases*, which can be rules or programs representing the RE-models defined in (4), and treat them as *atomic* pieces of information. Here we omit the technical aspects of this issue and, like Slota and Leite (2012a), dub them *rules*.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

before. We will recap two simple exception functions proposed by Slota and Leite (2012a), dubbed δ_{a} and δ_{b}, inspired by rule update semantics based on causal rejection, one of them closely related to the JU-semantics.

Since rule update semantics based on causal rejection make use of the concepts of *conflicting rules* (c.f. Def. 13) and *rejected rules* (c.f. Def. 14 for the case of the JU-semantics), which rely on rule syntax to which an exception function has no direct access, the semantic counterparts to these concepts were first defined.

Towards the definition of conflicting sets of RE-models, two preparatory concepts are required.

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

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

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

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

Two sets of three-valued interpretations $\mathcal{M}$, $\mathcal{N}$ are *in conflict on atom* p *w.r.t.* J, denoted by $\mathcal{M} \bowtie^J_{\mathsf{p}} \mathcal{N}$, if both $\mathcal{M}^J(\mathsf{p})$ and $\mathcal{N}^J(\mathsf{p})$ are defined and $\mathcal{M}^J(\mathsf{p}) \neq \mathcal{N}^J(\mathsf{p})$. The following example illustrates all these concepts.

Example 67
Consider rules $\pi_0 = (\mathsf{p}.)$, $\pi_1 = (\sim\mathsf{p} \leftarrow \sim\mathsf{q}.)$ with the respective sets of RE-models[9]

$$\mathcal{M}_0 = \{ (\mathsf{p}, \mathsf{p}), (\mathsf{p}, \mathsf{pq}), (\mathsf{pq}, \mathsf{pq}) \},$$
$$\mathcal{M}_1 = \{ (\emptyset, \emptyset), (\emptyset, \mathsf{q}), (\mathsf{q}, \mathsf{q}), (\emptyset, \mathsf{pq}), (\mathsf{p}, \mathsf{pq}), (\mathsf{q}, \mathsf{pq}), (\mathsf{pq}, \mathsf{pq}) \}.$$

Intuitively, $\mathcal{M}_0$ forces p to T w.r.t. all interpretations and π_1 forces p to F w.r.t. interpretations in which q is false. Formally it follows that $\mathcal{M}_0^{\emptyset}(\mathsf{p}) = \mathsf{T}$ because (p, p) belongs to $\mathcal{M}_0$ and neither $(\emptyset, \mathsf{p})$ nor $(\emptyset, \emptyset)$ belongs to $\mathcal{M}_0$. Similarly, it follows that $\mathcal{M}_1^{\emptyset}(\mathsf{p}) = \mathsf{F}$. Hence $\mathcal{M}_0 \bowtie^{\emptyset}_{\mathsf{p}} \mathcal{M}_1$. Using similar arguments we can conclude that $\mathcal{M}_0 \bowtie^{\mathsf{p}}_{\mathsf{p}} \mathcal{M}_1$. However, it does not hold that $\mathcal{M}_0 \bowtie^{\mathsf{pq}}_{\mathsf{p}} \mathcal{M}_1$ because $\mathcal{M}_1^{\mathsf{pq}}(\mathsf{p})$ is undefined.

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

Definition 68 (*Local Exception Function* δ_{a})
The local exception function δ_{a} is for all $\mathcal{M}, \mathcal{N} \subseteq \mathfrak{X}$ defined as

$$\delta_{\mathsf{a}}(\mathcal{M}, \mathcal{N}) = \{ (I, J) \in \mathfrak{X} \mid \exists \mathsf{p} : \mathcal{M} \bowtie^J_{\mathsf{p}} \mathcal{N} \}.$$

Thus if there is a conflict on some atom w.r.t. J, the exceptions introduced by δ_{a} are of the form (I, J) where I can be an arbitrary subset of J. This means that δ_{a} introduces

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

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

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 69 (*Syntactic Properties of δ_{a} Slota and Leite 2012a*)
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 cannot (cf. Theorem 64). However, these operators diverge from causal rejection, even on acyclic DLPs, when more than one update is performed.

Example 70
Consider again the rules π_0, π_1 and their sets of RE-models $\mathcal{M}_0$, $\mathcal{M}_1$ from Example 67 and some δ_{a}-driven rule update operator $\oplus$. Then, $\langle\!\langle \{\, \pi_0 \,\} \oplus \{\, \pi_1 \,\} \rangle\!\rangle_{\mathsf{RE}}$ will contain two elements: $\mathcal{M}_0'$ and $\mathcal{M}_1$, where

$$\mathcal{M}_0' = \mathcal{M}_0 \cup \delta_{\mathsf{a}}(\mathcal{M}_0, \mathcal{M}_1) = \mathcal{M}_0 \cup \{\, (\emptyset, \emptyset), (\emptyset, \mathsf{p}) \,\}\,.$$

An additional update by the fact $\{\, \mathsf{q}. \,\}$ then leads to the characterisation

$$\left\langle\!\!\left\langle \bigoplus \langle \{\, \pi_0 \,\}, \{\, \pi_1 \,\}, \{\, \mathsf{q}. \,\} \rangle \right\rangle\!\!\right\rangle_{\mathsf{RE}},$$

which contains three elements: $\mathcal{M}_0''$, $\mathcal{M}_1$ and $\mathcal{M}_2$, where

$$\mathcal{M}_0'' = \mathcal{M}_0' \cup \{\, (\emptyset, \mathsf{q}), (\mathsf{q}, \mathsf{q}) \,\}\,,$$

and $\mathcal{M}_2$ is the set of RE-models of $(\mathsf{q}.)$.

