Ann Math Artif Intell (2018) 83:277–295
https://doi.org/10.1007/s10472-017-9567-5

Equivalence between answer-set programs under (partially) fixed input

Bernhard Bliem[1] · **Stefan Woltran**[1]

Published online: 10 November 2017
© The Author(s) 2017. This article is an open access publication

Abstract Answer Set Programming has become an increasingly popular formalism for declarative problem solving. Among the huge body of theoretical results, investigations of different equivalence notions between logic programs play a fundamental role for understanding modularity and optimization. While strong equivalence between two programs holds if they can be faithfully replaced by each other in any context (facts and rules), uniform equivalence amounts to equivalent behavior of programs under any set of facts. Both notions (as well as several variants thereof) have been extensively studied. However, the somewhat reverse notion of equivalence which holds if two programs are equivalent under the addition of any set of proper rules (i.e., all rules except facts) has not been investigated yet. In this paper, we close this gap and give a thorough study of this notion, which we call rule equivalence , and its parameterized version where we allow facts over a given restricted alphabet to appear in the context. This notion of equivalence is thus a relationship between two programs whose input is (partially) fixed but where additional proper rules might still be added. Such a notion might be helpful in debugging of programs. We give full characterization results and a complexity analysis for the propositional case of rule equivalence and its relativized versions. Moreover, we address the problem of program simplification under rule equivalence. Finally, we show that rule equivalence is decidable in the non-ground case.

Keywords Answer set programming · Equivalence

Mathematics Subject Classification (2010) 68N17

✉ Bernhard Bliem
bliem@dbai.tuwien.ac.at

Stefan Woltran
woltran@dbai.tuwien.ac.at

[1] Institute of Information Systems, TU Wien, Karlsplatz 13, 1040 Wien, Austria

1 Motivation

In the area of Answer Set Programming [1] investigations of different equivalence notions have been a major research topic within the last 15 years. This is due to the fact that the straightforward notion of equivalence, which holds if two programs possess the same answer sets, is too weak to guarantee faithful replacements. In other words, replacing within a program R a subprogram P by program P' equivalent to that subprogram might change the answer sets of the thus modified program $R[P/P']$. Despite this effect being implicit to any nonmonotonic formalism, little was known how to characterize a sufficiently strong notion of equivalence in order to guarantee such faithful replacements.

In their seminal paper on strong equivalence between logic programs, Lifschitz, Pearce and Valverde [2], have found a strikingly simple and elegant solution. First, they explicitly defined the required notion: P and Q are strongly equivalent iff, for each further so-called context program R, $P \cup R$ and $Q \cup R$ possess the same answer sets. Secondly, they gave a characterization in terms of a monotonic non-classical logic: P and Q are strongly equivalent iff P and Q are equivalent in the logic of here-and-there; this characterization was later reformulated to be applied more directly to programs and their reducts by Turner [3] who introduced the notion of SE-models.

The difference between strong and ordinary equivalence motivated investigations of equivalence notions in between. Uniform equivalence, originally introduced by Sagiv [4] as an approximation for datalog equivalence, is such an example. Uniform equivalence tests whether, for each set F of facts, $P \cup F$ and $Q \cup F$ possess the same answer sets. This notion has been adapted to answer set programming by Eiter and Fink [5], who also provided a characterization that is based on a subset of SE-models (called UE-models); alternative characterizations can be found in [6]. A further direction is known as relativized equivalence or hyperequivalence. Here the atoms allowed to occur in the context programs are restricted to stem from a given alphabet [7–11].

Concerning further notions of equivalence, it was claimed that any "reasonable" attempt to syntactically restrict the context programs (i.e., where the restriction is defined rule-wise, for instance only allowing context programs with Horn rules) coincides with either ordinary, strong, or uniform equivalence (see, e.g., [12]). The reason behind this claim is the fact that strong equivalence coincides with a much simpler notion of equivalence. Define the class of unary program as consisting of facts and rules of the form $a \leftarrow b$ only. It can be shown that programs P and Q are strongly equivalent, if and only if, for any unary program R, the answer sets of $P \cup R$ and $Q \cup R$ coincide.

However, there is still room for finding another equivalence notion. What if we allow in the context programs R of an equivalence only proper rules, but no facts? Surprisingly, this form of equivalence, which we call *rule equivalence*, has not been investigated yet. Indeed such a notion can prove useful as a test for replacement of program modules under fixed known input and appears to be more natural than other notions of modular equivalence from the literature [11, 13, 14].

As a second contribution, we go towards relativized equivalence and relax our notion by allowing the potential addition of some facts (similar to relativized uniform equivalence). It turns out that this equivalence notion boils down to strong equivalence if both programs under consideration already contain facts, but is a different concept otherwise.

Understanding different notions of equivalence in Answer-Set Programming not only provided fundamental insights into the nature of nonmonotonic formalisms, it also gave pointers for static program optimization (see, e.g., [15, 16]). We shall contribute to this line

of research by considering the question under which circumstances a given program can be transformed into a rule-equivalent program from a simpler class.

We see our contribution mainly as a missing link between established equivalence notions. As our results suggest, rule equivalence is somewhat closer to strong equivalence than to uniform equivalence. This becomes in particular apparent by the observation that for programs containing at least one fact, strong and rule equivalence coincide. Moreover, in the non-ground case it was proven that strong equivalence is decidable while uniform equivalence is not [17]. We complement the picture and prove decidability of rule equivalence in the non-ground case.

To summarize, the main contribution of the paper is to provide a general and uniform semantic characterization for the newly introduced notions of rule equivalence. Our characterization will follow the tradition in Answer-Set Programming and is based on SE-models. We show some basic properties of rule equivalence and also include a complexity analysis. Furthermore, we discuss aspects of program simplification under this new equivalence notion. While our paper is focused on the ground case, we also prove that rule equivalence remains decidable for the non-ground case.

This work extends a conference publication [18]. Compared to that previous work, we added new results on complexity and program simplification (Section 5).

2 Background

We first recall syntax and semantics of ground logic programs and then outline the more general non-ground case.

Ground programs Throughout the paper, we assume an arbitrary but fixed alphabet $\mathcal{U}$ of atoms. A propositional disjunctive logic program (also called a ground program, or simply a program) is a finite set of rules of the form

$$a_1 \vee \cdots \vee a_l \leftarrow b_1, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_n, \tag{1}$$

($l \geq 0, n \geq m \geq 0$), and where all a_i and b_j are propositional atoms from $\mathcal{U}$ and *not* denotes default negation; for $l = 1$ and $n = 0$, we usually identify the rule (1) with the atom a_1, and call it a *fact*. A rule of the form (1) is called a *constraint* if $l = 0$; a *disjunctive fact* if $n = 0$; *proper* if $n \geq 1$; *positive* if $m = n$; *normal* if $l \leq 1$; and *unary* if $l = 1$ and $m = n \leq 1$. A program is proper (resp., positive, normal, unary) iff all its rules are proper (resp., positive, normal, unary). A program that is positive and normal is called *Horn*. If all atoms occurring in a program P are from a given alphabet $\mathcal{A} \subseteq \mathcal{U}$ of atoms, we say that P is a program *over* (alphabet) $\mathcal{A}$.

For a rule r of the form (1), we identify its head by $H(r) = \{a_1, \ldots, a_l\}$ and its body by $B^+(r) = \{b_1, \ldots, b_m\}$ and $B^-(r) = \{b_{m+1}, \ldots, b_n\}$. We shall write rules of the form (1) also as $H(r) \leftarrow B^+(r),\ not\ B^-(r)$. Moreover, we use $B(r) = B^+(r) \cup B^-(r)$. Finally, for a program P and $\alpha \in \{H, B, B^+, B^-\}$, let $\alpha(P) = \bigcup_{r \in P} \alpha(r)$.

The relation $Y \models r$ between an interpretation Y and a rule r is defined as usual, i.e., $Y \models r$ iff $H(r) \cap Y \neq \emptyset$ whenever jointly $B^+(r) \subseteq Y$ and $B^-(r) \cap Y = \emptyset$ hold; for a program P, $Y \models P$ holds iff for each $r \in P$, $Y \models r$. If $Y \models P$ holds, Y is called a *model* of P. An interpretation Y is an *answer set* of a program P iff it is a minimal (w.r.t. set inclusion) model of the *reduct* $P^Y = \{H(r) \leftarrow B^+(r) \mid Y \cap B^-(r) = \emptyset\}$ of P w.r.t. Y. The set of all answer sets of a program P is denoted by $\mathrm{AS}(P)$.

Next, we review some prominent notions of equivalence, which have been studied under the answer-set semantics: Programs P, Q are

- *strongly equivalent* [2], iff, for any program R, $\mathrm{AS}(P \cup R) = \mathrm{AS}(Q \cup R)$;
- P and Q are *uniformly equivalent* [5], iff, for any set F of facts, $\mathrm{AS}(P \cup F) = \mathrm{AS}(Q \cup F)$.

