Replacements in Non-Ground Answer-Set Programming*

Thomas Eiter, Michael Fink, Hans Tompits, Patrick Traxler, and Stefan Woltran

Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,michael,tompits,traxler,stefan}@kr.tuwien.ac.at

Abstract. In this paper, we propose a formal framework for specifying rule replacements in nonmonotonic logic programs within the answer-set programming paradigm. Of particular interest are replacement schemas retaining specific notions of equivalence, among them the prominent notions of strong and uniform equivalence, which have been introduced as theoretical tools for program optimization and verification. We derive some general properties of the replacement framework with respect to these notions of equivalence. Moreover, we generalize results about particular replacement schemas which have been established for ground programs to the non-ground case. Finally, we report a number of complexity results which address the problem of deciding how hard it is to apply a replacement to a given program. Our results provide an important step towards the development of effective optimization methods for non-ground answer-set programming, an issue which has not been addressed much so far.

1 Introduction

Answer-set programming (ASP) has emerged as an important paradigm for declarative problem solving, and provides a host for many different application domains on the basis of nonmonotonic logic programs [13]. The increasing interest in ASP has raised also the interest in semantic comparisons of programs in ASP, such as program equivalence [8, 4, 3] and correspondences [5, 11]. Comparisons of this kind are a basis for program optimization, where equivalence-preserving modifications are of primary interest; in particular, rewriting rules which allow to perform a local change in a program are important. Many such rules have been proposed in a propositional setting for different notions of equivalence (cf., e.g., [1, 10]).

Noticeably, except for the recent work by Lin and Chen [9], rewriting rules in the context of ASP have been considered more ad hoc rather than systematically, and were aimed at propositional programs. However, from a practical point of view, almost all programs use variables, and thus rewriting rules for this setting are essential.

In this paper, we address this issue and consider *replacements* for non-ground programs, according to which a subset of rules in a given program p may be exchanged with some other rules, possibly depending on a condition on p. For a simple example, consider an encoding of the three-coloring problem for graphs, which represents graphs using predicates $node$ and $edge$ and contains (among others) the two rules

$$r(X) \vee b(X) \leftarrow edge(X, a), node(a), node(X), \text{not } g(X), \tag{1}$$

$$r(Y) \vee b(Y) \vee g(Y) \leftarrow node(Y). \tag{2}$$

As our results show, Rule (1) is redundant in any program p which also contains Rule (2), i.e., we can replace (1) and (2) simply by (2). Similarly, we can replace (2) in p by its possible three "head-to-body" shifts, where all atoms in the head except one are moved to the body and negated, providing Rule (2) is head-cycle free in p.

Our contributions are briefly summarized as follows.

- We study replacements and replacement schemas in a general framework, paying attention to different natural types of replacements.
- We lift in this framework well-known replacement rules from the propositional case to the setting with variables. In particular, we focus on rules given by Brass and Dix [1] as well as by Eiter *et al.* [2], and generalize some of the results by Lin and Chen [9]. However, we also discuss some novel replacement rules.

* This work was partially supported by the Austrian Science Fund (FWF) under project P18019.

- We describe conditions under which replacements necessarily preserve strong equivalence [8]. We obtain interesting results which, to some extent, subsume recent results by Ferraris [6], who showed that strong equivalence is implicit with modular rewritings of ASP programs that preserve equivalence.
- Finally, we consider the computational complexity of applying specific replacement schemas, where we obtain bounds ranging from LOGSPACE up to PSPACE-completeness. These results provide a handle for deciding about efficient replacements in online and offline program optimization.

Our results extend and complement recent results about program equivalence to the relevant application setting. Furthermore, they provide a theoretical foundation for optimization techniques which in part are used ad hoc in ASP solvers.

2 Preliminaries

Logic programs are formulated in a language $\mathcal{L}$ containing a set $\mathcal{A}$ of *predicate symbols*, a set $\mathcal{V}$ of *variables*, and a set $\mathcal{C}$ of *constants* (also called the *domain* of $\mathcal{L}$). Each predicate symbol has an associated *arity* $n \geq 0$. An *atom* (over $\mathcal{L}$) is an expression of form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{A}$ is a predicate symbol 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 (*disjunctive*) *rule* (over $\mathcal{L}$), r, is of the form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m, \tag{3}$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, with $n \geq 0$, $m \geq k \geq 0$, and $n + m > 0$, and "not" denotes *default negation*. The *head* of r is the set $H(r) = \{a_1, \ldots, a_n\}$, and the *body* of r is $B(r) = \{b_1, \ldots, b_k, \text{ not } b_{k+1}, \ldots, \text{ not } b_m\}$. We also define $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$.

A rule r of form (3) is a *fact* if $m = 0$ and $n = 1$ (in which case "$\leftarrow$" is usually omitted). Moreover, r is *safe* if each variable occurring in $H(r) \cup B^-(r)$ also occurs in $B^+(r)$, and r is *ground* if all atoms occurring in it are ground.