Furthermore, due to the relationship between RE-models and stable models, the interpretation $J = \{\, \mathsf{q} \,\}$ is a stable model of $\bigoplus \langle \{\, \pi_0 \,\}, \{\, \pi_1 \,\}, \{\, \mathsf{q}. \,\} \rangle$ because (q, q) belongs to all sets of models in the set of sets of models $\langle\!\langle \bigoplus \langle \{\, \pi_0 \,\}, \{\, \pi_1 \,\}, \{\, \mathsf{q}. \,\} \rangle \rangle\!\rangle_{\mathsf{RE}}$ and $(\emptyset, \mathsf{q})$ does not belong to $\mathcal{M}_2$. However, J does not respect causal rejection and it is not a JU-model of $(\{\, \pi_0 \,\}, \{\, \pi_1 \,\}, \{\, \mathsf{q}. \,\})$.

This shortcoming of δ_{a} was overcome by Slota and Leite (2012a) as follows:

Definition 71 (*Local Exception Functions δ_b*)
The local exception functions δ_{b} is for all $\mathcal{M}, \mathcal{N} \subseteq \mathcal{X}$ defined as

$$\delta_{\mathsf{b}}(\mathcal{M}, \mathcal{N}) = \left\{\, (I, K) \in \mathcal{X} \mid \exists J\, \exists \mathsf{p} : \mathcal{M} \bowtie^J_{\mathsf{p}} \mathcal{N} \wedge I \subseteq J \subseteq K \wedge (\mathsf{p} \in K \setminus I \implies K = J) \,\right\}.$$

The function δ_{b} introduces 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. This leads to operators that satisfy all syntactic properties.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Theorem 72 (*Syntactic Properties of δ_b Slota and Leite 2012a*)
Let $\oplus$ be a δ_b-driven rule update operator. Then $\oplus$ respects support, language conservation, fact update, causal rejection, acyclic justified update, immunity to tautological and empty updates, and primacy of new information.

Under RR equivalence, the postulate that requires update operators to be syntax indepent can be defined as follows, for a rule update operator $\oplus$ and programs $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{U}$, and $\mathcal{V}$:

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

Theorem 73 (*Syntax Independence of δ_b Slota and Leite 2012a*)
Let $\oplus$ be a δ_b-driven rule update operator. Then, $\oplus$ respects $(\mathsf{PU4})_{\mathsf{RR}}$.

It is worth noting that δ_b-driven operators are very closely related to the JU-semantics, even on programs with cycles. They diverge from it only on programs with tautologies.

Theorem 74 (*Slota and Leite 2012a*)
Let $\boldsymbol{P}$ be a DLP, J an interpretation and $\oplus$ a δ_b-driven rule update operator. Then,

- $[\![\boldsymbol{P}]\!]_{\mathsf{S}_\oplus} \subseteq [\![\boldsymbol{P}]\!]_{\mathsf{JU}}$ and
- if $\mathsf{all}(\boldsymbol{P})$ contains no tautologies, then $[\![\boldsymbol{P}]\!]_{\mathsf{JU}} \subseteq [\![\boldsymbol{P}]\!]_{\mathsf{S}_\oplus}$.

This means that up to the case of tautologies, δ_b can be seen as semantic characterisation of the justified update semantics: it leads to stable models that, typically, coincide with justified update models.

Example 75
Consider again the program $\mathcal{P}$ from the example in the introduction, which contains the rules

$$\mathsf{goHome} \leftarrow {\sim}\mathsf{money}. \qquad \mathsf{goRestaurant} \leftarrow \mathsf{money}. \qquad \mathsf{money}.$$

and its update $\mathcal{U}$ with the rules

$${\sim}\mathsf{money} \leftarrow \mathsf{robbed}. \qquad \mathsf{robbed}.$$

Unsurprisingly, δ_b would only introduce exceptions to the third rule of $\mathcal{P}$. A δ_b-driven rule update operator $\oplus$ would produce, as the result of $\mathcal{P} \oplus \mathcal{U}$, a program RR-equivalent to:

$$\begin{aligned} \mathsf{goHome} &\leftarrow {\sim}\mathsf{money}.\\ \mathsf{goRestaurant} &\leftarrow \mathsf{money}.\\ \mathsf{money} &\leftarrow {\sim}\mathsf{robbed}.\\ {\sim}\mathsf{money} &\leftarrow \mathsf{robbed}.\\ \mathsf{robbed}.& \end{aligned}$$

Example 76
Going back to Example 65 where the programs $\mathcal{P}$, $\mathcal{U}$, and $\mathcal{V}$ were as follows:

$$\mathcal{P}: \quad \mathsf{p}. \quad \mathsf{q}. \qquad \mathcal{U}: \quad {\sim}\mathsf{p} \leftarrow \mathsf{q}. \qquad \mathcal{V}: \quad {\sim}\mathsf{q} \leftarrow \mathsf{p}.$$