Relativizations of these notions are as follows [8, 9]: For a given alphabet $\mathcal{A} \subseteq \mathcal{U}$, we call programs P, Q *strongly equivalent relative to* $\mathcal{A}$, iff, for any program R over $\mathcal{A}$, it holds that $\mathrm{AS}(P \cup R) = \mathrm{AS}(Q \cup R)$; P, Q are *uniformly equivalent relative to* $\mathcal{A}$, iff, for any set F of facts over $\mathcal{A}$, $\mathrm{AS}(P \cup F) = \mathrm{AS}(Q \cup F)$. In the case of strong equivalence (also in the relativized case), it was shown [2, 8] that the syntactic class of *counterexamples* (i.e., programs R such that $\mathrm{AS}(P \cup R) \neq \mathrm{AS}(Q \cup R)$) can always be restricted to the class of unary programs. All equivalence notions mentioned have been generalized to a uniform setting in [10]: Given alphabets $\mathcal{A}, \mathcal{B} \subseteq \mathcal{U}$, programs P, Q are called $(\mathcal{A}, \mathcal{B})$ equivalent iff, for any program R with $H(R) \subseteq \mathcal{A}$ and $B(R) \subseteq \mathcal{B}$, $\mathrm{AS}(P \cup R) = \mathrm{AS}(Q \cup R)$.

Given a program P, we call a pair (X, Y) SE-model of P (in symbols $(X, Y) \in \mathrm{SE}(P)$) if $X \subseteq Y$, $X \models P^Y$ and $Y \models P$. It is easy to see that Y is an answer set of P iff $(Y, Y) \in \mathrm{SE}(P)$ and $(X, Y) \notin \mathrm{SE}(P)$ holds for all $X \subset Y$. We call SE-models (X, Y) total if $X = Y$, and non-total otherwise. SE-models characterize strong equivalence in the following sense [3]. Two program P and Q are strongly equivalent iff $\mathrm{SE}(P) = \mathrm{SE}(Q)$. Based on SE-models similar (however, more involved) characterizations for the aforementioned equivalence notions have been introduced [9, 10]. We review here only the result for uniform equivalence [5]. Call an SE-model (X, Y) of P UE-model of P (in symbols $(X, Y) \in \mathrm{UE}(P)$) if for each $(X', Y) \in \mathrm{SE}(P)$ with $X \subset X' \subseteq Y$, $X' = Y$. Then, programs P and Q are uniformly equivalent iff $\mathrm{UE}(P) = \mathrm{UE}(Q)$.

Non-ground programs In the non-ground case, programs are formulated in a language containing a set $\mathcal{Q}$ of *predicate symbols*, a set $\mathcal{V}$ of *variables*, and a countably infinite set $\mathcal{C}$ of *constants*. Each predicate symbol p has an associated arity $n(p)$. Rules are of the form (1), but now an atom is an expression of form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{Q}$ is a predicate of arity n and $t_i \in \mathcal{C} \cup \mathcal{V}$, for $1 \leq i \leq n$. An atom is *ground* if no variable occurs in it. A rule r is *safe* if each variable occurring in $H(r) \cup B(r)$ also occurs in $B^+(r)$; r is *ground*, if all atoms occurring in r are ground. Given a rule r and $C \subseteq \mathcal{C}$, we define $\mathsf{G}(r, C)$ as the set of all rules obtained from r by all possible substitutions of elements of C for the variables in r. Moreover, we define $\mathsf{G}(P, C) = \bigcup_{r \in P} \mathsf{G}(r, C)$. Interpretations are now sets of ground atoms, and the concept of satisfaction ($\models$) is analogous to the ground case; likewise the reduct on ground programs is defined as expected. Finally, a set I of ground atoms is an *answer set* of a safe non-ground program P iff I is a subset-minimal set satisfying $\mathsf{G}(P, C_P)^I$, where C_P denotes the so-called active domain of P, i.e., the set of all constant symbols in P (if no such constant symbol exists, $C_P = \{c\}$ with an arbitrary constant symbol c). Equivalence notions for non-ground programs are defined as expected as well. In this work, whenever we speak of a *program*, we mean ground programs unless specified otherwise.

3 Definition and characterization

We now define the novel equivalence notion we are interested in here.

Definition 1 Given an alphabet $\mathcal{A} \subseteq \mathcal{U}$, we say that two programs P and Q are *rule-equivalent relative to* $\mathcal{A}$ and write $P \equiv_{\mathcal{A}} Q$ if for any set of rules R composed of

1. arbitrary proper rules and
2. facts $S \subseteq \mathcal{A}$,

$\mathrm{AS}(P \cup R) = \mathrm{AS}(Q \cup R)$ holds. If $P \equiv_{\mathcal{A}} Q$ holds for $\mathcal{A} = \emptyset$, we occasionally say that P and Q are *proper-rule-equivalent*. Whenever we leave $\mathcal{A}$ unspecified, we simply talk about *rule equivalence* between programs.

The main idea is that we assume knowledge about facts from $\mathcal{U} \setminus \mathcal{A}$ as already fixed (i.e., they will not be altered in the context R) while no further restriction is imposed in the equivalence test.

By definition, strong equivalence implies rule equivalence relative to any $\mathcal{A}$. If we set $\mathcal{A} = \mathcal{U}$ the notions coincide. In the other extreme case, $\mathcal{A} = \emptyset$, no facts will be added to the two programs under consideration. This particular notion of rule equivalence thus can be seen as an antipode to uniform equivalence. Hence, it is a natural question whether rule and uniform equivalence are incomparable notions.

Example 1 Let us for the moment fix $\mathcal{U} = \{a, b\}$ and consider the following programs:

$$\begin{aligned} P &= \{a \leftarrow b, \quad \leftarrow \mathit{not}\ a, \quad \leftarrow \mathit{not}\ b\} \\ Q &= \{a \leftarrow, \quad \leftarrow \mathit{not}\ b\} \\ R &= \{a \leftarrow b, \quad b \leftarrow a, \quad \leftarrow \mathit{not}\ a, \quad \leftarrow \mathit{not}\ b\} \end{aligned}$$

Their SE-models [1] are as follows:

$$\begin{aligned} \mathrm{SE}(P) &= \{(ab, ab), (a, ab), (\emptyset, ab)\} \\ \mathrm{SE}(Q) &= \{(ab, ab), (a, ab)\} \\ \mathrm{SE}(R) &= \{(ab, ab), (\emptyset, ab)\} \end{aligned}$$

By definition of UE-models $\mathrm{UE}(P) = \mathrm{UE}(Q)$, and thus P and Q are uniformly equivalent. However, they are not rule-equivalent relative to $\mathcal{A}$, even for $\mathcal{A} = \emptyset$. Consider $S = \{b \leftarrow a\}$. As is easily checked, $\mathrm{AS}(P \cup S) = \emptyset$ while $\mathrm{AS}(Q \cup S) = \{\{a, b\}\}$. On the other hand, P and R are not uniformly equivalent ($\mathrm{UE}(P) = \{(ab, ab), (a, ab)\} \neq \mathrm{UE}(R) = \{(ab, ab), (\emptyset, ab)\}$), while they are rule-equivalent relative to $\mathcal{A}$ with $\mathcal{A} \subseteq \{b\}$. We will see next how the latter result can be verified due to a suitable characterization. ◇

Also, observe that the notion of rule equivalence is *not* captured by the general $(\mathcal{A}, \mathcal{B})$ equivalence framework of [10], since our notion allows for any atoms in heads and bodies of proper rules, but restricts the atoms in heads only for facts, i.e., rules with empty bodies.

We now continue with the characterization for our main result.

Definition 2 An SE-model (X, Y) of a program P is called an *$\mathcal{A}$-RE-model* of P if:

(a) $(X = \emptyset$ and$(Y \cap \mathcal{A}) \neq \emptyset)$ or
(b) $(\emptyset, Y) \notin \mathrm{SE}(P)$ or
(c) $(X \cap \mathcal{A}) \neq \emptyset$ or
(d) $Y = \emptyset$.

[1] Within SE-models, we denote interpretations $\{a_1, \ldots, a_n\}$ by juxtaposition $a_1 \cdots a_n$ of their elements.

We write $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$ to indicate that (X, Y) is an $\mathcal{A}$-RE-model of P.

For the case $\mathcal{A} = \emptyset$ we can give a simpler characterization.

Definition 3 An SE-model (X, Y) of a program P is called an *RE-model* if $(\emptyset, Y) \notin \mathrm{SE}(P)$ holds whenever $Y \neq \emptyset$. We write $(X, Y) \in \mathrm{RE}(P)$ to indicate that (X, Y) is an RE-model of P.

Lemma 1 *For any program P the following relations hold:*

1. $RE(P) = RE^{\emptyset}(P)$
2. $SE(P) = RE^{\mathcal{U}}(P)$.

Proof 1 is straightforward from Definition 2. For 2, we observe the following implication from Definition 2: $(X, Y) \in \mathrm{RE}^{\mathcal{U}}(P)$ iff $(X, Y) \in \mathrm{SE}(P)$ and ((a) ($X = \emptyset$ and $Y \neq \emptyset$) or (b) $(\emptyset, Y) \notin \mathrm{SE}(P)$ or (c) $X \neq \emptyset$ or (d) $Y = \emptyset$) iff $(X, Y) \in \mathrm{SE}(P)$. □

The following close relationship between SE-models and RE-models is also immediate.