By a *program* (over $\mathcal{L}$) we understand a finite set of rules (over $\mathcal{L}$). We assume in what follows that rules are always safe. The set of variables occurring in an atom a (resp., a rule r, a program p) is denoted by V_a (resp., V_r, V_p). Furthermore, the set of all constants occurring in p is called the *Herbrand universe* of p, symbolically $\mathcal{C}_p$. If no constant appears in p, then $\mathcal{C}_p = \{c\}$, for an arbitrary constant c. Moreover, $\mathcal{C}_r$ denotes all constants occurring in a rule r. The set of all predicates occurring in p is denoted by $\mathcal{A}_p$. As usual, the *Herbrand base*, B_p, of a program p is the set of all ground atoms constructed from $\mathcal{A}_p$ and $\mathcal{C}_p$.

Given a rule r and a set of constants $C \subseteq \mathcal{C}$, we define $grd(r, C)$ as the set of all rules $r\vartheta$ obtained from r by all possible substitutions $\vartheta : V_r \to C$. Moreover, for any program p, the *grounding of p with respect to C* is given by $grd(p, C) = \bigcup_{r \in p} grd(r, C)$. In particular, $grd(p, \mathcal{C}_p)$ is referred to as the *grounding of p* simpliciter, written $grd(p)$.

By an *interpretation* we understand a set of ground atoms. A ground rule r is *satisfied* by an interpretation I iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program p iff each $r \in p$ is satisfied by I. The *Gelfond-Lifschitz reduct* [7] of a ground program p with respect to an interpretation I is given by

$$p^I = \{H(r) \leftarrow B^+(r) \mid r \in p,\ I \cap B^-(r) = \emptyset\}.$$

A set $I \subseteq B_p$ is an *answer set* of p iff I is a subset-minimal set satisfying $grd(p)^I$. The set of all answer sets of p is denoted by $\mathcal{AS}(p)$.

In order to compare programs, we shall make use of different equivalence relations. In particular, for a class S of programs such that $\emptyset \in S$, we define, for every program p, p' over $\mathcal{L}$, $p \equiv^S p'$ iff, for each $p'' \in S$, $\mathcal{AS}(p \cup p'') = \mathcal{AS}(p' \cup p'')$ holds. By instantiating the parameter set S, we obtain the following well-known notions:

- *ordinary equivalence*, symbolically $\equiv_o$, by setting $S = \{\emptyset\}$;
- *uniform equivalence*, symbolically $\equiv_u$, by setting S as the class of all finite sets of ground facts in language $\mathcal{L}$;
- *strong equivalence*, symbolically $\equiv_s$, by setting S as the set of all programs over $\mathcal{L}$.

Note that $p \equiv_o p'$ iff $\mathcal{AS}(p) = \mathcal{AS}(p')$.

We say that a binary relation R *implies* a binary relation R' iff $R \subseteq R'$. Obviously, we have that $\equiv_s$ implies $\equiv_u$, and $\equiv_u$ implies $\equiv_o$.

For further details about strong and uniform equivalence between non-ground programs, we refer to [3].

3 Replacements

Definition 1. *A* replacement *is a triple $\varrho = (\phi, i, o)$, where ϕ is a unary predicate ranging over programs, called the* proviso of ϱ*, and i, o are sets of rules.*

We say that ϱ is applicable *to a program p, or p is ϱ-*eligible*, if $i \subseteq p$ and $\phi(p)$ holds. The* result of p under ϱ *is defined as*

$$\varrho[p] = (p \setminus i) \cup o, \text{ if } \varrho \text{ is applicable to } p, .$$

Definition 2. *Let $\equiv$ be an equivalence relation. A replacement ϱ is* $\equiv$-preserving *if $p \equiv \varrho[p]$, for any ϱ-eligible program p.*

Clearly, any $\equiv_s$-preserving replacement is also $\equiv_u$-preserving, and any $\equiv_u$-preserving replacement is also $\equiv_o$-preserving.

Definition 3. *Let $\varrho = (\phi, i, o)$ be a replacement. Then, ϱ is called*

- independent*, if for every program p, $\phi(p)$ holds,*
- monotone*, if for all programs p, p', $\phi(p)$ and $p \subseteq p'$ implies $\phi(p')$,*
- closed under intersection*, if for all programs p, p', $\phi(p)$ and $\phi(p')$ implies $\phi(p \cap p')$.*

We sometimes identify the proviso of an independent replacement by the designated predicate $\top(p)$, which is true for every program p. As well, an independent replacement (ϕ, i, o) may also be identified with the pair (i, o). Note that any independent replacement is also monotone and closed under intersection.

For illustration, consider a replacement $\varrho = (\phi, \{t\}, \emptyset)$, with t denoting a concrete rule, say, e.g., $q(x_1, x_2, x_3) \leftarrow q(x_1, x_2, x_3)$, and $\phi(p)$ holds for any program p. Then, ϱ is applicable to each program p with $t \in p$, and, in these cases, we get $\varrho[p] = p \setminus \{t\}$. Indeed, ϱ is an independent replacement. As we will see later on, ϱ is also $\equiv_s$-preserving.

