Back and Forth Between Rules and SE-Models*

Martin Slota and João Leite

CENTRIA & Departamento de Informática
Universidade Nova de Lisboa
Quinta da Torre
2829-516 Caparica, Portugal

Abstract. Rules in logic programming encode information about mutual interdependencies between literals that is not captured by any of the commonly used semantics. This information becomes essential as soon as a program needs to be modified or further manipulated.
We argue that, in these cases, a program should not be viewed solely as the set of its models. Instead, it should be viewed and manipulated as the *set of sets of models* of each rule inside it. With this in mind, we investigate and highlight relations between the SE-model semantics and individual rules. We identify a set of representatives of rule equivalence classes induced by SE-models, and so pinpoint the exact expressivity of this semantics with respect to a single rule. We also characterise the class of sets of SE-interpretations representable by a single rule. Finally, we discuss the introduction of two notions of equivalence, both stronger than *strong equivalence* [1] and weaker than *strong update equivalence* [2], which seem more suitable whenever the dependency information found in rules is of interest.

1 Motivation

In this paper we take a closer look at the relationship between the SE-model semantics and individual rules of a logic program. We identify a set of representatives of rule equivalence classes, which we dub *canonical rules*, characterise the class of sets of SE-interpretations that are representable by a single rule, and show how the corresponding canonical rules can be reconstructed from them. We believe that these results pave the way to view and manipulate a logic program as the *set of sets of SE-models* of each rule inside it. This is important in situations when the set of SE-models of the whole program fails to capture essential information encoded in individual rules inside it, such as when the program needs to be modified or further manipulated. With this in mind, we briefly discuss two new notions of equivalence, stronger than *strong equivalence* [1] and weaker than *strong update equivalence* [2].

In many extensions of Answer-Set Programming, individual rules of a program are treated as first-class citizens – apart from their prime role of encoding the answer sets assigned to the program, they carry essential information about mutual interdependencies between literals that cannot be captured by answer sets. Examples that enjoy these

* An extended version of this paper with all the proofs is available at `http://arxiv.org/abs/1102.5385`

characteristics include the numerous approaches that deal with dynamics of logic programs, where inconsistencies between older and newer knowledge need to be resolved by "sacrificing" parts of an older program (such as in [3–11]). These approaches look at subsets of logic programs in search of plausible conflict resolutions. Some of them go even further and consider particular literals in heads and bodies of rules in order to identify conflicts and find ways to solve them. This often leads to definitions of new notions which are *too* syntax-dependent. At the same time, however, semantic properties of the very same notions need to be analysed, and their syntactic basis then frequently turns into a serious impediment.

Arguably, a *more* syntax-independent method for this kind of operations would be desirable. Not only would it be theoretically more appealing, but it would also allow for a better understanding of its properties with respect to the underlying semantics. Moreover, such a more semantic approach could facilitate the establishment of bridges with the area of Belief Change (see [12] for an introduction), and benefit from the many years of research where semantic change operations on monotonic logics have been studied, desirable properties for such operations have been identified, and constructive definitions of operators satisfying these properties have been introduced.

However, as has repeatedly been argued in the literature [4, 13], fully semantic methods do not seem to be appropriate for the task at hand. Though their definition and analysis is technically possible and leads to very elegant and seemingly desirable properties, there are a number of simple examples for which these methods fail to provide results that would be in line with basic intuitions [4]. Also, as shown in [13], these individual problems follow a certain pattern: intuitively, any purely semantic approach to logic program updates satisfying a few very straightforward and desirable properties cannot comply with the property of *support* [14, 15], which lies at the very heart of semantics for Logic Programs. This can be demonstrated on simple programs $\mathcal{P} = \{\, p., q. \,\}$ and $\mathcal{Q} = \{\, p., q \leftarrow p. \,\}$ which are *strongly equivalent*, thus indistinguishable from the semantic perspective, but while $\mathcal{P}$ does not contain any dependencies, $\mathcal{Q}$ introduces a dependence of atom q upon atom p. This has far-reaching consequences, at least with respect to important notions from the logic programming point of view, such as that of *support*, which are themselves defined in syntactic rather than semantic terms. For example, if we change our beliefs about p, and come to believe that it is false, we may expect different beliefs regarding q, depending on whether we start form $\mathcal{P}$, in which case q would still be true, or $\mathcal{Q}$, in which case q would no longer be true because it is no longer supported.

We believe that rules indeed contain information that, to the best of our knowledge, cannot be captured by any of the existing semantics for Logic Programs. In many situations, this information is essential for making further decisions down the line. Therefore, any operation on logic programs that is expected to respect syntax-based properties like *support* cannot operate solely on the semantic level, but rather has to look inside the program and acknowledge rules as the atomic pieces of knowledge. At the same time, however, rules need not be manipulated in their original form. The abstraction provided by Logic Programming semantics such as SE-models can be used to discard unimportant differences between the syntactic forms of rules and focus on their semantic content. Thus, while a program cannot be viewed as the set of its models for reasons

described above, it can still be viewed as a *set of sets of models* of rules in it. Such a shift of focus should make the approach easier to manage theoretically, while not neglecting the importance of literal dependencies expressed in individual rules. It could also become a bridge between existing approaches to rule evolution and properties as well as operator constructions known from Belief Change, not only highlighting the differences between them, but also clarifying why such differences arise in the first place.