Lemma 2 *Let P be a program such that $(\emptyset, Y) \in SE(P)$ implies $Y = \emptyset$. Then, $SE(P) = RE(P)$.*

Proof By Definition 3, if there is no SE-model of the form $(\emptyset, Y)$, each SE-model of P becomes an RE-model of P. On the other hand, if $(\emptyset, \emptyset) \in \mathrm{SE}(P)$, observe that for each $(Z, Z) \in \mathrm{SE}(P)$ also $(\emptyset, Z) \in \mathrm{SE}(P)$ (by definition of SE-models and since $P^Z \subseteq P^{\emptyset}$). By assumption, we then have $\mathrm{SE}(P) = \{(\emptyset, \emptyset)\}$ and by Definition 3, $\mathrm{SE}(P) = \mathrm{RE}(P)$ also in this case. □

As a final simple observation, we have the following result that generalizes the well-known property of SE-models that $(X, Y) \in \mathrm{SE}(P)$ implies $(Y, Y) \in \mathrm{SE}(P)$ for any program P.

Lemma 3 *For any $\mathcal{A} \subseteq \mathcal{U}$ and program P: $(X, Y) \in RE^{\mathcal{A}}(P)$ implies $(Y, Y) \in RE^{\mathcal{A}}(P)$.*

Proof Towards a contradiction, assume $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$ and $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$. From $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, $(X, Y) \in \mathrm{SE}(P)$ and thus also $(Y, Y) \in \mathrm{SE}(P)$. Hence, in order to have $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$, we need, in particular, $Y \neq \emptyset$, $(Y \cap \mathcal{A}) = \emptyset$, and $(\emptyset, Y) \in \mathrm{SE}(P)$. We conclude that $(X \cap \mathcal{A}) = \emptyset$, which together with the above observation contradicts $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$ by definition of $\mathcal{A}$-RE-models. □

We now present our main result. In the spirit of the seminal results for strong equivalence [2, 3] we not only show that our characterization decides the equivalence notion but also that the equivalence notion boils down to unary rules.

Theorem 1 *For any programs P, Q, and alphabet $\mathcal{A} \subseteq \mathcal{U}$, the following statements are equivalent:*

(i) *P and Q are rule-equivalent relative to $\mathcal{A}$.*
(ii) *For any set R of facts from $\mathcal{A}$ and proper unary rules, $AS(P \cup R) = AS(Q \cup R)$.*
(iii) *$RE^{\mathcal{A}}(P) = RE^{\mathcal{A}}(Q)$.*

Proof (i) $\Rightarrow$ (ii) is clear.

$(ii) \Rightarrow (iii)$. W.l.o.g. suppose $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P) \setminus \mathrm{RE}^{\mathcal{A}}(Q)$. First consider the case $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$. Then we define $R = (Y \cap \mathcal{A}) \cup \{a \leftarrow b \mid a, b \in Y\}$. We show $Y \in \mathrm{AS}(P \cup R)$ but $Y \notin \mathrm{AS}(Q \cup R)$. For the former observe that $Y \models P \cup R$ (since $(X, Y) \in \mathrm{SE}(P)$). If $Y = \emptyset$ or $Y \cap \mathcal{A} \neq \emptyset$, then $Y \in \mathrm{AS}(P \cup R)$ as no proper subset of Y would satisfy R^Y. On the other hand, if $Y \neq \emptyset$ and $Y \cap \mathcal{A} = \emptyset$ hold, $\emptyset$ is the only proper subset of Y satisfying R^Y. By Lemma 3, $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, and by definition of RE-models (b) applies and we obtain $(\emptyset, Y) \notin \mathrm{SE}(P)$. Thus, $\emptyset$ is not model of $P^Y \cup R = (P \cup R)^Y$. This shows $Y \in \mathrm{AS}(P \cup R)$. To see that $Y \notin \mathrm{AS}(Q \cup R)$, we distinguish the two possible reasons for $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$: First, $(Y, Y) \notin \mathrm{SE}(Q)$. Then, $Y \not\models Q$. Otherwise, we have

(a) $(Y \neq \emptyset$ or$Y \cap \mathcal{A} = \emptyset)$ and
(b) $(\emptyset, Y) \in \mathrm{SE}(Q)$ and
(c) $(Y \cap \mathcal{A}) = \emptyset$ and
(d) $Y \neq \emptyset$.

Thus, in particular (b), (c) and (d). We observe that $\emptyset \models Q^Y \cup R = (Q \cup R)^Y$. For both cases, thus $Y \notin \mathrm{AS}(Q \cup R)$.

It remains to consider the case $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$. Recall that we have $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$ and $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, which implies $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$ and $X \subset Y$.

Suppose $X \cap \mathcal{A} = \emptyset$. Since $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, $(X, Y) \in \mathrm{SE}(P)$ holds and

(a1) $(X = \emptyset$ and$(Y \cap \mathcal{A}) \neq \emptyset)$ or
(b1) $(\emptyset, Y) \notin \mathrm{SE}(P)$.

Since $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$ either $(X, Y) \notin \mathrm{SE}(Q)$ or jointly:

(a2) $(X \neq \emptyset$ or$(Y \cap \mathcal{A}) = \emptyset)$ and
(b2) $(\emptyset, Y) \in \mathrm{SE}(Q)$.

Suppose (a1) holds. Then (a2) is false and thus $(X, Y) \notin \mathrm{SE}(Q)$ with $X = \emptyset$. We can take $R = \{a \leftarrow b \mid a, b \in Y\}$ and obtain $Y \in \mathrm{AS}(Q \cup R) \setminus \mathrm{AS}(P \cup R)$. Otherwise, we have $(\emptyset, Y) \notin \mathrm{SE}(P)$, and at least one of $(\emptyset, Y) \in \mathrm{SE}(Q)$ and $(X, Y) \notin \mathrm{SE}(Q)$. For $(\emptyset, Y) \in \mathrm{SE}(Q)$, take $R = \{a \leftarrow b \mid a, b \in Y\}$ as before. Now however, $Y \in \mathrm{AS}(P \cup R) \setminus \mathrm{AS}(Q \cup R)$. For the case $(\emptyset, Y) \notin \mathrm{SE}(Q)$ and $(X, Y) \notin \mathrm{SE}(Q)$, take

$$R = \{a \leftarrow b \mid a, b \in X\} \cup \{c \leftarrow d \mid c \in Y, d \in Y \setminus X\}.$$

Now, $Y \in \mathrm{AS}(Q \cup R) \setminus \mathrm{AS}(P \cup R)$.

It remains to consider the case $X \cap \mathcal{A} \neq \emptyset$. Here we take

$$R = (X \cap \mathcal{A}) \cup \{a \leftarrow b \mid a, b \in X\} \cup \{c \leftarrow d \mid c \in Y, d \in Y \setminus X\}$$

and observe that the only models of R being a subset of Y are Y and X. We already know $(X, Y) \in \mathrm{SE}(P)$. From $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$ and $(X \cap \mathcal{A}) \neq \emptyset$, we get $(X, Y) \notin \mathrm{SE}(Q)$. This suffices to see that $Y \in \mathrm{AS}(Q \cup R) \setminus \mathrm{AS}(P \cup R)$.

(iii) $\Rightarrow$ (i). Assume the existence of R being a set of proper rules and facts $F \subseteq \mathcal{A}$, such that w.l.o.g. $Y \in \mathrm{AS}(P \cup R)$ and $Y \notin \mathrm{AS}(Q \cup R)$. From the former we get $Y \models P$ (hence, $(Y, Y) \in \mathrm{SE}(P)$) and $Y \models R$.

We first show $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$. Towards a contradiction, suppose this is not the case. By definition of $\mathcal{A}$-RE-models, we then have $(Y \cap \mathcal{A}) = \emptyset$, $(\emptyset, Y) \in \mathrm{SE}(P)$, $Y \neq \emptyset$. From the former we observe that R must not contain facts (otherwise $Y \not\models R$), hence $\emptyset$ is a model of R^Y. Together with $(\emptyset, Y) \in \mathrm{SE}(P)$, $\emptyset \models (P \cup R)^Y$. Contradiction to $Y \in \mathrm{AS}(P \cup R)$.

Hence $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, and we proceed by distinguishing two cases.

1. $Y \not\models Q \cup R$. Since we already have seen $Y \models R$, this amounts to $Y \not\models Q$. It follows that $(Y, Y) \notin \mathrm{SE}(Q)$, and consequently, $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$.
2. $Y \models Q$ and some $X \subset Y$ is model of $(Q \cup R)^Y = Q^Y \cup R^Y$; hence $(X, Y) \in \mathrm{SE}(Q)$; and moreover, $(X, Y) \notin \mathrm{SE}(P)$ – otherwise $Y \in \mathrm{AS}(P \cup R)$ cannot hold. Since $(X, Y) \notin \mathrm{SE}(P)$, $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$. Also recall that $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$.