In what follows, we show some general properties for replacements. In particular, the next property is central.

Theorem 1. *Let $\equiv$ be any equivalence relation implying $\equiv_o$. Then, any monotone replacement ϱ is $\equiv_s$-preserving, whenever ϱ is $\equiv$-preserving.*

Proof. Towards a contradiction, let $\varrho = (\phi, i, o)$ be a monotone $\equiv$-preserving replacement which is not $\equiv_s$-preserving. From the latter, we get that there exists some ϱ-eligible program p such that $p \not\equiv_s \varrho[p]$. Hence, there exists a program p' such that $\mathcal{AS}(p \cup p') \neq \mathcal{AS}(\varrho[p] \cup p')$. Without loss of generality, we can assume that $(p \cap p') = \emptyset$. Now, since ϱ is monotone and p is ϱ-eligible, $p \cup p'$ is ϱ-eligible as well. By hypothesis, ϱ is $\equiv$-preserving, and thus $p \cup p' \equiv \varrho[p \cup p']$ holds. This implies ordinary equivalence, i.e., $\mathcal{AS}(p \cup p') = \mathcal{AS}(\varrho[p \cup p'])$. Since $i \subseteq p$ and $(p \cap p') = \emptyset$, we obtain

$$\varrho[p \cup p'] = ((p \cup p') \setminus i) \cup o = (p \setminus i) \cup p' \cup o = ((p \setminus i) \cup o) \cup p' = \varrho[p] \cup p'.$$

Thus, $\mathcal{AS}(p \cup p') = \mathcal{AS}(\varrho[p] \cup p')$, a contradiction to $\mathcal{AS}(p \cup p') \neq \mathcal{AS}(\varrho[p] \cup p')$. □

Theorem 2. *An independent replacement (i, o) is $\equiv_s$-preserving iff $i \equiv_s o$.*

Proof. Let $\varrho = (i, o)$ be independent. The only-if direction is by definition when applying ϱ to i itself. For the if-direction, suppose that ϱ is not $\equiv_s$-preserving, i.e., there exists a program p with $i \subseteq p$ such that $p \not\equiv_s p'$, where $p' = (p \setminus i) \cup o$. Hence, for some program r, $\mathcal{AS}(p \cup r) \neq \mathcal{AS}(p' \cup r)$. In other words, for $p'' = (p \setminus i) \cup r$, we get $\mathcal{AS}(i \cup p'') \neq \mathcal{AS}(o \cup p'')$. Consequently, $i \not\equiv_s o$. □

We note that Ferraris [6] shows that in a propositional setting, for any function f which maps single rules r to sets R of rules such that $\mathcal{A}_R \subseteq \mathcal{A}_{\{r\}}$ the following holds: for all programs p, $p \equiv_o \bigcup_{r \in p} f(r)$ iff, for each r, $r \equiv_s f(r)$. This can be concluded from Theorems 1 and 2 as follows: Each function f_r that maps r to $f(r)$ and any other rule r' to itself can be viewed as an independent replacement. Thus, by Theorems 1 and 2, $r \equiv_s f(r)$ must hold if $p \equiv_o \bigcup_{r \in p} f(r)$ holds for all p (take $p = \{r\}$). The converse is obvious.

4 Replacement Schemas

So far, we only considered concrete replacements guided by fixed sets of rules i, o. However, in general, one wants to collect sets of replacements into a single *replacement schema*. This can be realized as follows:

Definition 4. *A* replacement schema, $\mathcal{R}$, *is a partial function mapping pairs* (i, o) *of programs into a unary predicate* $\mathcal{R}(i,\ o)$. *The domain of* $\mathcal{R}$ *is denoted by* $\mathsf{dom}(\mathcal{R})$.

A replacement (ϕ, i, o) *is an* instance *of* $\mathcal{R}$ *if* $(i, o) \in \mathsf{dom}(\mathcal{R})$ *and* $\phi = \mathcal{R}(i, o)$. *The set of all instances of* $\mathcal{R}$ *is denoted by* $inst(\mathcal{R})$.

We say that $\mathcal{R}$ *is* applicable *to a program* p, *or* p *is* $\mathcal{R}$-eligible, *if there exists some* $\varrho \in inst(\mathcal{R})$ *which is applicable to* p. *We refer to the result* $\varrho[p]$ *of* p *under an instance* $\varrho \in inst(\mathcal{R})$ *as a result of* p *under* $\mathcal{R}$. *By* $\mathcal{R}^\star[p]$ *we denote the set of all results of* p *under* $\mathcal{R}$, *i.e.,*

$$\mathcal{R}^\star[p] = \{\varrho[p] \mid \varrho \in inst(\mathcal{R})\}.$$

With an abuse of notation, we also write $\mathcal{R}[p]$ to refer to a result of p under $\mathcal{R}$.