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

A δ_{b}-driven rule update operator $\oplus$ would produce, up to RR-equivalence, the following results, which respect both *support* and *fact update*:

$$\mathcal{P} \oplus \mathcal{U}: \quad \begin{array}{l} \mathsf{p} \leftarrow {\sim}\mathsf{q}. \\ \mathsf{q}. \\ {\sim}\mathsf{p} \leftarrow \mathsf{q}. \end{array} \qquad \mathcal{P} \oplus \mathcal{V}: \quad \begin{array}{l} \mathsf{p}. \\ \mathsf{q} \leftarrow {\sim}\mathsf{p} \\ {\sim}\mathsf{q} \leftarrow \mathsf{p}. \end{array}$$

The new monotonic characterisation of rules – RE-models – and the 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, allowed the definition of concrete update operators that enjoy a combination of syntactic as well as semantic properties that had never been reconciled before.

Acknowledging rules as first-class objects by viewing a program as the set of sets of their models essentially amounts to adopting the view used by *base revision operators* (Gärdenfors 1992; Hansson 1993) where a theory is composed of a set of formulas, each considered an atomic piece of knowledge that could be falsified by an update. This view has also been more recently adopted within the context of description logic updates (Liu *et al*. 2006; De Giacomo *et al*. 2009; Calvanese *et al*. 2010; Lenzerini and Savo 2011).

Nevertheless, departing from HT-models and using RE-models instead may raise a few eyebrows. After all, characterising strong equivalence between programs through HT-models is one of the great landmarks in the history of answer-set programming. However, strong equivalence characterised by HT-models only considers the union of rules, while other kinds of operations must also be considered when dealing with belief change such as *updates* and *revisions*, for example, falsifying an atom that was previously true. Just as the rule $\mathsf{p} \leftarrow \mathsf{q}.$ specifies q as a justification for p, the rule ${\sim}\mathsf{p} \leftarrow \mathsf{q}.$ should be seen as specifying q as a justification for ${\sim}\mathsf{p}$ and the rule ${\sim}\mathsf{q} \leftarrow \mathsf{p}.$ as specifying p as a justification for ${\sim}\mathsf{q}$. Whereas the latter two rules are HT-equivalent – each can be modularly replaced by the other, the remaining rules staying the same – they provide different justifications in a dynamic setting, for example, when used to update the program $\mathcal{P} = \{\, \mathsf{p}., \mathsf{q} \,\}$: the former provides a justification to make p false while the latter a justification to make q false. Distinguishing between these two rules, and between them and the integrity constraint $\leftarrow \mathsf{p}, \mathsf{q}.$, is precisely the difference between HT-models and RE-models.

Other properties that are obeyed by a δ_{b}-driven rule update operator $\oplus$ include, for example, *initialisation* ($\emptyset \oplus \mathcal{U} \equiv \mathcal{U}$), *non-interference* (If $\mathcal{U}$, $\mathcal{V}$ are over disjoint alphabets, then $(\mathcal{P} \oplus \mathcal{U}) \oplus \mathcal{V} \equiv (\mathcal{P} \oplus \mathcal{V}) \oplus \mathcal{U}$), *absorption* ($(\mathcal{P} \oplus \mathcal{U}) \oplus \mathcal{U} \equiv \mathcal{P} \oplus \mathcal{U}$), *augmentation* (If $\mathcal{U} \subseteq \mathcal{V}$, then $(\mathcal{P} \oplus \mathcal{U}) \oplus \mathcal{V} \equiv \mathcal{P} \oplus \mathcal{V}$), among others. A more thorough discussion on δ_{b}-driven rule update operators' properties as well as on the reasons why they fail other semantic properties drawn from the KM postulates are given by Slota and Leite (2012a).

Despite this success, a closer inspection shows that RR-equivalence might still be slightly too strong for characterising updates, because programs such as $\{\, \mathsf{p}. \,\}$ and $\{\, \mathsf{p}., \mathsf{p} \leftarrow \mathsf{q}. \,\}$ are not considered RR-equivalent even though we expect the same behaviour from them when they are updated. A notion of program equivalence that is weaker than RR-equivalence but stronger than RE-equivalence so that both (BU4) and properties such as $\mathcal{P} \cup \mathcal{U} \models \mathcal{P} \oplus \mathcal{U}$ can be achieved under a single notion of program equivalence still

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

needs to be found. Slota and Leite (2011) speculate that this could be solved by adopting a weaker equivalence – SMR-equivalence – which discards rules whose set of models are supersets of the set of models of other rules. However, such equivalence is too weak since, when instantiated with RE-models, 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.

7 State of affairs – comments and outlook

McCarthy (1998) described *elaboration tolerance* as "*the ability to accept changes to a person's or computer program's representation of facts about a subject without having to start all over*", arguing that human-level AI will require representation formalisms that are elaboration tolerant. Representing knowledge as a DLP – a sequence of logic programs – equipped with automated mechanisms to deal with overlapping, possibly conflicting information, as those provided by the rule update semantics surveyed in this paper, constitutes one significant step towards equipping logic programming with elaboration tolerance. It allows for the kind of incremental specification proposed by McCarthy where one can focus on specifying what is new or what has changed, adding it to the end of the sequence, without having to rework a new encompassing representation.