   In case $(X, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$ or $(Y, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$, we immediately obtain $\mathrm{RE}^{\mathcal{A}}(P) \neq \mathrm{RE}^{\mathcal{A}}(Q)$, so suppose $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$ and $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$. Since $(X, Y) \in \mathrm{SE}(Q)$ and $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$ the following jointly hold:

   (a) $X \neq \emptyset$ or $(Y \cap \mathcal{A}) = \emptyset$
   (b) $(\emptyset, Y) \in \mathrm{SE}(Q)$
   (c) $(X \cap \mathcal{A}) = \emptyset$

   Then, $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$ must be due to $(Y \cap \mathcal{A}) \neq \emptyset$, which implies $(\emptyset, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$ by Definition 2 together with (b). We already have seen that $X \models R^Y$. Together with (c) $X \cap \mathcal{A} = \emptyset$, we can conclude that R must not contain facts (no facts from X are allowed due to (c), other facts violate $X \models R^Y$). It follows that $\emptyset$ is a model of R. We conclude $(\emptyset, Y) \notin \mathrm{SE}(P)$ – otherwise $Y \in \mathrm{AS}(P \cup R)$ cannot hold. Consequently, $(\emptyset, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$. Hence, for the case that $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(P)$, $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$, $(Y, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$, $(X, Y) \notin \mathrm{RE}^{\mathcal{A}}(Q)$, we have shown that $(\emptyset, Y) \notin \mathrm{RE}^{\mathcal{A}}(P)$ and $(\emptyset, Y) \in \mathrm{RE}^{\mathcal{A}}(Q)$. □

4 Properties of rule equivalence

In this section, we compare the notions of strong, uniform, and proper-rule equivalence. Let us first proceed with our example from above.

Example 2 Recall programs P, Q, R from Example 1. We already have observed that P and Q are not proper-rule-equivalent. In fact, the RE-models according to Definition 3 are given by $\mathrm{RE}(P) = \emptyset$ and $\mathrm{RE}(Q) = \mathrm{SE}(Q) = \{(ab, ab), (a, ab)\}$. On the other hand, we have claimed that P and R are rule-equivalent relative to $\mathcal{A}$ with $\mathcal{A} \subseteq \{b\}$. We can now verify this by observing $\mathrm{RE}(R) = \emptyset$. If we now set $\mathcal{A} = \{b\}$, we observe that $\mathrm{RE}^{\mathcal{A}}(P) = \{(ab, ab), (\emptyset, ab)\} = \mathrm{RE}^{\mathcal{A}}(R)$. For $\mathcal{A} \supseteq \{a\}$ however, we observe $\mathrm{RE}^{\mathcal{A}}(P) = \mathrm{SE}(P) \neq \mathrm{SE}(R) = \mathrm{RE}^{\mathcal{A}}(R)$, since in the case of P, condition (c) of Definition 2 applies for the SE-model (a, ab). ◇

Let us say that two notions of equivalence $\equiv$ and $\equiv'$ are incomparable if there exist programs P, P', Q, Q', such that $P \equiv Q$ and $P \not\equiv' Q$, and moreover $P' \not\equiv Q'$ and $P' \equiv' Q'$. Our examples thus imply the following result.

Theorem 2 *Proper-rule equivalence and uniform equivalence are incomparable notions; this holds already for normal programs.*

Interestingly, if we disallow negation, uniform equivalence turns out to be a strictly stronger notion than proper-rule equivalence.

Theorem 3 *For positive programs, uniform equivalence implies proper-rule equivalence, but proper-rule equivalence does not imply uniform equivalence.*

Proof Recall that for positive programs, strong and uniform equivalence coincide [9], thus the first direction is obvious. On the other hand, program $\{a \leftarrow b;\ b \leftarrow a\}$ can be seen to be proper-rule-equivalent to $\{a \leftarrow c;\ c \leftarrow a\}$, but these two programs are not uniformly equivalent. □

As a final result in this section, we define one simple class of programs for which rule equivalence coincides with strong equivalence, but where uniform equivalence remains a weaker concept.

Definition 4 Call a program P *factual* if it contains at least one disjunctive fact, i.e., at least one rule with empty body.

It is rather obvious that programs with at least one fact a, are able to "simulate" the presence of further facts b via proper rules $b \leftarrow a$. The forthcoming result links this observation to equivalence notions.

Theorem 4 *For factual programs, proper-rule equivalence implies strong equivalence (and thus uniform equivalence), but uniform equivalence does not imply proper-rule equivalence.*

Proof The first direction is due to the easy observation that factual programs do not have SE-models of the form $(\emptyset, Y)$ and thus by Lemma 2, SE-models and RE-models coincide. For the second direction consider programs $\{a \leftarrow;\ b \vee c \leftarrow\}$ and $\{a \leftarrow;\ b \leftarrow \mathit{not}\ c;\ c \leftarrow \mathit{not}\ b\}$. The programs are uniformly equivalent but neither strongly nor rule-equivalent; in fact, just add $R = \{b \leftarrow c, c \leftarrow b\}$. □

We observe that the above counter-example also shows that shifting atoms from the head into the body of a rule is not a faithful manipulation w.r.t. rule equivalence, while it is known that shifting is faithful w.r.t. uniform equivalence [9].

5 Complexity and program simplification

We first study the following decision problem (RE-equivalence): Given disjunctive programs P, Q and set of propositional atoms $\mathcal{A}$, are P and Q rule-equivalent relative to $\mathcal{A}$?

Lemma 4 *For a given program P, an alphabet $\mathcal{A} \subseteq \mathcal{U}$ and a pair of interpretations X, Y, checking whether (X, Y) is an $\mathcal{A}$-RE-model of P is feasible in polynomial time.*

Proof Verifying whether (X, Y) is SE-model of P is known to be tractable. Likewise checking conditions (a)–(d) from Definition 2 are all easy tests. □

Theorem 5 *RE-equivalence is* coNP*-complete. Hardness holds even for $\mathcal{A}$ being empty, P positive or normal and Q Horn.*

Proof We show hardness by the following construction borrowed from the proof of Theorem 6.17 in [9]. We reduce from UNSAT: Let $\psi = \bigwedge_{i=1}^{n} C_i$ be a Boolean formula in CNF over a nonempty set of atoms X. For each $x \in X$, let $\bar{x}$ be a fresh atom, and consider the positive program

$$P = \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \{\leftarrow \overline{C}_i \mid 1 \leq i \leq n\},$$

where $\overline{C}_i$ is the conjunction obtained from $\neg C_i$ after applying DeMorgan's law, and replacing negative literals $\neg x$ by $\bar{x}$. For each $x \in X$, the purpose of the rules $x \vee \bar{x} \leftarrow$ and $\leftarrow x, \bar{x}$ in P is to make exactly one of x and $\bar{x}$ true in every answer set, and the remaining rules make sure that, for each answer set, the elements of X that are true form a model of ψ. Moreover, let

$$Q = \{\bot \leftarrow\},$$

and $\mathcal{A} = \emptyset$. Since Q has no SE-model, it cannot have any RE-model as well. On the other hand, if ψ is unsatisfiable, then P has no RE-model because it has no classical model, and if ψ is satisfiable, then P has an RE-model because P has a classical model M and P^M has no empty model. Thus, $P \equiv_{\mathcal{A}} Q$ iff ψ is unsatisfiable. The same holds if we replace in P each disjunctive fact $x \vee \bar{x} \leftarrow$ by two normal rules $x \leftarrow \mathit{not}\ \bar{x}$, $\bar{x} \leftarrow \mathit{not}\ x$. Since the reduction is polynomial the assertion follows.

Membership can be seen via the following algorithm for the complementary problem. Thanks to Theorem 1 we know that P and Q are not rule-equivalent relative to $\mathcal{A}$ iff they possess different $\mathcal{A}$-RE-models. Thus, we guess a pair (X, Y) over the atoms from $P \cup Q$ and check whether it is $\mathcal{A}$-RE-model of exactly one program. Due to Lemma 4, this can be done polynomial time. This shows NP-membership for the complementary problem of RE-equivalence. □

Thus the complexity for checking rule equivalence matches the one of strong equivalence and is easier than uniform equivalence, which is known to Π_2^P-complete. However, it is notable that membership in coNP also holds in the relativized case of rule equivalence for arbitrary $\mathcal{A}$, while strong equivalence relative to $\mathcal{A}$ yields an increase in complexity up to Π_2^P. For an overview of the mentioned known complexity results, we refer to [9].

As a second contribution in this section, we are interested in the following problem: Given a program P, does there exist a program Q from an easier class of programs, such that P and Q are rule-equivalent? We will thus complement results in [16] for strong and uniform equivalence. Since this particular problem (to the best of our knowledge) has not been investigated for relativized notions of strong and uniform equivalence, we will restrict ourselves to the unrelativized notion of rule equivalence, i.e., proper-rule equivalence, here as well. Let us first recall some concepts and results from [16].[2]

Definition 5 A set S of pairs of interpretation (X, Y) such that $X \subseteq Y$ is called

- *closed under here-intersection* if for any $(X, Y), (X', Y) \in S$, also $(X \cap X', Y) \in S$;
- *closed under there-intersection* if for any $(X, X), (Y, Y) \in S$, also $(X \cap Y, X \cap Y) \in S$;
- *here-total* if $(X, Y) \in S$ implies $(X, X) \in S$.