The operator $\mathcal{R}^\star[\cdot]$ is used to compare replacement schemas as follows.

Definition 5. *Two replacement schemas,* $\mathcal{R}_1$ *and* $\mathcal{R}_2$, *are* equipollent *iff, for each program* p, $\mathcal{R}_1^\star[p] = \mathcal{R}_2^\star[p]$.

Properties for replacements are easily generalized to schemas as follows:

Definition 6. *A replacement schema* $\mathcal{R}$ *is said to be* $\equiv$-preserving (*resp.,* independent, monotone, intersection-closed) *if each instance of* $\mathcal{R}$ *is* $\equiv$*-preserving* (*resp., independent, monotone, intersection-closed*).

Note that for an independent replacement schema $\mathcal{R}$, we may identify $\mathsf{dom}(\mathcal{R})$ with $inst(\mathcal{R})$. Furthermore, the results about replacements, as given by Theorems 1 and 2, carry over in a straightforward way to replacement schemas as well.

We are now prepared to give particular replacement schemas. We start with a generalization of a concept considered by Brass and Dix [1] for the propositional case.

Definition 7. *The replacement schema* TAUT *is given as follows:*

- $\mathsf{dom}(\mathrm{TAUT}) = \{(\{s\}, \emptyset) \mid s$ *is a rule with* $H(s) \cap B^+(s) \neq \emptyset\}$;
- $\mathrm{TAUT}(i, o) = \top$, *for every* $(i, o) \in \mathsf{dom}(\mathrm{TAUT})$.

The instances of TAUT are then all replacements of the form $(\top, \{s\}, \emptyset)$, where $H(s) \cap B^+(s) \neq \emptyset$. For instance, let $p = \{s(X) \leftarrow s(X), q(Y); q(X) \leftarrow q(X), s(X); s(a)\}$. Then, $\mathrm{TAUT}[p]$ refers either to $p' = \{s(X) \leftarrow s(X), q(Y); s(a)\}$ or to $p'' = \{q(X) \leftarrow q(X), s(X); s(a)\}$. Hence, $\mathrm{TAUT}^\star[p] = \{p', p''\}$.

As an example of a non-monotone replacement schema, we define *local shifting*, LSH, extending a similar schema introduced in the propositional case by Eiter *et al.* [2]. The idea underlying local shifting has already been sketched in the introduction. Formally, we need the following concepts.

The (*positive*) *dependency graph*, G_p, of a ground program p is given by the pair (B_p, E_p), where $(a, b) \in E_p$ iff there exists a rule $r \in p$ such that $a \in H(r)$ and $b \in B^+(r)$. An atom a *positively depends* on b in p iff there exists a path from a to b in G_p. A ground rule r is *head-cycle free* (HCF) in p iff no distinct atoms $a, b \in H(r)$ mutually positively depend on each other in p.

For an arbitrary program p (not necessarily ground), $r \in p$ is HCF in p iff, for each finite $C \subseteq \mathcal{C}$ and each $r' \in grd(r, C)$, r' is HCF in $grd(p, C)$.

Definition 8. *The replacement schema* LSH *is given as follows:*

- $\mathsf{dom}(\mathrm{LSH})$ *consists of all pairs* $(\{r\}, o_r)$, *where*
  1. r *is a rule such that, for each* $\vartheta : V_r \to \mathcal{C}$, $|H(r\vartheta)| > 1$, *and*
  2. $o_r = \{h \leftarrow B(r), \mathrm{not}\ (H(r) \setminus h) \mid h \in H(r)\}$;[1]
- *for every* $(i, o) \in \mathsf{dom}(\mathrm{LSH})$, $\mathrm{LSH}(i, o) = \phi$, *where* $\phi(p)$ *holds iff* r *is HCF in* p *and* $i = \{r\}$.

Note that LSH is, for instance, not applicable to the program $q(X_1) \vee q(X_2) \leftarrow r(X_1, X_2)$, since ϑ mapping X_1 and X_2 to the same constant c yields $H(r\vartheta) = \{q(c)\}$ with cardinality $= 1$.

We mention that LSH is intersection-closed, but neither monotone nor independent. Equivalence-preserving properties for TAUT and LSH will be provided in the next section.

[1] For a set $S = \{a_1, \ldots, a_n\}$ of atoms, not S denotes the expression $\mathrm{not}\ a_1, \ldots, \mathrm{not}\ a_n$.

5 Equivalence Preserving Replacement Schemas

This section collects a number of concrete replacement schemas. In particular, we generalize ideas from propositional ASP, where such replacements have been stipulated by Brass and Dix [1] and further investigated and developed by several authors [10, 9, 12, 2].

The section is organized as follows. First, we consider $\equiv_s$-preserving replacement schemas. Then, we deal with monotone replacement schemas—in particular, we relate our framework to the one discussed by Lin and Chen [9]. Finally, we consider replacement schemas which are not $\equiv_s$-preserving but $\equiv_u$- or $\equiv_o$-preserving.