However, before a deeper investigation of such an approach can begin, we do need to know more about the relation of SE-models and individual rules. This is the aim of this paper, where we:

- identify a set of representatives of rule equivalence classes induced by the SE-model semantics, which we dub *canonical rules*;
- show how to reconstruct *canonical rules* from their sets of SE-models;
- based on the above, characterise the sets of SE-interpretations that are representable by a single rule;
- reveal connections between the set of SE-models of a rule and convex sublattices of the set of classical interpretations;
- introduce two new notions of equivalence – stronger than *strong equivalence* [1] and weaker than *strong update equivalence* [2] – and argue that they are more suitable when rules are to be treated as first-class citizens.

We believe that these results provide important insights into the workings of SE-models with respect to individual rules and will serve as a toolset for manipulating logic programs at the semantic level.

The rest of this document is structured as follows: We introduce syntax and semantics of logic programs in Sect. 2 while in Sect. 3 we define the set of representatives for rule equivalence classes and introduce transformations pinpointing the expressivity of SE-model semantics with respect to individual rules. We also give two characterisations of the sets of SE-interpretations that are representable by a single rule. In Sect. 4 we discuss the relevance of our results and propose the two new notions of equivalence.

2 Preliminaries

We assume to be given a nonempty, finite set of propositional atoms $\mathcal{L}$ from which we construct both propositional formulae and rules.

Propositional formulae are formed in the usual way from propositional atoms in $\mathcal{L}$, the logical constants $\top$ an $\bot$, and the connectives $\neg, \wedge, \vee, \subset, \supset, \equiv$. An *interpretation* is any subset of $\mathcal{L}$, naturally inducing a truth assignment to all propositional formulae. If a formula ϕ is true under interpretation I, we also say that I is a *model of* ϕ. The set of all interpretations is denoted by $\mathcal{I}$.

Similarly as for propositional formulae, the basic syntactic building blocks of rules are propositional atoms from $\mathcal{L}$. A *negative literal* is an atom preceded by $\sim$, denoting default negation. A *literal* is either an atom or a negative literal. As a convention, double default negation is absorbed, so that $\sim\sim p$ denotes the atom p. Given a set of literals X, we introduce the following notation:

$$X^+ = \{\, p \in \mathcal{L} \mid p \in X \,\} \quad X^- = \{\, p \in \mathcal{L} \mid \sim p \in X \,\} \quad \sim X = \{\, \sim p \mid p \in X \cap \mathcal{L} \,\}$$

Given natural numbers k, l, m, n and atoms $p_1, \ldots, p_k,\ q_1, \ldots, q_l,\ r_1, \ldots, r_m,\ s_1, \ldots, s_n$, we say the pair of sets of literals

$$\langle \{\ p_1, \ldots, p_k, {\sim}q_1, \ldots, {\sim}q_l\ \}, \{\ r_1, \ldots, r_m, {\sim}s_1, \ldots, {\sim}s_n\ \} \rangle \tag{1}$$

is a *rule*. The first component of a rule (1) is denoted by $H(r)$ and the second by $B(r)$. We say $H(r)$ is the *head of* r, $H(r)^+$ is the *positive head of* r, $H(r)^-$ is the *negative head of* r, $B(r)$ is the *body of* r, $B(r)^+$ is the *positive body of* r and $B(r)^-$ is the *negative body of* r. Usually, for convenience, instead of a rule r of the form (1) we write the expression

$$p_1; \ldots; p_k; {\sim}q_1; \ldots; {\sim}q_l \leftarrow r_1, \ldots, r_m, {\sim}s_1, \ldots, {\sim}s_n. \tag{2}$$

or, alternatively, $H(r)^+; {\sim}H(r)^- \leftarrow B(r)^+, {\sim}B(r)^-$. A rule is called *positive* if its head and body contain only atoms. A *program* is any set of rules.

We also introduce the following non-standard notion which we will need throughout the rest of the paper:

Definition 1 (Canonical Tautology). *Let p_ε be an arbitrary but fixed atom. The* canonical tautology*, denoted by ε, is the rule $p_\varepsilon \leftarrow p_\varepsilon$.*

In the following, we define two semantics for rules. One is that of classical models, where a rule is simply treated as a classical implication. The other is based on the logic of Here-and-There [16, 17], more accurately on a reformulation of the here-and-there semantics, called *SE-model semantics*, defined for rules [18]. This second semantics is strictly more expressive than both classical models and the stable model semantics [19].

We introduce the classical model of a rule by translating the rule into a propositional formula: Given a rule r of the form (2), we define the propositional formula $\overline{r}$ as $\bigvee \{\ p_1, \ldots, p_k, \neg q_1, \ldots, \neg q_l\ \} \subset \bigwedge \{\ r_1, \ldots, r_m, \neg s_1, \ldots, \neg s_n\ \}$. Note that $\bigvee \emptyset \equiv \bot$ and $\bigwedge \emptyset \equiv \top$. A classical model, or *C-model*, of a rule r is any model of the formula $\overline{r}$.