In [16] it is shown that for any program P there exists a uniformly equivalent normal program, but for strong equivalence this holds if and only if the SE-models of P are closed under here-intersection. In particular, the following result is shown in [16].

Proposition 1 *Let S be a set of pairs of interpretations such that for each $(X, Y) \in S$ it holds that (i) $X \subseteq Y$, (ii) $(Y, Y) \in S$, (iii) $(X, Z) \in S$ for any $(Z, Z) \in S$ with $Y \subseteq Z$. Moreover, let S be closed under here-intersection. Then there exists a normal program Q such that $SE(Q) = S$.*

[2] We use the terminology from [16], which is motivated by the fact that SE-models are closely related to the logic of here-and-there as used in [2].

We next show that proper-rule equivalence follows the same pattern as strong equivalence.

Theorem 6 *Let P a program. Then, there exists a normal program Q such that P and Q are proper-rule-equivalent iff $\mathrm{RE}(P)$ is closed under here-intersection.*

Proof We first show that $\mathrm{RE}(P)$ satisfies the following properties, for any program P:

(i) for each $(X, Y) \in \mathrm{RE}(P)$ it holds that $X \subseteq Y$;
(ii) for each $(X, Y) \in \mathrm{RE}(P)$, also $(Y, Y) \in \mathrm{RE}(P)$;
(iii) for each $(X, Y), (Z, Z) \in \mathrm{RE}(P)$ with $Y \subseteq Z$, also $(X, Z) \in \mathrm{RE}(P)$;
(iv) $(\emptyset, Y) \in \mathrm{RE}(P)$ implies $Y = \emptyset$.

(i) holds by the fact that the RE-models are a subset of the SE-models; (ii) is by Lemma 3; (iii) is trivial for $Z = \emptyset$ and can be shown as follows for nonempty Z: first, $(X, Y) \in \mathrm{RE}(P)$ implies $(X, Y) \in \mathrm{SE}(P)$; second $(Z, Z) \in \mathrm{RE}(P)$ implies $(\emptyset, Z) \notin \mathrm{SE}(P)$ and $(Z, Z) \in \mathrm{SE}(P)$; $(X, Z) \in \mathrm{SE}(P)$ follows by definition of SE-models and since $P^Z \subseteq P^Y$; finally, since $(\emptyset, Z) \notin \mathrm{SE}(P)$, $(X, Z) \in \mathrm{RE}(P)$ holds; (iv) is by definition of RE-models.

For the if-direction of the claim, we proceed as follows. Since $\mathrm{RE}(P)$ satisfies (i)–(iii) and is closed under here-intersection, Proposition 1 can be applied. Thus, there exists a normal program Q such that $\mathrm{SE}(Q) = \mathrm{RE}(P)$. Hence $\mathrm{SE}(Q)$ satisfies (iv) and by Lemma 2, $\mathrm{RE}(Q) = \mathrm{SE}(Q)$ follows. We arrive at $\mathrm{RE}(P) = \mathrm{RE}(Q)$. By Theorem 1, P and Q are proper-rule-equivalent.

For the only-if direction, assume towards a contradiction that there exists a normal program Q such that $\mathrm{RE}(P) = \mathrm{RE}(Q)$, but $\mathrm{RE}(P)$ is not closed under here-intersection. Hence, there exist $(X, Y), (X', Y) \in \mathrm{RE}(P)$ such that $(X \cap X', Y) \notin \mathrm{RE}(P)$. By the definition of RE-models and the assumption that $\mathrm{RE}(P) = \mathrm{RE}(Q)$, we observe that $(X, Y), (X', Y) \in \mathrm{SE}(Q)$. Since Q is normal, $(X \cap X', Y) \in \mathrm{SE}(Q)$ as well. Since $(X \cap X', Y) \notin \mathrm{RE}(Q)$, $(\emptyset, Y) \in \mathrm{SE}(Q)$ by definition of RE-models. But then, neither $(X, Y) \in \mathrm{RE}(Q)$ nor $(X', Y) \in \mathrm{RE}(Q)$. Contradiction to $\mathrm{RE}(P) = \mathrm{RE}(Q)$. □

The corresponding complexity result is as follows.

Theorem 7 *Given a program P, checking whether there exists a normal program Q that is proper-rule-equivalent to P is* coNP*-complete.*

Proof Membership for the complementary problem is by guessing two pairs of interpretations (X, Y), (X', Y), and checking whether $(X, Y) \in \mathrm{RE}(P)$, $(X', Y) \in \mathrm{RE}(P)$, $(X \cap X', Y) \notin \mathrm{RE}(P)$. By Lemma 4 these checks are feasible in polynomial time, thus membership in coNP follows.

For hardness, we exploit the reduction from [16] for proving that checking whether the SE-models of a given program are closed under here-intersection is coNP-hard. They reduce from the problem of validity of a DNF formula. To this end, consider ϕ in 3-DNF, i.e., $\phi = \bigvee_{i=1}^{m} l_{i,1} \wedge l_{i,2} \wedge l_{i,3}$, and let

$$P = \{w \leftarrow l_{i,1}, l_{i,2}, l_{i,3} \mid 1 \leq i \leq m\} \cup \{a \vee b \leftarrow \mathit{not}\ w\},$$

where in P negative literals $l_{i,j} = \neg v$ are written as *not* v and a, b, and w are new atoms.

We show that $\mathrm{RE}(P)$ is closed under here-intersection iff ϕ is valid. Together with Theorem 6 and the fact that the reduction can be done in polynomial time, the claim follows.

Consider the case that ϕ is valid. Then, $w \in Y$ holds for every model Y of P. Hence, for each $(Y, Y) \in \mathrm{SE}(P)$, P^Y is Horn. As models of Horn programs are closed under intersection, $\mathrm{SE}(P)$ is closed under here-intersection. Then, also $\mathrm{RE}(P)$ is closed under here-intersection. This can be seen as follows: Let $(X, Y), (X', Y) \in \mathrm{RE}(P)$. It follows that $(\emptyset, Y) \notin \mathrm{SE}(P)$. Moreover, since $(X, Y), (X', Y) \in \mathrm{SE}(P)$, $(X \cap X', Y) \in \mathrm{SE}(P)$, and by the fact that $(\emptyset, Y) \notin \mathrm{SE}(P)$, $(X \cap X', Y) \in \mathrm{RE}(P)$.

If ϕ is not valid we have some $(Y, Y) \in \mathrm{SE}(P)$ with $w \notin Y$, and $\{a, b\} \subseteq Y$. One can check that then $(Y \setminus \{a\}, Y) \in \mathrm{SE}(P)$, $(Y \setminus \{b\}, Y) \in \mathrm{SE}(P)$, but $(Y \setminus \{a, b\}, Y) \notin \mathrm{SE}(P)$, and $(\emptyset, Y) \notin \mathrm{SE}(P)$. It follows that $(Y \setminus \{a\}, Y) \in \mathrm{RE}(P)$, $(Y \setminus \{b\}, Y) \in \mathrm{RE}(P)$, but $(Y \setminus \{a, b\}, Y) \notin \mathrm{RE}(P)$. Thus $\mathrm{RE}(P)$ is not closed under here-intersection. □

Concerning simplification into positive (resp. Horn) programs, [16] have shown that any program P such that $\mathrm{SE}(P)$ is here-total (resp. here-total and closed under there-intersection) possesses a strongly equivalent positive (resp. Horn) program as well as a uniformly equivalent positive (resp. Horn) program.

The corresponding results for proper-rule equivalence can be given accordingly.

Theorem 8 *Let P a program. Then, there exists a positive program Q such that P and Q are proper-rule-equivalent iff $RE(P)$ is here-total.*

Proof For the if-direction, we argue as in the proof of Theorem 6. We consider the positive program Q with $\mathrm{SE}(Q) = \mathrm{RE}(P)$; such a program exists due to the results on strongly equivalent positive programs (similar to Proposition 1), cf. [16]. Analogously, we can then deduce $\mathrm{RE}(Q) = \mathrm{SE}(Q)$ thanks to Lemma 2. This shows $\mathrm{RE}(P) = \mathrm{RE}(Q)$. Hence, P and Q are proper-rule-equivalent.

For the only-if direction, assume towards a contradiction that there exists a positive program Q such that $\mathrm{RE}(P) = \mathrm{RE}(Q)$, but $\mathrm{RE}(P)$ is not here-total. Hence, there exist $(X, Y) \in \mathrm{RE}(P)$ such that $(X, X) \notin \mathrm{RE}(P)$. By the definition of RE-models and the assumption that $\mathrm{RE}(P) = \mathrm{RE}(Q)$, we observe that $(X, Y) \in \mathrm{SE}(Q)$. Since Q is positive, also $(X, X) \in \mathrm{SE}(Q)$. Since $(X, X) \notin \mathrm{RE}(Q)$, we must have $(\emptyset, X) \in \mathrm{SE}(Q)$ by definition of RE-models. But then, $(X, X) \notin \mathrm{RE}(Q)$. Contradiction to $\mathrm{RE}(P) = \mathrm{RE}(Q)$. □

Concerning complexity of checking here-totality (and thus existence of an equivalent positive program), [16] has shown Π_2^P-completeness for UE-models and coNP-completeness for SE-models. Given the already established close connection between strong and proper-rule equivalence, the next result is not surprising.