5.1 Independent Replacement Schemas

We already gave an independent replacement schema above, namely TAUT. A very similar schema is CONTRA, defined below. Like TAUT, CONTRA has been introduced in the propositional setting by Brass and Dix [1], and, with respect to equivalence notions, studied further by Eiter *et al.* [2] and Osorio *et al.* [10].

Definition 9. *The replacement schema* CONTRA *is given as follows:*

- $\mathsf{dom}(\text{CONTRA}) = \{(\{s\}, \emptyset) \mid s \text{ is a rule with } B^+(s) \cap B^-(s) \neq \emptyset\}$;
- $\text{CONTRA}(i, o) = \top$, *for every* $(i, o) \in \mathsf{dom}(\text{CONTRA})$.

However, an alternative way to capture the nature of TAUT and CONTRA is the following:

Definition 10. *Schemas* ϑ-TAUT *and* ϑ-CONTRA *are given as follows:*

- $\mathsf{dom}(\vartheta\text{-TAUT}) = \{(\{s\}, \emptyset) \mid s \text{ is a rule such that, for each } \vartheta : V_s \to \mathcal{C},\ H(s\vartheta) \cap B^+(s\vartheta) \neq \emptyset\}$;
- $\mathsf{dom}(\vartheta\text{-CONTRA}) = \{(\{s\}, \emptyset) \mid s \text{ is a rule such that, for each } \vartheta : V_s \to \mathcal{C},\ B^+(s\vartheta) \cap B^-(s\vartheta) \neq \emptyset\}$;
- $\mathcal{R}(i, o) = \top$, *for every* $(i, o) \in \mathsf{dom}(\mathcal{R})$, *with* $\mathcal{R} \in \{\vartheta\text{-TAUT}, \vartheta\text{-CONTRA}\}$.

Theorem 3. *The following properties hold:*

1. TAUT *and* CONTRA *are* $\equiv_s$*-preserving;*
2. TAUT *and* ϑ-TAUT *are equipollent; and*
3. CONTRA *and* ϑ-CONTRA *are equipollent.*

We finally give four more replacement schemas which generalize corresponding replacement rules given in the literature for ground programs. In particular, the ground pendants of schemas RED^- and NONMIN have been introduced by Brass and Dix [1], the ground version of S-IMPL is due to Wang and Zhou [12], and that of SUB is discussed by Lin and Chen [9].

Definition 11. *The schemas* $\mathcal{R} \in \{\text{RED}^-, \text{NONMIN}, \text{S-IMPL}, \text{SUB}\}$ *are given as follows:*

- $\mathsf{dom}(\mathcal{R})$ *consists of all pairs* $(\{r, s\}, \{s\})$, *where* r, s *are rules, such that*
  - *for* $\mathcal{R} = \text{RED}^-$, $H(s) \subseteq B^-(r)$ *and* $B(s) = \emptyset$, *and*
  - *for* $\mathcal{R} \in \{\text{NONMIN}, \text{S-IMPL}, \text{SUB}\}$, *there exists* $\vartheta : V_s \to V_r \cup \mathcal{C}_r$ *such that* $B^+(s\vartheta) \subseteq B^+(r)$ *and*
    - *for* $\mathcal{R} = \text{NONMIN}$, $H(s\vartheta) \subseteq H(r)$ *and* $B^-(s\vartheta) \subseteq B^-(r)$,
    - *for* $\mathcal{R} = \text{S-IMPL}$, *there is some* $A \subseteq B^-(r)$ *with* $H(s\vartheta) \subseteq H(r) \cup A$ *and* $B^-(s\vartheta) \subseteq B^-(r) \setminus A$, *and*
    - *for* $\mathcal{R} = \text{SUB}$, $H(s\vartheta) \subseteq H(r) \cup B^-(r)$ *and* $B^-(s\vartheta) \subseteq B^-(r)$; *and*
- $\mathcal{R}(i, o) = \top$, *for every* $(i, o) \in \mathsf{dom}(\mathcal{R})$.

Observe that the safety condition of rules implies that RED^- is only applicable in case s is a ground disjunctive fact. This is the reason why, in contrast to the other three schemas, there is no need for ϑ in the definition for RED^-.

The four schemas introduced above stand in the following relationships to each other:

– if RED$^-$ or NONMIN is applicable to a program p, then S-IMPL is applicable to p, and
– if S-IMPL is applicable to a program p, then SUB is applicable to p.

Hence, the schema SUB the most unconstrained among the four, being applicable whenever any of the other three is.

Theorem 4. *The replacement schemas* RED$^-$, NONMIN, S-IMPL, *and* SUB *are all* $\equiv_s$*-preserving.*

Like for TAUT and CONTRA, also the above replacement schemas can be defined in an alternative way, explicitly referring to all groundings of the rules involved. We leave a further discussion of this point to a full version of this paper.