Given a rule r and an interpretation J, we define the *reduct of r relative to J*, denoted by r^J, as follows: If some atom from $H(r)^-$ is false under J or some atom from $B(r)^-$ is true under J, then r^J is ε; otherwise r^J is $H(r)^+ \leftarrow B(r)^+$. Intuitively, the reduct r^J is the positive part of a rule r that "remains" after all its negative literals are interpreted under interpretation J. The two conditions in the definition check whether the rule is satisfied based on the negative atoms in its head and body, interpreted under J. If this is the case, the reduct is by definition the canonical tautology. If none of these conditions is satisfied, the positive parts of r are kept in the reduct, discarding the negative ones.

An *SE-interpretation* is a pair of interpretations $\langle I, J \rangle$ such that I is a subset of J. The set of all SE-interpretations is denoted by $\mathcal{I}^{\mathsf{SE}}$. We say that an SE-interpretation $\langle I, J \rangle$ is an *SE-model* of a rule r if J is a C-model of r and I is a C-model of r^J. The set of all SE-models of a rule r is denoted by $\mathsf{mod}_{\mathsf{SE}}(r)$. The SE-models of a program $\mathcal{P}$ are the SE-models of all rules in $\mathcal{P}$. A set of SE-interpretations $\mathcal{S}$ is called **rule-representable** if there exists a rule r such that $\mathcal{S} = \mathsf{mod}_{\mathsf{SE}}(r)$.

We say that a rule r is **SE-tautological** if $\mathsf{mod}_{\mathsf{SE}}(r) = \mathcal{I}^{\mathsf{SE}}$. Note that the canonical tautology ε (c.f. Definition 1) is SE-tautological. We say that two rules r, r' are *strongly equivalent*, or *SE-equivalent*, if they have the same set of SE-models.

3 Rule Equivalence Classes and their Canonical Rules

Our goal is to find useful insights into the inner workings of the SE-model semantics with respect to single rules. In order to do so, we first introduce a set of representatives of rule equivalence classes induced by SE-models and show how the representative of a class can be constructed given one of its members. Then we show how to reconstruct a representative from the set of its SE-models. Finally, we pinpoint the conditions under which a set of SE-interpretations is rule-representable.

3.1 Canonical Rules

We start by bringing out simple but powerful transformations that simplify a given rule while preserving its SE-models. Most of these results have already been formulated in various ways [20, 2, 21]. The following result summarises the conditions under which a rule is SE-tautological:

Lemma 2 (Consequence of Theorem 4.4 in [2]; part i) of Lemma 2 in [21]). *Let H and B be sets of literals and p be an atom. Then a rule is SE-tautological if it takes any of the following forms:*

$$p; H \leftarrow p, B. \qquad H; {\sim}p \leftarrow B, {\sim}p. \qquad H \leftarrow B, p, {\sim}p.$$

Thus, repeating an atom in different "components" of the rule frequently causes the rule to be SE-tautological. In particular, this happens if the same atom occurs in the positive head and positive body, or in the negative head and negative body, or in the positive and negative bodies of a rule. How about the cases when the head contains a negation of a literal from the body? The following Lemma clarifies this situation:

Lemma 3 (Consequence of (3) and (4) in Lemma 1 in [21]). *Let H and B be sets of literals and L be a literal. Then rules of the following forms are SE-equivalent:*

$$H; {\sim}L \leftarrow L, B. \qquad H \leftarrow L, B. \tag{3}$$

So if a literal is present in the body of a rule, its negation can be removed from the head.

Until now we have seen that a rule r that has a common atom in at least two of the sets $H(r)^+ \cup H(r)^-$, $B(r)^+$ and $B(r)^-$ is either SE-tautological, or SE-equivalent to a rule where the atom is omitted from the rule's head. So such a rule is always SE-equivalent either to the canonical tautology ε, or to a rule without such repetitions. Perhaps surprisingly, repetitions in positive and negative head cannot be simplified away. For example, over the alphabet $\mathcal{L}_p = \{ p \}$, the rule "$p; {\sim}p \leftarrow$." has two SE-models, $\langle \emptyset, \emptyset \rangle$ and $\langle \{ p \}, \{ p \} \rangle$, so it is not SE-tautological, nor is it SE-equivalent to any of the facts "p." and "${\sim}p$.". Actually, it is not very difficult to see that it is not SE-equivalent to *any* other rule, even over larger alphabets. So the fact that an atom is in both $H(r)^+$ and $H(r)^-$ cannot all by itself imply that some kind of SE-models preserving rule simplification is possible.

The final Lemma reveals a special case in which we can eliminate the whole negative head of a rule and move it to its positive body. This occurs whenever the positive head is empty.