Theorem 9 *Given a program P, checking whether there exists a positive program Q that is proper-rule-equivalent to P is* coNP*-complete.*

Proof Membership for the complementary problem is by guessing a pair (X, Y) of interpretations, and checking whether $(X, Y) \in \mathrm{RE}(P)$ and $(X, X) \notin \mathrm{RE}(P)$. Membership in coNP follows by Lemma 4.

To show coNP-hardness we reduce from UNSAT re-using ideas from the proof of Theorem 5. Let $\psi = \bigwedge_{i=1}^{n} C_i$ be a Boolean formula in CNF over atoms X. Let a and $\bar{c}$, where $c \in X$, be fresh atoms, and consider

$$P = \{x \vee \bar{x} \leftarrow;\ \leftarrow x, \bar{x} \mid x \in X\} \cup \{\leftarrow \overline{C}_i \mid 1 \leq i \leq n\} \cup \{a \leftarrow \mathit{not}\ a\},$$

where $\overline{C}_i$ is the conjunction obtained from $\neg C_i$ after applying DeMorgan's law, and replacing negative literals $\neg x$ by $\bar{x}$. We show that ψ is satisfiable iff RE(P) is not here-total).

If ψ is unsatisfiable, P has no SE-model and thus no RE-model. Here-totality of RE(P) is then trivially satisfied. If ψ is satisfiable, then each model of P and P^Y (Y being an arbitrary interpretation) contains at least some x or $\bar{x}$. From that SE(P) = RE(P) follows by definition. It remains to see that SE(P) is not here-total, which is due to the rule $a \leftarrow \mathit{not}\ a$ and yields to the fact that for any Y with $a \in Y$ that is model of P, $(Y \setminus \{a\}, Y) \in \mathrm{SE}(P)$ but $(Y \setminus \{a\}, Y \setminus \{a\}) \notin \mathrm{SE}(P)$. Since the reduction is polynomial, this proves coNP-hardness of deciding whether RE(P) is here-total. Together with Theorem 8, the result follows. □

Our final result concerns simplification to Horn programs. We omit here a detailed complexity analysis, although coNP-membership is similar to the previous proofs. However, we do not have a matching lower bound. Note that coNP-hardness for the according problem on strong equivalence has also been left open in [16].

Theorem 10 *Let P a program. Then, there exists a Horn program Q such that P and Q are proper-rule-equivalent iff RE(P) is here-total and closed under there-intersection.*

Proof The if-direction is analogous to the previous proofs.

For the only-if direction, assume towards a contradiction that there exists a Horn program Q such that RE(P) = RE(Q), but RE(P) is not jointly here-total and closed under there-intersection. We already know that if RE(P) is not here-total, then there exists no positive program which is proper-rule-equivalent to P; hence, there cannot be such a Horn program. Thus assume there exist $(X, X) \in \mathrm{RE}(P)$ and $(Y, Y) \in \mathrm{RE}(P)$, such that $(X \cap Y, X \cap Y) \notin \mathrm{RE}(P)$. By the definition of RE-models and the assumption that RE(P) = RE(Q), we observe that $(X, X), (Y, Y) \in \mathrm{SE}(Q)$. Since Q is Horn, also $(X \cap Y, X \cap Y) \in \mathrm{SE}(Q)$. Since $(X \cap Y, X \cap Y) \notin \mathrm{RE}(Q)$. we must have $(\emptyset, X \cap Y) \in \mathrm{SE}(Q)$. But then, $(\emptyset, X) \in \mathrm{SE}(Q)$ as well (since $Q^{(X \cap Y)} \subseteq Q^X$). By definition of RE-models $(X, X) \notin \mathrm{RE}(Q)$. Contradiction to RE($P$) = RE($Q$). □

6 Rule equivalence in the non-ground case

Our final result is concerned with the question whether rule equivalence remains decidable in the non-ground case of Answer-Set Programming. We emphasize that the answer to this question is not obvious. While strong equivalence is known to be decidable, uniform equivalence is, in fact, undecidable [17].

We will restrict ourselves here to the case of proper-rule equivalence, i.e., rule equivalence relative to $\mathcal{A} = \emptyset$. Thus, with some little abuse of notation, we will from now on refer to rule equivalence when we mean rule equivalence relative to $\mathcal{A} = \emptyset$. Our result is based on a transformation τ such that programs P and Q are rule-equivalent iff $\tau(P)$ and $\tau(Q)$ are strongly equivalent. We shall first introduce τ for the propositional case and then adapt the idea to the non-ground case.

6.1 Reducing rule equivalence to strong equivalence

For the sake of presentation, we first present τ as a translation from disjunctive programs (the class we have focused on in this paper) to the more general class of programs with

nested expressions [19]. In this class, rule bodies can be an arbitrary Boolean combination of atoms and default-negated atoms.[3] Concerning semantics, it is only the definition of the reduct that changes. Formally, given a nested logic program P and an interpretation I, define P^I as obtained from P by replacing every occurrence of a literal *not* a by $\bot$ if $a \in I$, and by $\top$ otherwise. As before, an interpretation I is an answer set of a program P iff it is a minimal model (w.r.t. set inclusion) of P^I. With the modified reduct at hand, SE-models are likewise easily generalized to nested logic programs.

We now turn to the expected behavior of the translation τ. In what follows, we let V be the set of atoms occurring in a program P and $\overline{V}$, V', $\overline{V}'$ be fresh copies of V.

Definition 6 A translation τ mapping disjunctive programs to programs with nested expressions satisfies the *RE-to-SE property* if the following holds for any program P:

1. for any $I \subseteq V$,
$$I \cup (\overline{V \setminus I}) \cup V' \cup \overline{V}' \models \tau(P)$$
2. for any $J \subseteq I \subseteq V$,
$$I \cup (\overline{V \setminus I}) \cup J' \cup (\overline{V \setminus J})' \models \tau(P)^{I \cup (\overline{V \setminus I}) \cup V' \cup \overline{V}'}$$
$$\Longleftrightarrow$$
$$(J, I) \in \mathrm{SE}(P) \text{ and either } (\emptyset, I) \notin \mathrm{SE}(P) \text{ or } I = \emptyset$$
3. all models of $\tau(P)$ and reducts of $\tau(P)$ are implicitly given above; formally for any X, Y such that $Y \models \tau(P)$ and $X \models \tau(P)^Y$ it holds that (i) Y is of the form $I_Y \cup V' \cup \overline{V}'$ for some $I \subseteq V$ with $I_Y = I \cup (\overline{V \setminus I})$; (ii) if $X \neq Y$ then X is of the form $I_Y \cup J' \cup (\overline{V \setminus J})'$ for some J such that $(J, I) \in \mathrm{SE}(P)$ and either $(\emptyset, I) \notin \mathrm{SE}(P)$ or $I = \emptyset$.

Intuitively, the idea is to embed all RE-models (J, I) of P as models of reducts of $\tau(P)$ using the unprimed atoms to express I and the primed ones to express J. More formally, for each RE-model (J, I) of P we will have that $I \cup (\overline{V \setminus I}) \cup J' \cup (\overline{V \setminus J})' \models \tau(P)^{I \cup (\overline{V \setminus I}) \cup V' \cup \overline{V}'}$. The reason why we need to express RE-models in this rather cumbersome way is that we need to express a negative test $(\emptyset, I) \notin \mathrm{SE}(P)$ with $\tau(P)$; although the test is conceptually easy, in order to express it via a logic program, we have to make use of a certain saturation[4] encoding to realize it.

For the forthcoming theorem, observe that the fact that P and Q contain exactly the same set of atoms is not a severe restriction, since we can always add a "tautological rule" $a \leftarrow a$ for a missing atom a.

Theorem 11 *Given programs P, Q both containing the same set of atoms V and function τ satisfying the RE-to-SE property, we have that P and Q are rule-equivalent iff $\tau(P)$ and $\tau(Q)$ are strongly equivalent.*

Proof (sketch) The proof follows from the following observations: (1) the total SE-models (Y, Y) of $\tau(P)$ and $\tau(Q)$ coincide; (2) for any $J, I \subseteq V$ we have by definition that

[3] In fact, the programs we use here form a proper subclass of programs compared to [19], where, e.g., also double negation is allowed. However, for our purpose it is sufficient to consider this weaker class.

[4] The concept of saturation refers to a programming technique, where reasons for a candidate answer set I to be ruled out are not explicitly stated via constraints, but in terms of rules which ensure that a certain model $J \subset I$ of the program's reduct with respect to I exists, see, e.g., [20].

$(J, I) \in \mathrm{RE}(R)$ iff $(X_J, Y_I) \in \mathrm{SE}(\tau(R))$ with $Y_I = I \cup (\overline{V \setminus I}) \cup V' \cup \overline{V}'$ and $X_J = I \cup (\overline{V \setminus I}) \cup J' \cup (\overline{V \setminus J})'$; (3) there are no other non-total SE-models of $\tau(P)$ or $\tau(Q)$ than those corresponding to an RE-model. □

We now actually give a translation τ satisfying the RE-to-SE property. Below we intuitively explain the functioning of the three parts.