5.2 Monotone Replacement Schemas

For monotone replacement schemas, there is an interesting relation to independent replacement schemas as follows:

Theorem 5. *Any replacement schema which is monotone, closed under intersection, and* $\equiv_s$*-preserving is equipollent to an independent replacement schema.*

Proof. Let $\mathcal{R}$ be a replacement schema which is monotone, closed under intersection, and $\equiv_s$-preserving. Consider some $\varrho \in \mathit{inst}(\mathcal{R})$ with $\varrho = (\phi, i, o)$. Since ϱ is monotone and closed under intersection, there exists a unique program, p_0, such that $\phi(p)$ holds for each $p \supseteq p_0$ but $\phi(p')$ does not hold for any $p' \subset p_0$. Obviously, $\varrho' = (\top, i \cup p_0, o \cup (p_0 \setminus i))$ then satisfies $\varrho[s] = \varrho'[s]$ for every program s. It follows that the replacement schema $\mathcal{R}'$, defined by setting $\mathsf{dom}(\mathcal{R}') = \{(i \cup p_0, o \cup (p_0 \setminus i)) \mid (i, o) \in \mathsf{dom}(\mathcal{R})\}$ and $\mathcal{R}'(i, o) = \top$, for every $(i, o) \in \mathsf{dom}(\mathcal{R}')$, is equipollent to $\mathcal{R}$. Moreover, $\mathcal{R}'$ is clearly independent. □

In recent work, Lin and Chen [9] captured certain classes of strongly equivalent propositional programs by considering problems of the following form:

> Given rules $r_1, \ldots, r_k, u_1, \ldots, u_m, v_1, \ldots, v_n$, is $\{r_1, \ldots, r_k, u_1, \ldots, u_m\}$ strongly equivalent to $\{r_1, \ldots, r_k, v_1, \ldots, v_n\}$?

Such a problem is referred to as a *k-m-n-problem*. The main focus of Lin and Chen's work is to find computationally effective, *necessary* and *sufficient* conditions, for small k, m, n, making a k-m-n-problem true. In general, any condition that guarantees a positive answer to a k-m-n-problem, for fixed k, m, and n, obviously yields a monotone replacement schema. Moreover, the conditions given by Lin and Chen [9] for particular problem classes additionally enforce that the corresponding schema is closed under intersection. In fact, Theorem 5 constitutes a generalization of observations made Lin and Chen [9].

We next deal with properties for 0-1-0-problems. To this end, we introduce the following replacement schemas.

Definition 12. *Schemas* LC$_{0\text{-}1\text{-}0}$ *and* ϑ-LC$_{0\text{-}1\text{-}0}$ *are given as follows:*

– $\mathsf{dom}(\text{LC}_{0\text{-}1\text{-}0}) = \{(\{s\}, \emptyset) \mid s \text{ is a rule with } B^+(s) \cap (H(s) \cup B^-(s)) \neq \emptyset\}$;
– $\mathsf{dom}(\vartheta\text{-LC}_{0\text{-}1\text{-}0}) = \{(\{s\}, \emptyset) \mid s \text{ is a rule such that, for each } \vartheta : V_s \to \mathcal{C},\ B^+(s\vartheta) \cap (H(s\vartheta) \cup B^-(s\vartheta)) \neq \emptyset\}$;
– $\mathcal{R}(i, o) = \top$, *for every* $(i, o) \in \mathsf{dom}(\mathcal{R})$, *with* $\mathcal{R} \in \{\text{LC}_{0\text{-}1\text{-}0}, \vartheta\text{-LC}_{0\text{-}1\text{-}0}\}$.

Obviously, the syntactic criterion of LC$_{0\text{-}1\text{-}0}$ combines, in a sense, the conditions for TAUT and CONTRA. This is made precise as follows:

Theorem 6. $\text{LC}_{0\text{-}1\text{-}0}^\star[p] = \text{TAUT}^\star[p] \cup \text{CONTRA}^\star[p]$, *for any program* p.

In view of this and previous results, the next theorem comes at no surprise:

Theorem 7. LC$_{0\text{-}1\text{-}0}$ *is* $\equiv_s$*-preserving. Furthermore,* LC$_{0\text{-}1\text{-}0}$ *is equipollent to* ϑ-LC$_{0\text{-}1\text{-}0}$.

As mentioned above, Lin and Chen [9] are concerned with conditions making k-m-n problems true, for small k, m, n. The following proposition rephrases a result of that endeavour:

Proposition 1 ([9]). *For any ground rule r, $\{r\} \equiv_s \emptyset$ iff $\mathrm{LC}_{0\text{-}1\text{-}0}$ is applicable to $\{r\}$.*

This result can be lifted to the non-ground case, yielding a syntactic criterion when a single rule is redundant in a program.

Theorem 8. *For any rule r, $\{r\} \equiv_s \emptyset$ iff $\mathrm{LC}_{0\text{-}1\text{-}0}$ is applicable to $\{r\}$.*

Finally, we remark that the replacement schema SUB, introduced in the previous section, is the generalization of another condition given by Lin and Chen [9] for propositional programs.