Lemma 4 (Related to Corollary 4.10 in [20] and Corollary 1 in [21]). *Let H^- be a set of negative literals, B be a set of literals and p be an atom. Then rules of the following forms are SE-equivalent:*

$$\sim p; H^- \leftarrow B. \qquad\qquad H^- \leftarrow p, B.$$

Armed with the above results, we can introduce the notion of a canonical rule. Each such rule represents a different equivalence class on the set of all rules induced by the SE-model semantics. In other words, every rule is SE-equivalent to exactly one canonical rule. After the definition, we provide constructive transformations which show that this is indeed the case. Note that the definition can be derived directly from the Lemmas above:

Definition 5 (Canonical Rule). *We say a rule r is* canonical *if either it is ε, or the following conditions are satisfied:*

1. *The sets $H(r)^+ \cup H(r)^-$, $B(r)^+$ and $B(r)^-$ are pairwise disjoint.*
2. *If $H(r)^+$ is empty, then $H(r)^-$ is also empty.*

This definition is closely related with the notion of a *fundamental rule* introduced in Definition 1 of [21]. There are two differences between canonical and fundamental rules: (1) a fundamental rule must satisfy condition 1. above, but need not satisfy condition 2.; (2) no SE-tautological rule is fundamental. As a consequence, fundamental rules do not cover all rule-representable sets of SE-interpretations, and two distinct fundamental rules may still be SE-equivalent. From the point of view of rule equivalence classes induced by SE-model semantics, there is one class that contains no fundamental rule, and some classes contain more than one fundamental rule. In the following we show that canonical rules overcome both of these limitations of fundamental rules. In other words, every rule is SE-equivalent to exactly one canonical rule. To this end, we define constructive transformations that directly show the mutual relations between rule syntax and semantics.

The following transformation provides a direct way of constructing a canonical rule that is SE-equivalent to a given rule r.

Definition 6 (Transformation into a Canonical Rule). *Given a rule r, by $\mathsf{can}(r)$ we denote a canonical rule constructed as follows: If any of the sets $H(r)^+ \cap B(r)^+$, $H(r)^- \cap B(r)^-$ and $B(r)^+ \cap B(r)^-$ is nonempty, then $\mathsf{can}(r)$ is ε. Otherwise, $\mathsf{can}(r)$ is of the form $H^+; \sim H^- \leftarrow B^+, \sim B^-$. where*

- *$H^+ = H(r)^+ \setminus B(r)^-$.*
- *If H^+ is empty, then $H^- = \emptyset$ and $B^+ = B(r)^+ \cup H(r)^-$.*
- *If H^+ is nonempty, then $H^- = H(r)^- \setminus B(r)^+$ and $B^+ = B(r)^+$.*
- *$B^- = B(r)^-$.*

Correctness of the transformation follows directly from Lemmas 2 to 4.

Theorem 7. *Every rule r is SE-equivalent to the canonical rule $\mathsf{can}(r)$.*

What remains to be proven is that no two different canonical rules are SE-equivalent. In the next Subsection we show how every canonical rule can be reconstructed from the set of its SE-models. As a consequence, no two different canonical rules can have the same set of SE-models.

3.2 Reconstructing Rules

In order to reconstruct a rule r from the set $\mathcal{S}$ of its SE-models, we need to understand how exactly each literal in the rule influences its models. The following Lemma provides a useful characterisation of the set of countermodels of a rule in terms of syntax:

Lemma 8 (Different formulation of Theorem 4 in [21]). *Let r be a rule. An SE-interpretation $\langle I, J\rangle$ is not an SE-model of r if and only if the following conditions are satisfied:*

1. $H(r)^- \cup B(r)^+ \subseteq J$ *and* $J \subseteq \mathcal{L} \setminus B(r)^-$.
2. *Either* $J \subseteq \mathcal{L} \setminus H(r)^+$ *or both* $B(r)^+ \subseteq I$ *and* $I \subseteq \mathcal{L} \setminus H(r)^+$.

The first condition together with the first disjunct of the second condition hold if and only if J is not a C-model of r. The second disjunct then captures the case when I is not a C-model of r^J.

If we take a closer look at these conditions, we find that the presence of a negative body atom in J guarantees that the first condition is falsified, so $\langle I, J\rangle$ is a model of r, regardless of the content of I. Somewhat similar is the situation with positive head atoms – whenever such an atom is present in I, it is also present in J, so the second condition is falsified and $\langle I, J\rangle$ is a model of r. Thus, if $\mathcal{S}$ is the set of SE-models of a rule r, then every atom $p \in B(r)^-$ satisfies

$$p \in J \text{ implies } \langle I, J\rangle \in \mathcal{S} \qquad (C_{B^-})$$

and every atom $p \in H(r)^+$ satisfies

$$p \in I \text{ implies } \langle I, J\rangle \in \mathcal{S} \ . \qquad (C_{H^+})$$

If we restrict ourselves to canonical rules different from ε, we find that these conditions are not only necessary, but, when combined properly, also sufficient to decide what atoms belong to the negative body and positive head of the rule.

For the rest of this Subsection, we assume that r is a canonical rule different from ε and $\mathcal{S}$ is the set of SE-models of r. Keeping in mind that every atom that satisfies condition (C_{B^-}) also satisfies condition (C_{H^+}) (because I is a subset of J), and that $B(r)^-$ is by definition disjoint from $H(r)^+$, we arrive at the following results:

Lemma 9. *An atom p belongs to $B(r)^-$ if and only if for all $\langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}}$, the condition (C_{B^-}) is satisfied. An atom p belongs to $H(r)^+$ if and only if it does not belong to $B(r)^-$ and for all $\langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}}$, the condition (C_{H^+}) is satisfied.*