One of the most important lessons learnt so far is that the syntactic nature of answer-set programming cannot be ignored when programs are subjected to a belief change operation such as an update, or even a revision. Whereas this was at the heart of the rejection of model-based updates, which led to the myriad of approaches developed during the second era – syntax-based updates – it was partially forgotten at the beginning of the third era – semantics-based updates – with the promise that characterising a program through its HT-models would move us away from having to rely on the program's syntax. Theorem 64 killed this hope by showing that such a characterisation of programs based on HT-models simply cannot be used if fundamental properties such as *support* and *fact update* are required. And these are indeed two fundamental properties. *Fact update* is a very straightforward, simple and rather undisputed property, which corresponds to the simple unconditional change operations – insertion and deletion – performed on extensional (relational) databases, for example, corresponding to the SQL commands `INSERT/DELETE fact FROM P`. The second property, *support*, is related to the fact that implication in rules plays a significantly different role than material implication in classical logic, for example, not allowing for the contrapositive, instead being tied to the intuitionistic logic of here-and-there (Heyting 1930; Pearce 1997) where the notion of truth is tied to the notion of proof/justification, somehow provided by rules from their bodies to their heads. Accordingly, unsupported atoms are usually not accepted in logic programming.

The negative result encoded by Theorem 64 should certainly make us look back to the so-called syntactical approaches through a different set of lenses, less critical of the fact that its syntactic features play a central role, and see whether they indeed provide a viable alternative, such as the RD-semantics, which is the only one immune to tautological updates while maintaining all other syntactical properties discussed here. But at the same time, Theorem 64 should not drive us away from the pursuit of a semantical characterisation of updates that somehow reconciles the syntactical properties such as

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

support, *fact update*, and *causal rejection* with the semantic properties encoded in both the AGM and KM postulates.

Acknowledging the importance of these syntactical properties, Binnewies *et al.* (2018) also adopted HT-models as the underlying semantic characterisation of logic programs to introduce partial meet and ensconcement constructions for logic program belief change, which allowed them to define syntax-preserving operators that obey *support*.

The introduction of the *abstract exception-driven update* abstract framework (Slota and Leite 2012b) seems to be another important step in the direction of reconciling the syntactical nature of logic programming with the semantic properties encoded in the AGM family of postulates. On the one hand, it served as a framework to capture updates of logic programs, namely the JU-semantics, reconciling semantical as well as syntactical properties. It did so by using RE-models to characterise rules, and viewing a program as the *set of sets of the RE-models of its rules*. This way, it acknowledged rules as the atomic pieces of knowledge while, at the same time, abstracting away from unimportant differences between their syntactic forms, focusing on their semantic content. On the other hand, this abstract framework was shown to also capture several update operators used for ontology updates, such as the model-based Winslett's operator, or the formula-based WIDTIO and *bold* operators (Liu *et al.* 2006; De Giacomo *et al.* 2009; Calvanese *et al.* 2010; Lenzerini and Savo 2011).

The *exception-driven update* framework also seems to provide a promising vehicle to reconcile updates in classical logic with updates of logic programs, opening up a promising avenue to investigate updates of hybrid knowledge bases composed of rules and ontologies (Slota 2012; Slota *et al.* 2015). Additionally, the logic programming instantiation of the exception-driven update framework shed new light into the long lasting problem of *state condensing*,[10] which was solved by Slota and Leite (2013) for the JU and AS semantics, by resorting to more expressive classes of answer-set programs, namely nested and disjunctive.

The historical account of the research on updating logic programs found in this paper is almost entirely focused on the declarative side of the problem, leaving out most procedural and computational aspects. Our choice to follow this path was somehow grounded on the fact that over the years the procedural and computational aspects have played a secondary role in this line of research. As far as we know, there are currently no efficient native implementations available to compute the S-models for any of the semantics S presented in this paper. The absence of efficient native implementations is more important given the known negative results regarding *state condensing*, which in general prevent the direct usage of existing efficient solvers, since it is in general not possible to construct a single logic program that corresponds to the result of the updates encoded in a DLP, written in the same language $\mathcal{L}$ as the DLP. Several authors have addressed the procedural and computational aspects of updating logic programs through the definition of general transformations that convert any DLP into a single logic program – written in a language that extends to the original language $\mathcal{L}$ with new propositional symbols to allow the representation of the *temporal* aspect of the DLP, and other notions such