5.3 Non-Monotone Replacement Schemas

Theorem 9. LSH *is $\equiv_u$-preserving, but not $\equiv_s$-preserving.*

Indeed, the fact that LSH is not $\equiv_s$-preserving already follows from an analogous result in the propositional case [2]. However, to illustrate this property, consider the following example in the non-ground setting: Take p as consisting of the single rule $r = q(X) \vee r(X) \leftarrow s(X,Y)$. Clearly, r is HCF in p, and thus LSH is applicable to p. However, for $p' = \mathrm{LSH}[p]$, we have $p \not\equiv_s p'$, which can be seen by considering, e.g., $p'' = \{q(Y) \leftarrow r(Y); r(X) \leftarrow q(X); s(a,b)\}$, for which we get that $\mathcal{AS}(p \cup p'') = \{s(a,b), q(a), r(a)\}$ while $\mathcal{AS}(p' \cup p'') = \emptyset$.

We next introduce a $\equiv_u$-preserving replacement schema, which, to the best of our knowledge, has not been considered before, even in a propositional setting. Note that in the definition below, δ is required to be a (bijective) renaming rather than a substitution.

Definition 13. *The replacement schema* FOLD *is given as follows:*

- $\mathsf{dom}(\mathrm{FOLD})$ *is the set of all pairs $(\{r,s\},\{t\})$, where r,s,t are rules and there exists a renaming δ and an atom $a \in B^-(r\delta) \cap B^+(s)$ such that $H(r\delta) = H(s) = H(t)$ and $(B(r\delta) \setminus \{\mathrm{not}\ a\}) = (B(s) \setminus \{a\}) = B(t)$;*
- *for every $(i,o) \in \mathsf{dom}(\mathrm{FOLD})$, $\mathrm{FOLD}(i,o) = \phi$, where $\phi(p)$ holds iff, for each head atom b in p and each $\vartheta_a : V_a \to \mathcal{C}$ and $\vartheta_b : V_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$, with a as above.*

Theorem 10. FOLD *is $\equiv_u$-preserving, but not $\equiv_s$-preserving.*

For illustration, consider $p = \{q(X,X) \leftarrow r(X), \mathrm{not}\ s(X); q(Y,Y) \leftarrow r(Y), s(Y)\}$. We can apply FOLD to p since no atom $s(\cdot)$ occurs in a head of p. The result of the replacement is $p' = \mathrm{FOLD}[p] = \{q(Y,Y) \leftarrow r(Y)\}$. By the theorem above, $p \equiv_u p'$. For instance, adding $t = \{r(a)\}$ yields $\mathcal{AS}(p \cup t) = \mathcal{AS}(p' \cup t) = \{r(a), q(a,a)\}$. On the other hand, adding $t' = \{r(a), s(X) \leftarrow q(X,Y)\}$ results in $\mathcal{AS}(p \cup t') = \emptyset$, while $\mathcal{AS}(p' \cup t') = \{r(a), s(a), q(a,a)\}$. This shows that FOLD is not $\equiv_s$-preserving; a corresponding counterexample can also be constructed for the propositional setting as well. Furthermore, FOLD is applicable to the program $p \cup t$ as well, but it is not applicable to $p \cup t'$. Since $t' \supset t$, we observe that FOLD is not monotone.

Finally, we briefly discuss a replacement schema which is $\equiv_o$-preserving but not $\equiv_u$-preserving. For the propositional case, this replacement schema was first considered by Brass and Dix [1].

Definition 14. *The replacement schema* RED^+ *is given as follows:*

- $\mathsf{dom}(\mathrm{RED}^+)$ *is the set of all pairs $(\{r\},\{t\})$, where r, t are rules such that $H(r) = H(t)$ and $B(r) = B(t) \cup \{\mathrm{not}\ a\}$;*
- *for every $(i,o) \in \mathsf{dom}(\mathrm{RED}^+)$, $\mathrm{RED}^+(i,o) = \phi$, where $\phi(p)$ holds iff, for each head atom b in p and each $\vartheta_a : V_a \to \mathcal{C}$ and $\vartheta_b : V_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$, where a is an atom such that $B(i) = B(o) \cup \{\mathrm{not}\ a\}$.*

Note that RED^+ is, to some extent, a simplification of FOLD, where the second rule, s, of i having a positive in its body is not mandatory anymore. As a consequence, the equivalence notion preserved by RED^+ is weaker.

Theorem 11. RED^+ *is $\equiv_o$-preserving, but not $\equiv_u$-preserving.*

As in the case of LSH, the fact that RED^+ is not $\equiv_u$-preserving follows immediately from a corresponding result in the propositional case [2].

6 Complexity of Applicability

In this section, we deal with the computational complexity of the *applicability problem* for a given replacement schema $\mathcal{R}$, which is the task of determining whether $\mathcal{R}$ is applicable to a given program.

Our first result concerns the schemas TAUT, CONTRA, and RED^-.

Theorem 12. *The applicability problem for* $\mathcal{R} \in \{\text{TAUT}, \text{CONTRA}, \text{RED}^-\}$ *is in* LOGSPACE.