As can be seen from Lemma 8, the role of positive body and negative head atoms is dual to that of negative body and positive head atoms. Intuitively, their absence in J, and sometimes also in I, implies that $\langle I, J\rangle$ is an SE-model of r. It follows from the first condition of Lemma 8 that if p belongs to $H(r)^- \cup B(r)^+$, then the following condition is satisfied:

$$p \notin J \text{ implies } \langle I, J\rangle \in \mathcal{S} \ . \qquad (C_{H^-})$$

Furthermore, the second condition in Lemma 8 implies that every $p \in B(r)^+$ satisfies the following condition:

$$p \notin I \text{ and } J \cap H(r)^+ \neq \emptyset \text{ implies } \langle I, J\rangle \in \mathcal{S} \ . \qquad (C_{B^+})$$

These observations lead to the following results:

Lemma 10. *An atom p belongs to $B(r)^+$ if and only if for all $\langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}}$, the conditions (C_{H^-}) and (C_{B^+}) are satisfied. An atom p belongs to $H(r)^-$ if and only if it does not belong to $B(r)^+$ and for all $\langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}}$, the condition (C_{H^-}) is satisfied.*

Together, the two Lemmas above are sufficient to reconstruct a canonical rule from its set of SE-models. The following definition sums up these results by introducing the notion of a rule induced by a set of SE-interpretations:

Definition 11 (Rule Induced by a Set of SE-Interpretations).
Let $\mathcal{S}$ be a set of SE-interpretations.

*An atom p is called an $\mathcal{S}$-*negative-body atom *if every SE-interpretation $\langle I, J\rangle$ with $p \in J$ belongs to $\mathcal{S}$. An atom p is called an $\mathcal{S}$-*positive-head atom *if it is not an $\mathcal{S}$-negative-body atom and every SE-interpretation $\langle I, J\rangle$ with $p \in I$ belongs to $\mathcal{S}$.*

*An atom p is called an $\mathcal{S}$-*positive-body atom *if every SE-interpretation $\langle I, J\rangle$ with $p \notin J$ belongs to $\mathcal{S}$, and every SE-interpretation $\langle I, J\rangle$ with $p \notin I$ and J containing some $\mathcal{S}$-positive-head atom also belongs to $\mathcal{S}$. An atom p is called an $\mathcal{S}$-*negative-head atom *if it is not an $\mathcal{S}$-positive-body atom and every SE-interpretation $\langle I, J\rangle$ with $p \notin J$ belongs to $\mathcal{S}$.*

The sets of all $\mathcal{S}$-negative-body, $\mathcal{S}$-positive-head, $\mathcal{S}$-positive-body and $\mathcal{S}$-negative-head atoms are denoted by $B(\mathcal{S})^-$, $H(\mathcal{S})^+$, $B(\mathcal{S})^+$ and $H(\mathcal{S})^-$, respectively. The rule induced by *$\mathcal{S}$, denoted by* $\mathsf{rule}(\mathcal{S})$*, is defined as follows: If $\mathcal{S} = \mathcal{I}^{\mathsf{SE}}$, then* $\mathsf{rule}(\mathcal{S})$ *is ε; otherwise,* $\mathsf{rule}(\mathcal{S})$ *is of the form*

$$H(\mathcal{S})^+; {\sim}H(\mathcal{S})^- \leftarrow B(\mathcal{S})^+, {\sim}B(\mathcal{S})^-.$$

The main property of induced rules is that every canonical rule is induced by its own set of SE-models and can thus be "reconstructed" from its set of SE-models. This follows directly from Definition 11 and Lemmas 9 and 10.

Theorem 12. *For every canonical rule r,* $\mathsf{rule}(\mathsf{mod}_{\mathsf{SE}}(r)) = r$.

This result, together with Theorem 7, has a number of consequences. First, for any rule r, the canonical rule $\mathsf{can}(r)$ is induced by the set of SE-models of r.

Corollary 13. *For every rule r,* $\mathsf{rule}(\mathsf{mod}_{\mathsf{SE}}(r)) = \mathsf{can}(r)$.

Furthermore, Theorem 12 directly implies that for two different canonical rules r_1, r_2 we have $\mathsf{rule}(\mathsf{mod}_{\mathsf{SE}}(r_1)) = r_1$ and $\mathsf{rule}(\mathsf{mod}_{\mathsf{SE}}(r_2)) = r_2$, so $\mathsf{mod}_{\mathsf{SE}}(r_1)$ and $\mathsf{mod}_{\mathsf{SE}}(r_2)$ must differ.

Corollary 14. *No two different canonical rules are SE-equivalent.*

Finally, the previous Corollary together with Theorem 7 imply that for every rule there not only exists an SE-equivalent canonical rule, but this rule is also unique.

Corollary 15. *Every rule is SE-equivalent to exactly one canonical rule.*

3.3 Sets of SE-Interpretations Representable by a Rule

Naturally, not all sets of SE-interpretations correspond to a single rule, otherwise any program could be reduced to a single rule. The conditions under which a set of SE-interpretations is rule-representable are worth examining.

A set of SE-models $\mathcal{S}$ of a program is always *well-defined*, i.e. whenever $\mathcal{S}$ contains $\langle I, J\rangle$, it also contains $\langle J, J\rangle$. Moreover, for every well-defined set of SE-interpretations $\mathcal{S}$ there exists a program $\mathcal{P}$ such that $\mathcal{S} = \mathsf{mod}_{\mathsf{SE}}(\mathcal{P})$ [10].