[10] *State condensing* is the problem of finding a single logic program that faithfully represents a sequence of logic programs (DLP), that is, that (i) is written in the same alphabet, (ii) has the same set of stable models, and (iii) is equivalent to the sequence of programs when subject to further updates.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

as the rejection of rules – whose stable models, restricted to the original language $\mathcal{L}$, correspond to the S-models according to the rule update semantics S in question. Each of the semantics based on causal rejection has one such corresponding transformation. With these transformations, the S-models of the DLP can be computed through the resulting logic program using existing efficient solvers such as *clasp* (Gebser *et al.* 2011) and *dlv* (Leone *et al.* 2006), even though the syntactical overhead introduced by the transformations could result in a significant computational cost. Delgrande *et al.* (2007), on the other hand, define their semantics through a transformation into a prioritised logic program, which doesn't need the extension of the language $\mathcal{L}$, but for which there are no efficient native implementations, as far as we know. Hence, the resulting prioritised logic program would have to be further converted into a logic program to enable the use of existing efficient solvers, with a similar syntactical overhead as the transformations mentioned above. Other semantics either do not mention procedural and computational aspects, or provide transformations that only deal with DLPs of length two, whose result cannot be further updated, as discussed earlier in this paper. All in all, whereas the known results regarding the procedural and computational aspects of many of the semantics presented in this paper are enough to allow the development of prototypical implementations – some of which have actually been created by the authors of the original semantics, even if no longer easily available – more research is needed to develop native efficient implementations that could support large applications.

Despite the progress towards a full understanding of updates of logic programs, many open questions remain, whose difficulty in answering seems to always be tied to the tension created by the syntactical nature of logic programs, and the wish for a semantical characterisation of belief change operations. We discuss some of the most important ones below.

Can other syntactical rule update semantics be captured by the exception-driven update *framework?* The need to detect non-tautological irrelevant updates (Alferes *et al.* 2005; Šefránek 2006, 2011) to capture, for example, the RD-semantics poses a number of challenges. For instance, simple exception functions based on local exceptions such as δ_{c} cannot distinguish an update of $\{\,\mathsf{p}.\,\}$ by $\mathcal{U} = \{\,{\sim}\mathsf{p} \leftarrow {\sim}\mathsf{q}.\,, {\sim}\mathsf{q} \leftarrow {\sim}\mathsf{p}.\,\}$, where it is plausible to introduce the exception $\langle \emptyset, \emptyset \rangle$, from an update of $\{\,\mathsf{p}.\,, \mathsf{q}.\,\}$ by $\mathcal{U}$, 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. Such functions would also have the potential of satisfying properties such as (BU3) and associativity $(\mathcal{P} \oplus (\mathcal{U} \oplus \mathcal{V}) \equiv (\mathcal{P} \oplus \mathcal{U}) \oplus \mathcal{V})$.

Can we characterise rule update semantics through binary operators on some class of programs over the same alphabet so that they can be iterated, hence addressing the state condensing problem? Whereas answering the previous question on this list might certainly provide hints on how to characterise rule update semantics through binary operators, it may well be a problem without solution for some of the existing semantics, even resorting to more expressive classes of logic programs. This is probably the case with the PRZ and the RVS semantics because of how they are defined, which involves the construction of possibly more than one program, using maximality criteria. Given its properties and relationship with the JU and AS semantics, for which a solution already exists, perhaps the bigger reward lies in addressing this question for the RD semantics.

What is the appropriate logic corresponding to the RE-models? Just as the logic of here-and-there (Heyting 1930; Pearce 1997), a nonclassical logic extending intuitionistic

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

logic, served as a basis for a logical characterisation of answer-set programming, finding the nonclassical logic that subsumes RE-models and provides an adequate basis to characterise the dynamics of logic programs would be most useful. The difficulties in extending even RE-models to more general classes of logic programs hints on the difficulty of this task.

Can a characterisation based on RE-models bring added value when used to describe logic programs that undergo other belief change operations such as revision, contraction, erasure, forgetting and ensconcement? Ultimately, the answer to this question boils down to whether the rule $\sim p \leftarrow q.$, the rule $\sim q \leftarrow p.$, and the integrity constraint $\leftarrow p, q.$ should be considered equivalent, or not, within the belief change operation in question. In the context of updates of logic programs, as we have seen throughout this paper, some semantics distinguish them, as for example those based on causal rejection, where updating the program $\{ p., q. \}$ with $\sim p \leftarrow q.$ results in the model $\{ q \}$ while updating it with $\sim q \leftarrow p.$ would result in the model $\{ p \}$. If this distinction makes sense in other belief change operations – for example, one might conceive a revision operator with a similar behaviour – then a characterisation based on RE-models might be more suitable than using, for example, HT-models.

How can the syntax-based updates be extended to deal with disjunctive logic programs? Interestingly, updating disjunctive logic programs under the answer-set semantics has received very little attention so far. Whereas extending some of the syntax-based approaches seems straightforward – in fact, the RVS-semantics was defined by Sakama and Inoue (2003) for disjunctive logic programs – dealing with semantics such as those based on the causal rejection principle might prove more difficult. For example, conflicts that provide a cause for rejecting rules would no longer be restricted to pairs of rules.

How to precisely characterise the related though distinct belief change operations of revision and update within the context of answer-set programming? At an abstract level, the difference between these two forms of belief change rests on whether we are acquiring better information about a static world (revision) or whether we are acquiring newer information about a changing world (update). The influence of such difference when dealing with belief change in answer-set programming has been limited and, to some extent, restricted to the extent to which one seeks to deal with inconsistencies/contradictions: whereas in revision they are removed at any cost, even as the result of an empty/tautological belief change operation, in updates such an empty/tautological operation should have no effect. More is needed in order to better understand and characterise what revision and update have in common, and in difference, when taken within the context of answer-set programming.