Definition 7 Given P over V, we define

$$\tau(P) = \tau_G(P) \cup \tau_M(P) \cup \tau_R(P)$$

where

$$\begin{aligned}
\tau_G(P) &= \{v \vee \overline{v} \leftarrow;\ \leftarrow v, \overline{v};\ \leftarrow \mathit{not}\ v';\ \leftarrow \mathit{not}\ \overline{v}'; && (2)\\
&\quad v' \vee \overline{v}' \leftarrow;\ v \leftarrow v' \mid v \in V\} && (3)\\
\tau_M(P) &= \{v' \leftarrow B^+(r), \overline{B^-(r)}, \overline{H(r)}; && (4)\\
&\quad \overline{v}' \leftarrow B^+(r), \overline{B^-(r)}, \overline{H(r)}; && (5)\\
&\quad v' \leftarrow (B^+(r))', \overline{B^-(r)}, (\overline{H(r)})'; && (6)\\
&\quad \overline{v}' \leftarrow (B^+(r))', \overline{B^-(r)}, (\overline{H(r)})' \mid r \in P, v \in V\} && (7)\\
\tau_R(P) &= \{v' \leftarrow w, \bigwedge_{r \in P, B^+(r) = \emptyset} \bigvee_{y \in B^-(r)} y; && (8)\\
&\quad \overline{v}' \leftarrow w, \bigwedge_{r \in P, B^+(r) = \emptyset} \bigvee_{y \in B^-(r)} y \mid v, w \in V\} && (9)
\end{aligned}$$

$\tau_G(P)$ is responsible for the guess of interpretations; (2) forces the classical models of $\tau(P)$ to be of the required form $I_Y \cup V' \cup \overline{V}'$ for $I \subseteq V$ with $I_Y = I \cup (\overline{V \setminus I})$. (3) restricts the proper submodels of reducts $\tau(P)^{I_Y \cup V' \cup \overline{V}'}$ to be of the form $J \cup (\overline{V \setminus J}) \cup I_Y$ for $J \subseteq I$. $\tau_M(P)$ is responsible for eliminating models of the latter kind in case (J, I) is not SE-model of P, i.e., either $I \not\models P$ (rules (4)+(5)) or $J \not\models P^I$ (rules (6)+(7)). The final rules from $\tau_R(P)$ now do the following. They eliminate reduct models $J \cup (\overline{V \setminus J}) \cup I_Y$ in case $J \neq \emptyset$ and $(\emptyset, I) \in \mathrm{SE}(P)$. To this end, in rules (8)+(9) we use atom w to make the rules applicable only if $J \neq \emptyset$. Moreover, in order to have $(\emptyset, I) \in \mathrm{SE}(P)$ we need that each rule r of P with an empty positive body, i.e., $B^+(r) = \emptyset$, is removed in the construction of P^I, i.e., for each such r, $B^-(r) \cap I \neq \emptyset$ has to hold. We model this via the disjunction in the rule bodies of rules (8)+(9).

Lemma 5 *Translation τ from Definition 7 satisfies the RE-to-SE property.*

In fact, it is only $\tau_R(P)$ which uses non-standard rules, i.e., rules with nested Boolean expressions. As known from [19], these rule can be transformed to standard rules without changing the SE-models, by applying standard laws of distributivity within rule bodies and a so-called splitting rule

$$\{A \leftarrow B \vee C\} \ \Rightarrow\ \{A \leftarrow B;\ A \leftarrow C\}.$$

Applying this transformation rules to $\tau_R(P)$ sufficiently often, we end up with a standard disjunctive logic program. This can be given as follows: Let $P_0 = \{r_1, \ldots, r_k\}$ be the set of

all rules of P with empty positive body and $P_0^- = \{\langle y_1, \ldots, y_k\rangle \mid y_1 \in B^-(r_1), \ldots, y_k \in B^-(r_k)\}$ any selection of negative body atoms of rules in P_0. We then can rewrite $\tau_R(P)$ as:

$$\begin{aligned}\tau'_R(P) = \{\ & v' \leftarrow w, y_1, \ldots, y_k; \\ & \overline{v}' \leftarrow w, y_1, \ldots, y_k \mid v, w \in V, \langle y_1, \ldots y_k\rangle \in P_0^-\}\end{aligned}$$

Let $\tau'(\cdot)$ be the result of replacing in $\tau(\cdot)$ the program $\tau_R(\cdot)$ by $\tau'_R(\cdot)$.

Corollary 1 *For any finite programs P, Q over the same atoms V, it holds that P and Q are rule-equivalent iff $\tau'(P)$ and $\tau'(Q)$ are strongly equivalent.*

We observe that $\tau'(P)$ is exponential in the size of the original program P (whereas $\tau(P)$ is efficiently constructible). Although our complexity results suggest that better translations from disjunctive programs (being rule-equivalent to each other) to disjunctive programs (being strongly equivalent to each other) might exist, we leave this question of an efficient reduction for future work. Indeed, since we are mainly interested in a reduction for the non-ground case in order to show decidability, the only issue that counts is that we have a computable translation.

6.2 Decidability in the non-ground case

For strong equivalence it is possible that the context programs R extend the active domain from C_P to $C_{P\cup R}$. This also holds for rule equivalence where a context program R can contain any rule of the form $p(c) \leftarrow q(X)$ where $c \in \mathcal{C}$ is any constant not occurring in the programs that are compared to each other. Since $\mathcal{C}$ is not bounded, decidability is thus not trivial.

We now define the translation for the non-ground case. We use some abbreviations: $dom(X, n)$ is a shorthand for the sequence $dom(X_1), \ldots, dom(X_n)$, and $p(X)$ denotes $p(X_1, \ldots, X_{n(p)})$. As before we use new predicate symbols $\overline{p}$, p', $\overline{p}'$. The main observation is that we have a new set of rules τ_D^* to collect all constant symbols occurring in the program plus in the possibly added rules of the equivalence test. The remaining parts basically just generalize the program for the propositional case from Definition 7 to the non-ground case, using the *dom* predicate to ensure the safety of rules.

Definition 8 Given a non-ground program P over predicates $\mathcal{Q}$, we define

$$\tau^*(P) = \tau_D^*(P) \cup \tau_G^*(P) \cup \tau_M^*(P) \cup \tau_R^*(P)$$

where

$$\begin{aligned}\tau_D^*(P) &= \{dom(X_i) \leftarrow p(X_1, \ldots, X_{n(p)}) \mid p \in \mathcal{Q}, 1 \le i \le n(p)\} \\ \tau_G^*(P) &= \{p(X) \vee \overline{p}(X) \leftarrow dom(X, n(p)); \\ &\quad \leftarrow p(X), \overline{p}(X); \\ &\quad \leftarrow \mathit{not}\ p'(X), dom(X, n(p)); \\ &\quad \leftarrow \mathit{not}\ \overline{p}'(X), dom(X, n(p)); \\ &\quad p'(X) \vee \overline{p}'(X) \leftarrow dom(X, n(p)); \\ &\quad p(X) \leftarrow p'(X) \mid p \in \mathcal{Q}\}\end{aligned}$$

$$\begin{aligned}
\tau^*_M(P) \;=\; \{&p'(X) \leftarrow B^+(r), \overline{B^-(r)}, \overline{H(r)}, dom(X, n(p));\\
&\overline{p}'(X) \leftarrow B^+(r), \overline{B^-(r)}, \overline{H(r)}, dom(X, n(p));\\
&p'(X) \leftarrow (B^+(r))', \overline{B^-(r)}, (\overline{H(r)})', dom(X, n(p));\\
&\overline{p}'(X) \leftarrow (B^+(r))', \overline{B^-(r)}, (\overline{H(r)})', dom(X, n(p)) \mid r \in P, p \in \mathcal{Q}\}\\
\tau^*_R(P) \;=\; \{&p'(X) \leftarrow w(Y), y_1, \ldots, y_k, dom(X, n(p));\\
&\overline{p}'(X) \leftarrow w(Y), y_1, \ldots, y_k, dom(X, n(p)) \mid p, w \in \mathcal{Q}, \langle y_1, \ldots y_k\rangle \in P_0^-\}
\end{aligned}$$

and $P_0^- = \{\langle y_1, \ldots y_k\rangle \mid y_1 \in B^-(r_1), \ldots, y_k \in B^-(r_k)\}$ is any selection of negative body atoms of rules in P with empty positive body.

As before, we mention that the forthcoming theorem is not restricted by the fact that the compared programs need to contain the same predicate and constant symbols. Again, adding tautological rules of the form $p(a) \leftarrow p(a)$ is possible without changing any equivalence notion.

Theorem 12 *Let P and Q be non-ground programs over the same set of predicate symbols and constants. Then, P and Q are rule-equivalent iff $\tau^*(P)$ and $\tau^*(Q)$ are strongly equivalent.*

Proof The proof idea relies on the fact the we can reduce the non-ground setting to the ground one which will allow us to employ Corollary 1. However, some care is needed to restrict ourselves to finite ground programs. Recall that we assumed an infinite set $\mathcal{C}$ of constant symbols (otherwise the rule equivalence problem would be decidable by trivial means).