The independent replacement schemas involving two rules, which we considered, are more complex, however.

Theorem 13. *The applicability problem for* $\mathcal{R} \in \{\text{NONMIN}, \text{S-IMPL}, \text{SUB}\}$ *is* NP*-complete.* NP*-hardness holds even if the arities of the predicates in the given program are bounded by a constant.*

We now turn to non-monotone replacements.

Theorem 14. *The applicability problem for* LSH *is* PSPACE*-complete. If each predicate in the given program has its arity bounded by a constant, the problem is* NLOGSPACE*-complete.*

Informally, the difficult part is solving the HCF test, which amounts to test reachability in an implicitly represented graph, which is PSPACE-complete. Note that, in the practical relevant setting of programs having bounded predicate arities, LSH-applicability can be tested in NLOGSPACE, and thus in polynomial time. Here, the implicit graph can be effectively constructed using logarithmic workspace.

While LSH is computationally involving in the general case, the other two non-monotone replacement schemas turn out to be easier.

Theorem 15. *The applicability problem for* FOLD *is polynomially equivalent* (*under Turing-reductions*) *to the graph isomorphism problem.*

Here, computational hardness is located in the check whether two rules in the given program yield an instance of FOLD, rather than the test involving the proviso. Indeed, the problem of finding a bijective renaming δ allows for a representation of graph isomorphism already if we restrict ourselves to programs over binary atoms. In turn, we can show that FOLD-applicability can be decided by a polynomial number of tests for graph isomorphism. Graph isomorphism is in NP but it is not known to be NP-complete or belonging to P.

Our final result provides a tractable case.

Theorem 16. *The applicability problem for* RED^+ *is* LOGSPACE*-complete.*

7 Conclusion

Our results on replacements provide a basis for program optimization by rewriting in the practicably important setting of non-ground programs. While many rewriting rules have been proposed for propositional programs, generalizations to the non-ground case have not yet been considered in an answer-set programming setting. We have addressed this issue considering safe programs. However, safety is not necessarily required and, in many cases, unsafe rules can be taken into account if their replacement does not change the active domain of the program. We leave further details on these issues for future work.

Applying replacements for program optimization requires the program to be scanned for applicable replacements. Depending on the considered replacement schema, this test requires different computational effort, ranging from tractable cases up to PSPACE.

An implementation of the applicability tests for some of the most general replacement schemas which we have presented is currently under development, using the ASP solver DLV and Perl. Note that for all schemas considered in this paper, these tests are cheaper than the complexity of computing an answer set (which is NEXP^{NP}-hard in general for disjunctive programs)—in fact, with the exception of LSH, these tests are *drastically* cheaper. Thus, the schemas might be considered also for *online optimization* and not only for static *offline optimization*.

References

1. S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.
2. T. Eiter, M. Fink, H. Tompits, and S. Woltran. Simplifying Logic Programs Under Uniform and Strong Equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'04*), volume 2923 of *LNCS*, pages 87–99. Springer Verlag, 2004.
3. T. Eiter, M. Fink, H. Tompits, and S. Woltran. 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*), pages 695–700. AAAI Press, 2005.
4. T. Eiter, M. Fink, and S. Woltran. Semantical Characterizations and Complexity of Equivalences in Stable Logic Programming. Technical Report INFSYS RR-1843-05-01, Institut für Informationssysteme, Technische Universität Wien, Austria, 2005. Accepted for publication in *ACM Transactions on Computational Logic*.
5. T. Eiter, H. Tompits, and S. Woltran. On Solution Correspondences in Answer Set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI'05*), pages 97–102, 2005.
6. P. Ferraris. On Modular Translations and Strong Equivalence. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR'05*), volume 3552 of *LNCS*, pages 79–91. Springer Verlag, 2005.
7. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
8. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
9. F. Lin and Y. Chen. Discovering Classes of Strongly Equivalent Logic Programs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI'05*), pages 516–521, 2005.
10. M. Osorio, J. A. Navarro, and J. Arrazola. Equivalence in Answer Set Programming. In *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation* (*LOPSTR'01*), *Selected Papers*, volume 2372 of *LNCS*, pages 57–75. Springer Verlag, 2001.
11. D. Pearce and A. Valverde. Synonymous Theories in Answer Set Programming and Equilibrium Logic. In *Proceedings of the 16th European Conference on Artificial Intelligence* (*ECAI'04*), pages 388–392. IOS Press, 2004.
12. K. Wang and L. Zhou. Comparisons and Computation of Well-founded Semantics for Disjunctive Logic Programs. *ACM Transactions on Computational Logic*, 6(2):295–327, 2005.
13. S. Woltran (ed.). Answer Set Programming: Model Applications and Proofs-of-Concept. Technical Report WP5, Working Group on Answer Set Programming (WASP, IST-FET-2001-37004). Available at `http://www.kr.tuwien.ac.at/projects/WASP/report.html`.