Does it make sense to consider other causes for rejection within causal-rejection based semantics? The answer to this question is, to some extent, related to the previous question, inasmuch as it relates to finding other conflicts that, if not resolved, could result in a contradiction/incoherence which should be avoided in the context of an update. But other avenues for extending the causes for rejection exist such as, for example, explicitly declaring certain pairs of otherwise unrelated atoms as being in conflict.

All in all, the quest for the *perfect* approach to updating answer-set programs has been quite fruitful. We are not there yet – and we may never be – but we are certainly much closer.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

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., 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.

Alferes, J. J., Brogi, A., Leite, J. A. and Pereira, L. M. 2003. An evolvable rule-based e-mail agent. In *Proceedings of the 11th Portuguese Conference Artificial Intelligence (EPIA 2003)*, F. Moura-Pires and S. Abreu, Eds. Lecture Notes in Computer Science, vol. 2902. Springer, Beja, Portugal, 394–408.

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. and Pereira, L. M. 1996. Update-programs can update programs. In *Proceedings of the 6th Workshop on Non-Monotonic Extensions of Logic Programming (NMELP 1996)*, J. Dix, L. M. Pereira and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer, Bad Honnef, Germany, 110–131.

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*, J. Minker, Ed. Morgan Kaufmann, San Francisco, CA, USA, 89–148.

Banti, F., Alferes, J. J., Brogi, A. and Hitzler, P. 2005. The well supported semantics for multidimensional dynamic logic programs. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2005)*, C. Baral, G. Greco, N. Leone and G. Terracina, Eds. Lecture Notes in Computer Science, vol. 3662. Springer, Diamante, Italy, 356–368.

Binnewies, S., Zhuang, Z., Wang, K. and Stantic, B. 2018. Syntax-preserving belief change operators for logic programs. *ACM Transactions on Computational Logic 19,* 2, 12:1–12:42.

Brewka, G. and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence 109,* 1-2, 297–356.

Buccafurri, F., Faber, W. and Leone, N. 1999. Disjunctive logic programs with inheritance. In *Proceedings of the 1999 International Conference on Logic Programming (ICLP 1999)*, D. D. Schreye, Ed. The MIT Press, Las Cruces, New Mexico, USA, 79–93.

Cabalar, P. and Ferraris, P. 2007. Propositional theories are strongly equivalent to logic programs. *Theory and Practice of Logic Programming 7,* 6, 745–759.

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

Calvanese, D., Kharlamov, E., Nutt, W. and Zheleznyakov, D. 2010. Evolution of DL-Lite knowledge bases. In *Proceedings of the 9th International Semantic Web Conference (ISWC 2009)*, P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks and B. Glimm, Eds. Lecture Notes in Computer Science, vol. 6496. Springer, Shanghai, China, 112–128.

Dalal, M. 1988. Investigations into a theory of knowledge base revision. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988)*, H. E. Shrobe, T. M. Mitchell and R. G. Smith, Eds. AAAI Press/The MIT Press, St. Paul, MN, USA, 475–479.

De Giacomo, G., Lenzerini, M., Poggi, A. and Rosati, R. 2009. On instance-level update and erasure in description logic ontologies. *Journal of Logic and Computation 19,* 5, 745–770.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Delgrande, J. P. 2010. A program-level approach to revising logic programs under the answer set semantics. *Theory and Practice of Logic Programming 10,* 4-6, 565–580.

Delgrande, J. P., Peppas, P. and Woltran, S. 2013. Agm-style belief revision of logic programs under answer set semantics. In *Proceeding of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 264–276.

Delgrande, J. P., Schaub, T. and Tompits, H. 2003. A framework for compiling preferences in logic programs. *Theory and Practice of Logic Programming 3,* 2, 129–187.

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

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

Delgrande, J. P., Schaub, T., Tompits, H. and Woltran, S. 2013. A model-theoretic approach to belief change in answer set programming. *ACM Transactions on Computational Logic 14,* 2, 14.

Delgrande, J. P. and Wang, K. 2015. A syntax-independent approach to forgetting in disjunctive logic programs. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, B. Bonet and S. Koenig, Eds. AAAI Press, 1482–1488.

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

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 2,* 6, 721–777.

Eiter, T., Gottlob, G. and Mannila, H. 1997. Disjunctive datalog. *ACM Transactions on Database Systems 22,* 3, 364–418.

Erdem, E., Gelfond, M. and Leone, N. 2016. Applications of answer set programming. *AI Magazine 37,* 3, 53–68.

Erdem, E. and Patoglu, V. 2018. Applications of ASP in robotics. *Künstliche Intelligenz 32,* 2-3, 143–149.

Fages, F. 1994. Consistency of clark's completion and existence of stable models. *Methods of Logic in Computer Science 1,* 1, 51–60.

Falkner, A. A., Friedrich, G., Schekotihin, K., Taupe, R. and Teppan, E. C. 2018. Industrial applications of answer set programming. *Künstliche Intelligenz 32,* 2-3, 165–176.