We offer two approaches to find a similar condition for the class of rule-representable sets of SE-interpretations. The first is based on induced rules defined in the previous Subsection, while the second is formulated using lattice theory and is a consequence of Lemma 8.

The first characterisation follows from two properties of the $\mathsf{rule}(\cdot)$ transformation. First, it can be applied to any set of SE-interpretations, even those that are not rule-representable. Second, if $\mathsf{rule}(\mathcal{S}) = r$, then it holds that $\mathsf{mod}_{\mathsf{SE}}(r)$ is a subset of $\mathcal{S}$.

Lemma 16. *The set of all SE-models of a canonical rule r is the least among all sets of SE-interpretations $\mathcal{S}$ such that* $\mathsf{rule}(\mathcal{S}) = r$.

Thus, to verify that $\mathcal{S}$ is rule-representable, it suffices to check that all interpretations from $\mathcal{S}$ are models of $\mathsf{rule}(\mathcal{S})$.

The second characterisation follows from Lemma 8 which tells us that if $\mathcal{S}$ is rule-representable, then its complement consists of SE-interpretations $\langle I, J\rangle$ following a certain pattern. Their second component J always contains a fixed set of atoms and is itself contained in another fixed set of atoms. Their first component I satisfies a similar property, but only if a certain further condition is satisfied by J. More formally, for the sets

$$I^{\perp} = B(r)^{+}, \quad I^{\top} = \mathcal{L} \setminus H(r)^{+}, \quad J^{\perp} = H(r)^{-} \cup B(r)^{+}, \quad J^{\top} = \mathcal{L} \setminus B(r)^{-},$$

it holds that all SE-interpretations from the complement of $\mathcal{S}$ are of the form $\langle I, J\rangle$ where $J^{\perp} \subseteq J \subseteq J^{\top}$ and either $J \subseteq I^{\top}$ or $I^{\perp} \subseteq I \subseteq I^{\top}$. It turns out that this also holds vice versa: if the complement of $\mathcal{S}$ satisfies the above property, then $\mathcal{S}$ is rule-representable. Furthermore, to accentuate the particular structure that arises, we can substitute the condition $J^{\perp} \subseteq J \subseteq J^{\top}$ with saying that J belongs to a convex sublattice of $\mathcal{I}$.[1] A similar substitution can be performed for I, yielding:

Theorem 17. *Let $\mathcal{S}$ be a set of SE-interpretations. Then the following conditions are equivalent:*

1. *The set of SE-interpretations $\mathcal{S}$ is rule-representable.*
2. *All SE-interpretations from $\mathcal{S}$ are SE-models of* $\mathsf{rule}(\mathcal{S})$.
3. *There exist convex sublattices L_1, L_2 of $\langle \mathcal{I}, \subseteq\rangle$ such that the complement of $\mathcal{S}$ relative to $\mathcal{I}^{\mathsf{SE}}$ is equal to*

$$\left\{ \langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}} \;\middle|\; I \in L_1 \wedge J \in L_2 \right\} \cup \left\{ \langle I, J\rangle \in \mathcal{I}^{\mathsf{SE}} \;\middle|\; J \in L_1 \cap L_2 \right\} \quad .$$

[1] A sublattice L of L' is *convex* if $c \in L$ whenever $a, b \in L$ and $a \leq c \leq b$ holds in L'. For more details see e.g. [22].

4 Discussion

The presented results mainly serve to facilitate the transition back and forth between a rule and the set of its SE-models. They also make it possible to identify when a given set of SE-models is representable by a single rule. We believe that in situations where information on literal dependencies, expressed in individual rules, is essential for defining operations on logic programs, the advantages of dealing with rules on the level of semantics instead of on the level of syntax are significant. The semantic view takes care of stripping away unnecessary details and since the introduced notions and operators are defined in terms of semantic objects, it should be much easier to introduce and prove their semantic properties.

These results can be used for example in the context of program updates to define an update semantics based on the *rule rejection principle* [4] and operating on *sets of sets of SE-models*. Such a semantics can serve as a bridge between syntax-based approaches to rule updates, and the principles and semantic distance measures known from the area of Belief Change. The next steps towards such a semantics involve a definition of the notion of support for a literal by a set of SE-models (of a rule). Such a notion can then foster a better understanding of desirable properties for semantic rule update operators.

On a different note, viewing a logic program as the *set of sets of SE-models* of rules inside it leads naturally to the introduction of the following new notion of program equivalence:

Definition 18 (Strong Rule Equivalence). *Programs $\mathcal{P}_1, \mathcal{P}_2$ are* SR-equivalent*, denoted by $\mathcal{P}_1 \equiv_{\mathsf{SR}} \mathcal{P}_2$, if*

$$\{\, \mathsf{mod}_{\mathsf{SE}}(r) \mid r \in \mathcal{P}_1 \cup \{\, \varepsilon \,\} \,\} = \{\, \mathsf{mod}_{\mathsf{SE}}(r) \mid r \in \mathcal{P}_2 \cup \{\, \varepsilon \,\} \,\} \quad .$$

Thus, two programs are SR-equivalent if they contain the same rules, modulo the SE-model semantics. We add ε to each of the two programs in the definition so that presence or absence of tautological rules in a program does not influence program equivalence. SR-equivalence is stronger than strong equivalence, in the following sense:

Definition 19 (Strength of Program Equivalence). *Let $\equiv_1, \equiv_2$ be equivalence relations on the set of all programs. We say that $\equiv_1$* is at least as strong as *$\equiv_2$, denoted by $\equiv_1 \succeq \equiv_2$, if $\mathcal{P}_1 \equiv_1 \mathcal{P}_2$ implies $\mathcal{P}_1 \equiv_2 \mathcal{P}_2$ for all programs $\mathcal{P}_1, \mathcal{P}_2$. We say that $\equiv_1$* is stronger than *$\equiv_2$, denoted by $\equiv_1 \succ \equiv_2$, if $\equiv_1 \succeq \equiv_2$ but not $\equiv_2 \succeq \equiv_1$.*

Thus, using the notation of the above definition, we can write $\equiv_{\mathsf{SR}} \succ \equiv_{\mathsf{S}}$, where $\equiv_{\mathsf{S}}$ denotes the relation of strong equivalence. An example of programs that are strongly equivalent, but not SR-equivalent is $\mathcal{P} = \{\, p., q. \,\}$ and $\mathcal{Q} = \{\, p., q \leftarrow p. \,\}$, which in many cases need to be distinguished from one another. We believe that this notion of program equivalence is much more suitable for cases when the dependency information contained in a program is of importance.

In certain cases, however, SR-equivalence may be too strong. For instance, it may be desirable to treat programs such as $\mathcal{P}_1 = \{\, p \leftarrow q. \,\}$ and $\mathcal{P}_2 = \{\, p \leftarrow q., p \leftarrow q, r. \,\}$ in the same way because the extra rule in $\mathcal{P}_2$ is just a weakened version of the rule in $\mathcal{P}_1$.

For instance, the notion of *update equivalence* introduced in [23], which is based on a particular approach to logic program updates, considers programs $\mathcal{P}_1$ and $\mathcal{P}_2$ as equivalent because the extra rule in $\mathcal{P}_2$ cannot influence the result of any subsequent updates. Since these programs are not SR-equivalent, we also introduce the following notion of program equivalence, which in terms of strength falls between strong equivalence and SR-equivalence.

Definition 20 (Strong Minimal Rule Equivalence). *Programs $\mathcal{P}_1, \mathcal{P}_2$ are* SMR-equivalent*, denoted by $\mathcal{P}_1 \equiv_{\mathsf{SMR}} \mathcal{P}_2$, if*

$$\min \{\, \mathsf{mod}_{\mathsf{SE}}(r) \mid r \in \mathcal{P}_1 \cup \{\, \varepsilon \,\} \,\} = \min \{\, \mathsf{mod}_{\mathsf{SE}}(r) \mid r \in \mathcal{P}_2 \cup \{\, \varepsilon \,\} \,\} \quad ,$$

where $\min \mathcal{S}$ denotes the set of subset-minimal elements of $\mathcal{S}$.

In order for programs to be SMR-equivalent, they need not contain exactly the same rules (modulo strong equivalence), it suffices if rules with subset-minimal sets of SE-models are the same (again, modulo strong equivalence). Certain programs, such as $\mathcal{P}_1$ and $\mathcal{P}_2$ above, are not SR-equivalent but they are still SMR-equivalent.

Related to this is the very strong notion of equivalence which was introduced in [2]:

Definition 21 (Strong Update Equivalence, c.f. Definition 4.1 in [2]). *Two programs $\mathcal{P}_1$, $\mathcal{P}_1$ are SU-equivalent, denoted by $\mathcal{P}_1 \equiv_{\mathsf{SU}} \mathcal{P}_2$, if for any programs $\mathcal{Q}$, $\mathcal{R}$ it holds that the program $((\mathcal{P}_1 \setminus \mathcal{Q}) \cup \mathcal{R})$ has the same answer sets as the program $((\mathcal{P}_2 \setminus \mathcal{Q}) \cup \mathcal{R})$.*

Two programs are strongly update equivalent only under very strict conditions – it is shown in [2] that two programs are SU-equivalent if and only if their symmetric difference contains only SE-tautological rules. This means that programs such as $\mathcal{Q}_1 = \{\, \sim p. \,\}$, $\mathcal{Q}_2 = \{\, \leftarrow p. \,\}$ and $\mathcal{Q}_3 = \{\, \sim p \leftarrow p. \,\}$ are considered to be mutually non-equivalent, even though the rules they contain are mutually SE-equivalent. This may be seen as too sensitive to rule syntax.

The following result formally establishes the relations between the discussed notions of program equivalence:

Theorem 22. *SU-equivalence is stronger than SR-equivalence, which itself is stronger than SMR-equivalence, which in turn is stronger than strong equivalence. That is,*

$$\equiv_{\mathsf{SU}} \succ \equiv_{\mathsf{SR}} \succ \equiv_{\mathsf{SMR}} \succ \equiv_{\mathsf{S}} \quad .$$

The other notion of program equivalence introduced in [2], *strong update equivalence on common rules*, or SUC-equivalence, is incomparable in terms of strength to our new notions of equivalence. On the one hand, SR- and SMR-equivalent programs such as $\{\, \sim p. \,\}$ and $\{\, \sim p., \leftarrow p. \,\}$ are not SUC-equivalent. On the other hand, programs such as $\{\, p., q \leftarrow p. \,\}$ and $\{\, q., p \leftarrow q. \,\}$ are neither SR- nor SMR-equivalent, but they are SUC-equivalent. We believe that both of these examples are more appropriately treated by the new notions of equivalence.