For the if-direction, suppose $\tau^*(P)$ and $\tau^*(Q)$ are not strongly equivalent. From the results in [17], it follows that then there is a finite $C \subseteq \mathcal{C}$ such that $\mathsf{G}(\tau^*(P), C)$ is not strongly equivalent to $\mathsf{G}(\tau^*(Q), C)$, and moreover that for the witnessing SE-model (X, Y) in the symmetric difference $\mathrm{SE}(\mathsf{G}(\tau^*(P), C)) \triangle \mathrm{SE}(\mathsf{G}(\tau^*(Q), C))$ we can assume w.l.o.g. that all constants from C occur in the set of ground atoms Y. Let us now denote by P^* the program obtained from $\mathsf{G}(\tau^*(P) \setminus \tau^*_D(P), C)$ by removing all atoms of the form $dom(a)$. Analogously, define Q^*. Inspecting the usage of the dom atoms in $\tau^*(\cdot)$ and using the fact that all constants of C occur in atoms of Y, one can show that (X, Y) remains a witnessing SE-model, i.e., $(X, Y) \in \mathrm{SE}(P^*) \triangle \mathrm{SE}(Q^*)$. Moreover, observe that $P^* = \tau(\mathsf{G}(P, C))$ and $Q^* = \tau(\mathsf{G}(Q, C))$. We thus can now apply Corollary 1 and obtain that $\mathsf{G}(P, C)$ is not rule-equivalent to $\mathsf{G}(Q, C)$. From this observation, the result that P is not rule-equivalent to Q follows quite easily.

For the only-if direction, suppose P and Q are not rule-equivalent, hence there exists a set of proper non-ground rules R such that $\mathrm{AS}(P \cup R) \neq \mathrm{AS}(Q \cup R)$. Let C_0 be the set of constants occurring in P (and thus in Q) and C be the set of constants occurring in $P \cup R$ (and thus in $Q \cup R$). By definition of answer sets, we have $\mathrm{AS}(\mathsf{G}(P \cup R, C)) \neq \mathrm{AS}(\mathsf{G}(Q \cup R, C))$. Note that $\mathsf{G}(P \cup R, C) = \mathsf{G}(P, C) \cup \mathsf{G}(R, C)$ and likewise, $\mathsf{G}(Q \cup R, C) = \mathsf{G}(Q, C) \cup \mathsf{G}(R, C)$. Hence, $\mathsf{G}(P, C)$ and $\mathsf{G}(Q, C)$ are not rule-equivalent. In case C is not finite, we need some additional argument. Since $\mathsf{G}(P, C)$ and $\mathsf{G}(Q, C)$ are ground, we can employ our characterization from Theorem 1 and observe that there exists a pair (X, Y) which is RE-model of exactly one of the programs. Inspecting the definition of RE-models, one can then show that we can safely restrict C to C' containing C_0 plus k further constants where k is the maximal number of variables in the rules of P and Q. Thus C' is finite. Moreover, $(X|_{C'}, Y|_{C'})$ is then RE-model of exactly one of the

programs $\mathsf{G}(P, C')$ and $\mathsf{G}(Q, C')$, where $Z|_{C'}$ denotes the set of all ground atoms in Z which are built from constants in C' only. We conclude (again by Theorem 1) that $\mathsf{G}(P, C')$ and $\mathsf{G}(Q, C')$ are not rule-equivalent. In case C was finite, we continue with $C' = C$ and in both cases apply Corollary 1, which yields that $\tau(\mathsf{G}(P, C'))$ and $\tau(\mathsf{G}(Q, C'))$ are not strongly equivalent. Hence, there exists a pair (U, Z) that is SE-model of exactly one of the two programs. Let $D = \{dom(a) \mid a \in C'\}$. It can be shown that then $(U \cup D, Z \cup D)$ is SE-models of exactly one of the programs $\tau^*(\mathsf{G}(P, C'))$ and $\tau^*(\mathsf{G}(Q, C'))$. Moreover, $\tau^*(\mathsf{G}(P, C')) = \mathsf{G}(\tau^*(P, C'))$ and likewise, $\tau^*(\mathsf{G}(Q, C')) = \mathsf{G}(\tau^*(Q, C'))$. Thus $\mathsf{G}(\tau^*(P, C'))$ is not strongly equivalent to $\mathsf{G}(\tau^*(Q, C'))$. From this, it easily follows that $\tau^*(P)$ is not strongly equivalent to $\tau^*(Q)$. □

From the above result we can immediately conclude the following.

Theorem 13 *Deciding rule equivalence between non-ground programs is decidable.*

7 Conclusion

In this paper, we have studied an equivalence notion in Answer-Set Programming that has been overlooked in previous work. This notion weakens strong equivalence in the sense that programs are compared under scenarios where arbitrary proper rules are allowed to be added but the addition of facts is restricted, or even forbidden. Our results show that an SE-model-based characterization for this problem is quite involved but remains on the coNP complexity level, thus showing the same complexity as strong equivalence. This correspondence also carries over to non-ground programs, where we have shown decidability for (the unrelativized version of) rule equivalence.

Future work includes the following research questions. First, we aim for a mix of relativized strong equivalence and rule equivalence where one alphabet restricts the proper rules and a second alphabet restricts the facts in the potential context programs. Second, we are interested in finding better translations from rule equivalence to strong equivalence, avoiding the exponential blow-up we have witnessed in Section 6.2. Finally, we claim that decidability also holds for the relativized variants of rule equivalence.

Acknowledgments Open access funding provided by Austrian Science Fund (FWF). This work was supported by the Austrian Science Fund (FWF) projects P25607 and Y698.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Commun. ACM **54**(12), 92–103 (2011)
2. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**(4), 526–541 (2001)
3. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory Pract. Logic Program. **3**(4-5), 602–622 (2003)

4. Sagiv, Y.: Optimizing datalog programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 659–698. Morgan Kaufmann, Burlington (1988)
5. Eiter, T., Fink, M.: Uniform equivalence of logic programs under the stable model semantics. In: Palamidessi, C. (ed.) Proceedings of the 19th International Conference on Logic Programming (ICLP'03). number 2916 in LNCS, pp. 224–238. Springer, Berlin (2003)
6. Fink, M.: A general framework for equivalences in answer-set programming by countermodels in the logic of here-and-there. Theory Pract. Logic Program. **11**(2–3), 171–202 (2011)
7. Inoue, K., Sakama, C.: Equivalence of logic programs under upyears. In: Alferes, J.J., Leite, J.A. (eds.) Proceedings of the 9th European Conference on Logics in Artificial Intelligence, JELIA'04, vol. 3229 of LNCS, pp. 174–186. Springer, Berlin (2004)
8. Woltran, S.: Characterizations for relativized notions of equivalence in answer set programming. In: Alferes, J.J., Leite, J.A. (eds.) Proceedings of the 9th European Conference on Logics in Artificial Intelligence, JELIA'04, vol. 3229 of LNCS, pp. 161–173. Springer (2004)
9. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Trans. Comput. Log **8**(3), 17:1–17:53 (2007)
10. Woltran, S.: A common view on strong, uniform, and other notions of equivalence in answer-set programming. TPLP **8**(2), 217–234 (2008)
11. Truszczynski, M., Woltran, S.: Relativized hyperequivalence of logic programs for modular programming. TPLP **9**(6), 781–819 (2009)
12. Pearce, D., Valverde, A.: Uniform equivalence for equilibrium logic and logic programs. In: Lifschitz, V., Niemelä, I. (eds.) Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'04). vol. 2923 of LNCS, pp. 194–206. Springer, Berlin (2004)
13. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. J. Artif. Intell. Res. (JAIR) **35**, 813–857 (2009)
14. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006), pp. 412–416. IOS Press, Amsterdam (2006)
15. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: Lifschitz, V., Niemelä, I. (eds.) Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7). vol. 2923 of LNCS, pp. 87–99. Springer, Berlin (2004)
16. Eiter, T., Fink, M., Pührer, J., Tompits, H., Woltran, S.: Model-based recasting in answer-set programming. J. Appl. Non-Classical Logics **23**(1–2), 75–104 (2013)
17. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and uniform equivalence in answer-set programming: characterizations and complexity results for the non-ground case. In: Proceedings of the 20th national conference on artificial intelligence (AAAI'05), pp. 695–700. AAAI Press (2005)
18. Bliem, B., Woltran, S.: Equivalence between answer-set programs under (partially) fixed input. In: Gyssens, M., Simari, G.R. (eds.) Proceedings of the 9th International Symposium on Foundations of Information and Knowledge Systems (FoIKS'16). vol. 9616 of LNCS, pp. 95–111. Springer, Berlin (2016)
19. Lifschitz, V., Tang, L., Turner, H.: Nested expressions in logic programs. Ann. Math. Artif. Intell. **25**(3–4), 369–389 (1999)
20. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Log. **7**(3), 499–562 (2006)