Fermé, E. L. and Hansson, S. O. 2011. AGM 25 years - twenty-five years of research in belief change. *Journal of Philosophical Logic 40,* 2, 295–331.

Garcia, L., Lefèvre, C., Stéphan, I., Papini, O. and Würbel, E. 2019. A semantic characterization ASP base revision. *Journal of Artificial Intelligence Research 66*, 989–1029.

Gärdenfors, P. 1992. *Belief Revision.* Cambridge University Press, Chapter Belief Revision: An Introduction, 1–28.

Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T. and Schneider, M. T. 2011. Potassco: The potsdam answer set solving collection. *AI Communincations 24,* 2, 107–124.

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

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

Gonçalves, R., Knorr, M. and Leite, J. 2016. The ultimate guide to forgetting in answer-set programming. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, C. Baral, J. Delgrande and F. Wolter, Eds. AAAI Press.

Hansson, S. O. 1993. Reversing the Levi identity. *Journal of Philosophical Logic 22,* 6, 637–669.

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.

Hitzler, P. and Wendt, M. 2005. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming 5,* 1-2, 93–121.

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

Ilic, M., Leite, J. and Slota, M. 2008. Explicit dynamic user profiles for a collaborative filtering recommender system. In *Proceedings of the 11th Ibero-American Conference on Artificial Intelligence (IBERAMIA 2008)*, H. Geffner, R. Prada, I. M. Alexandre and N. David, Eds. LNAI, vol. 5290. Springer-Verlag, 352–361.

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

Katsuno, H. and Mendelzon, A. O. 1989. A unified view of propositional knowledge base updates. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI 1989)*, N. S. Sridharan, Ed. Morgan Kaufmann, 1413–1419.

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

Keller, A. M. and Winslett, M. 1985. On the use of an extended relational model to handle changing incomplete information. *IEEE Transactions on Software Engineering 11,* 7, 620–633.

Krümpelmann, P. 2012. Dependency semantics for sequences of extended logic programs. *Logic Journal of the IGPL 20,* 5, 943–966.

Krümpelmann, P. and Kern-Isberner, G. 2010. On belief dynamics of dependency relations for extended logic programs. In *Proceedings of the 13th International Workshop on Non-Monotonic Reasoning (NMR 2010)*, T. Meyer and E. Ternovska, Eds. Toronto, Canada.

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

Leite, J. A. and Pereira, L. M. 1998. Generalizing updates: From models to programs. In *Proceedings of the 3rd International Workshop on Logic Programming and Knowledge Representation (LPKR 1997)*, J. Dix, L. M. Pereira and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1471. Springer, 224–246.

Lenzerini, M. and Savo, D. F. 2011. On the evolution of the instance level of DL-Lite knowledge bases. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, R. Rosati, S. Rudolph and M. Zakharyaschev, Eds. CEUR Workshop Proceedings, vol. 745. CEUR-WS.org, Barcelona, Spain.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S. and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic 7,* 3, 499–562.

Lifschitz, V. 1999. Action languages, answer sets, and planning. In *The Logic Programming Paradigm: A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczynski and D. S. Warren, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 357–373.

Lifschitz, V. 2008. Twelve definitions of a stable model. In *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008)*, M. G. de la Banda and E. Pontelli, Eds. Lecture Notes in Computer Science, vol. 5366. Springer, 37–51.

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

Liu, H., Lutz, C., Miličić, M. and Wolter, F. 2006. Updating description logic ABoxes. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, P. Doherty, J. Mylopoulos and C. A. Welty, Eds. AAAI Press, Lake District of the United Kingdom, 46–56.

Łukasiewicz, J. 1941. Die Logik und das Grundlagenproblem. In *Les Entretiens de Zürich sue les Fondements et la méthode des sciences mathématiques 1938.* Zürich, 82–100.

Marek, V. W. and Truszczynski, M. 1994. Revision specifications by means of programs. In *Proceedings of the 4th European Workshop on Logics in Artificial Intelligence (JELIA 2094)*, C. MacNish, D. Pearce and L. M. Pereira, Eds. Lecture Notes in Computer Science, vol. 838. Springer, 122–136.

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

Marek, V. W. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, K. R. Apt, V. W. Marek, M. Truszczynski and D. S. Warren, Eds. Springer Berlin Heidelberg, Berlin, Heidelberg, 375–398.

McCarthy, J. 1998. Elaboration tolerance. In *Working Papers of the Fourth International Symposium on Logical formalizations of Commonsense Reasoning (Commonsense 1998).*

Newtono, I. 1726. *Philosophiæ Naturalis Principia Mathematica.* Editio tertia & aucta emendata. Apud Guil & Joh. Innys, Regiæ Societatis typographos.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

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.

Osorio, M. and Zepeda, C. 2007. Update sequences based on minimal generalized pstable models. In *Proceedings of the 6th Mexican International Conference on Artificial Intelligence (MICAI 2007)*, A. F. Gelbukh and A. F. K. Morales, Eds. Lecture Notes in Computer Science, vol. 4827. Springer, Aguascalientes, Mexico, 283–293.