The introduction of canonical rules, which form a set of representatives of rule equivalence classes induced by SE-models, also reveals the exact expressivity of SE-model semantics with respect to a single rule. From their definition we can see that

SE-models are capable of distinguishing between any pair of rules, except for (1) a pair of rules that only differ in the number of repetitions of literals in their heads and bodies; (2) an integrity constraint and a rule whose head only contains negative literals. We believe that in the former case, there is little reason to distinguish between such rules and so the transition from rules to their SE-models has the positive effect of stripping away of unnecessary details. However, the latter case has more serious consequences. Although rules such as

$$\sim p \leftarrow q. \qquad \text{and} \qquad \leftarrow p, q.$$

are usually considered to carry the same meaning, some existing work suggests that they should be treated differently – while the former rule gives a reason for atom p to become false whenever q is true, the latter rule simply states that the two atoms cannot be true at the same time, without specifying a way to resolve this situation if it were to arise [4, 8]. If we view a rule through the set of its SE-models, we cannot distinguish these two kinds of rules anymore. Whenever this is important, either *strong update equivalence* is used, which is perhaps *too* sensitive to the syntax of rules, or a new characterisation of Answer-Set Programming needs to be discovered, namely one that is not based on the logic of Here-and-There [16, 17].

Acknowledgement

We would like to thank Han The Anh, Matthias Knorr and the anonymous reviewers for their comments that helped to improve the paper. Martin Slota is supported by FCT scholarship SFRH / BD / 38214 / 2007.

References

1. Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
2. Katsumi Inoue and Chiaki Sakama. Equivalence of logic programs under updates. In José Júlio Alferes and João Alexandre Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, volume 3229 of *Lecture Notes in Computer Science*, pages 174–186, Lisbon, Portugal, September 27-30 2004. Springer.
3. Carlos Viegas Damásio, Luís Moniz Pereira, and Michael Schroeder. REVISE: Logic programming and diagnosis. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 354–363, Dagstuhl Castle, Germany, July 28-31 1997. Springer.
4. José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1-3):43–70, September/October 2000.
5. Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6):721–777, 2002.
6. Chiaki Sakama and Katsumi Inoue. An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming*, 3(6):671713, 2003.

7. Yan Zhang. Logic program-based updates. *ACM Transactions on Computational Logic*, 7(3):421–472, 2006.
8. José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.
9. James P. Delgrande, Torsten Schaub, and Hans Tompits. A preference-based framework for updating logic programs. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 4483 of *Lecture Notes in Computer Science*, pages 71–83, Tempe, AZ, USA, May 15-17 2007. Springer.
10. James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran. Belief revision of logic programs under answer set semantics. In Gerhard Brewka and Jérôme Lang, editors, *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, pages 411–421, Sydney, Australia, September 16-19 2008. AAAI Press.
11. James P. Delgrande. A Program-Level Approach to Revising Logic Programs under the Answer Set Semantics. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming Special Issue*, 10(4-6):565–580, July 2010.
12. Peter Gärdenfors. *Belief Revision*, chapter Belief Revision: An Introduction, pages 1–28. Cambridge University Press, 1992.
13. Martin Slota and João Leite. On semantic update operators for answer-set programs. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *Proceedings of the 19th European Conference on Artificial Intelligence*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 957–962, Lisbon, Portugal, August 16-20 2010. IOS Press.
14. Krzysztof R. Apt, Howard A. Blair, and Adrian Walker. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, 1988.
15. Jürgen Dix. A classification theory of semantics of normal logic programs: II. Weak properties. *Fundamenta Informaticae*, 22(3):257–288, 1995.
16. Jan Łukasiewicz. Die Logik und das Grundlagenproblem. In *Les Entretiens de Zürich sue les Fondements et la méthode des sciences mathématiques 1938*, pages 82–100. Zürich, 1941.
17. David Pearce. A new logical characterisation of stable models and answer sets. In Jürgen Dix, Luís Moniz Pereira, and Teodor C. Przymusinski, editors, *Proceedings of the 6th Workshop on Non-Monotonic Extensions of Logic Programming*, volume 1216 of *Lecture Notes in Computer Science*, pages 57–70, Bad Honnef, Germany, September 5-6 1997. Springer.
18. Hudson Turner. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4-5):609–622, 2003.
19. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080, Seattle, Washington, August 15-19 1988. MIT Press.
20. Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.
21. Pedro Cabalar, David Pearce, and Agustín Valverde. Minimal logic programs. In Verónica Dahl and Ilkka Niemelä, editors, *Proceedings of the 23rd International Conference on Logic Programming (ICLP 2007)*, volume 4670 of *Lecture Notes in Computer Science*, pages 104–118, Porto, Portugal, September 8-13 2007. Springer.
22. Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.
23. João Alexandre Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers of Artificial Intelligence and Applications, xviii + 307 p. Hardcover*. IOS Press, 2003.