Pearce, D. 1997. A new logical characterisation of stable models and answer sets. In *Proceedings of the 6th Workshop on Non-Monotonic Extensions of Logic Programming (NMELP 1996)*, J. Dix, L. M. Pereira and T. C. Przymusinski, Eds. Lecture Notes in Computer Science, vol. 1216. Springer, Bad Honnef, Germany, 57–70.

Przymusinski, T. C. and Turner, H. 1995. Update by means of inference rules. In *Proceedings of the 3rd International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 1995)*, V. W. Marek and A. Nerode, Eds. Lecture Notes in Computer Science, vol. 928. Springer, 156–174.

Przymusinski, T. C. and Turner, H. 1997. Update by means of inference rules. *The Journal of Logic Programming 30,* 2, 125–143.

Saias, J. and Quaresma, P. 2004. A methodology to create legal ontologies in a logic programming based web information retrieval system. *Artificial Intelligence and Law 12,* 4, 397–417.

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

Satoh, K. 1988. Nonmonotonic reasoning by minimal belief revision. In *Proceedings of the International Conference on Fifth Generation Computer Systems, (FGCS 1988)*. OHMSHA Ltd. Tokyo and Springer-Verlag, 455–462.

Schaub, T. and Wang, K. 2003. A semantic framework for preference handling in answer set programming. *Theory and Practice of Logic Programming 3,* 4-5, 569–607.

Schwind, N. and Inoue, K. 2016. Characterization of logic program revision as an extension of propositional revision. *Theory and Practice of Logic Programming 16,* 1, 111–138.

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

Šefránek, J. 2011. Static and dynamic semantics: Preliminary report. In *Proceedings of the 10th Mexican International Conference on Artificial Intelligence (MICAI 2011)*, I. Z. Batyrshin and G. Sidorov, Eds. IEEE Computer Society, Los Alamitos, CA, USA, 36–42.

Siska, J. 2006. Dynamic logic programming and world state evaluation in computer games. In *Proceedings of the 20th Workshop on Logic Programming (WLP 2006)*, M. Fink, H. Tompits and S. Woltran, Eds. INFSYS Research Report, vol. 1843-06-02. Technische Universität Wien, Austria, Vienna, Austria, 64–70.

Slota, M. 2012. *Updates of Hybrid Knowledge Bases*. Ph.D. thesis, Universidade Nova de Lisboa.

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

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

Slota, M. and Leite, J. 2012a. Robust equivalence models for semantic updates of answer-set programs. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, G. Brewka, T. Eiter and S. A. McIlraith, Eds. AAAI Press, Rome, Italy, 158–168.

Slota, M. and Leite, J. 2012b. A unifying perspective on knowledge updates. In *Proceedings of the 13th European Conference on Logics in Artificial Intelligence (JELIA 2012)*, L. F. del Cerro, A. Herzig and J. Mengin, Eds. Logics in Artificial Intelligence (LNAI), vol. 7519. Springer, Toulouse, France, 372–384.

Slota, M. and Leite, J. 2013. On condensing a sequence of updates in answer-set programming. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, F. Rossi, Ed. IJCAI/AAAI, 1097–1103.

Slota, M. and Leite, J. 2014. The rise and fall of semantic rule updates based on SE-models. *Theory and Practice of Logic Programming 14,* 6, 869–907.

Slota, M., Leite, J. and Swift, T. 2011. Splitting and updating hybrid knowledge bases. *Theory and Practice of Logic Programming 11,* 4-5, 801–819.

Slota, M., Leite, J. and Swift, T. 2015. On updates of hybrid knowledge bases composed of ontologies and rules. *Artificial Intelligence 229*, 33–104.

Wang, Y., Wang, K. and Zhang, M. 2013. Forgetting for answer set programs revisited. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, F. Rossi, Ed. IJCAI/AAAI.

https://doi.org/10.1017/S1471068422000060 Published online by Cambridge University Press

Wang, Y., Zhang, Y., Zhou, Y. and Zhang, M. 2012. Forgetting in logic programs under strong equivalence. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, G. Brewka, T. Eiter and S. A. McIlraith, Eds. AAAI Press, 643–647.

Wang, Y., Zhang, Y., Zhou, Y. and Zhang, M. 2014. Knowledge forgetting in answer set programming. *Journal of Artificial Intelligence Research 50*, 31–70.

Winslett, M. 1988. Reasoning about action using a possible models approach. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 1988)*, H. E. Shrobe, T. M. Mitchell and R. G. Smith, Eds. AAAI Press/The MIT Press, Saint Paul, MN, USA, 89–93.

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

Zhang, Y. 2003. Two results for prioritized logic programming. *Theory and Practice of Logic Programming 3,* 2, 223–242.

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

Zhang, Y. and Foo, N. Y. 1998. Updating logic programs. In *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI 1998)*, H. Prade, Ed. John Wiley and Sons, Chichester, Brighton, UK, 403–407.

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

Zhuang, Z., Delgrande, J. P., Nayak, A. C. and Sattar, A. 2016. Reconsidering agm-style belief revision in the context of logic programs. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, G. A. Kaminka, M. Fox, P. Bouquet, E. Hüllermeier, V. Dignum, F. Dignum and F. van Harmelen, Eds. Frontiers in Artificial Intelligence and Applications, vol. 285. IOS Press, 671–679.