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.

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 (Baral 2003; Gelfond & Leone 2002; Woltran 2005). The increasing interest in ASP lead also to the investigation of semantic comparisons of programs in ASP, giving rise to the introduction of different notions of program equivalence (Lifschitz, Pearce, & Valverde 2001; Eiter, Fink, & Woltran 2005; Eiter *et al.* 2005) and program correspondence (Pearce & Valverde 2004b; Eiter, Tompits, & Woltran 2005). Such comparison relations are essential for program-optimization tasks, where equivalence-preserving modifications are of primary interest—in particular, *rewriting rules*, which allow to perform local changes in programs, are fundamental. Many such rules have been proposed in a propositional setting for different notions of equivalence (cf., e.g., Brass & Dix (1999) and Osorio, Navarro, & Arrazola (2001)).

Noticeably, except for the recent work put forth by Lin & Chen (2005), rewriting rules in the context of ASP have been considered more ad hoc rather than systematically, and were aimed at propositional programs only. 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

$$\begin{array}{rcll} r(X) \vee b(X) & \leftarrow & edge(a,X), node(a), & (1)\\ & & node(X), \text{not } g(X), & \\ r(Y) \vee b(Y) \vee g(Y) & \leftarrow & node(Y). & (2)\end{array}$$

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 can be briefly summarized as follows.

- We study replacements and replacement schemas in a general framework, paying attention to different natural types of replacements and analyzing relations between them. In particular, we describe conditions under which replacements necessarily preserve strong equivalence (Lifschitz, Pearce, & Valverde 2001).
- We lift in this framework well-known replacement rules from the propositional case to the setting with variables. We focus hereby on rules discussed by Brass & Dix (1999) as well as by Eiter *et al.* (2004), and generalize some of the results by Lin & Chen (2005). However, we also discuss some novel replacement rules.
- 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

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

solvers. We focus on *safe programs* in our development, which is the pre-eminent setting for the currently available ASP solvers (like, e.g., DLV (Leone *et al.* 2002) or Smodels (Simons, Niemelä, & Soininen 2002)), but this is no real restriction in general.

For the sake of presentation, the proofs of most results are relegated to an appendix, which also contains ancillary notation and characterizations not required in the main body of the paper.

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 is a predicate symbol from $\mathcal{A}$ 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 an ordered pair of form

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

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms ($n \geq 0, m \geq k \geq 0$, $n + m > 0$), and "not" denotes *default negation*. The *head* of r is given by $H(r) = \{a_1, \ldots, a_n\}$, 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 use $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$. We call r 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 $\mathcal{V}_a$ (resp., $\mathcal{V}_r$, $\mathcal{V}_P$). 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 applying all possible substitutions $\vartheta : \mathcal{V}_r \rightarrow C$ to r. 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, symbolically $I \models r$, iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. I satisfies a ground program P, symbolically $I \models P$, iff $I \models r$, for each $r \in P$. The *Gelfond-Lifschitz reduct* (Gelfond & Lifschitz 1991) 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 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 $\mathcal{L}$; and
- *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 ρ *implies* a binary relation ρ' iff $\rho \subseteq \rho'$. 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 Eiter *et al.* (2005).

Replacements

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

We say that ϱ is applicable *to a program P, or P is ϱ-*eligible*, if $M \subseteq P$ and $\phi(P)$ holds. The* result of *P under ϱ is $\varrho[P] = (P \setminus M) \cup N$, in case ϱ 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.*

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

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

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

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, M, N) may also be identified with the pair (M, N). Note that any independent replacement is also monotone and closed under intersection.

For illustration, consider a replacement $\varrho = (\phi, \{r\}, \emptyset)$, with r 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 $r \in P$, and, in these cases, we get $\varrho[P] = P \setminus \{r\}$. 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 $\varrho = (\phi, M, N)$ 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 $M \subseteq P$ and $(P \cap P') = \emptyset$, we obtain

$$\begin{aligned}\varrho[P \cup P'] &= ((P \cup P') \setminus M) \cup N \\ &= (P \setminus M) \cup P' \cup N \\ &= ((P \setminus M) \cup N) \cup P' \\ &= \varrho[P] \cup P'.\end{aligned}$$

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 (M, N) is $\equiv_s$-preserving iff $M \equiv_s N$.*

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

Replacement Schemas

So far, we only considered concrete replacements guided by fixed sets M, N of rules. 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 (M, N) of programs into a unary predicate $\mathcal{R}(M, N)$. The domain of $\mathcal{R}$ is denoted by* $\mathsf{dom}(\mathcal{R})$.

A replacement (ϕ, M, N) is an instance *of $\mathcal{R}$ if $(M, N) \in$* $\mathsf{dom}(\mathcal{R})$ *and $\phi = \mathcal{R}(M, N)$. 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 $\mathcal{R}$. Furthermore, $\mathcal{R}^\star[P]$ denotes the set $\{\varrho[P] \mid \varrho \in inst(\mathcal{R})\}$.*

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 replacement 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 $inst(\mathcal{R})$ with $\mathsf{dom}(\mathcal{R})$. 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 examples of particular replacement schemas. We start with a generalization of a concept considered by Brass & Dix (1999) for the propositional case.

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

- $\mathsf{dom}(\text{TAUT}) = \{(\{r\}, \emptyset) \mid r$ *is a rule with* $H(r) \cap B^+(r) \neq \emptyset\}$*;*
- $\text{TAUT}(M, N) = \top$*, for every* $(M, N) \in \mathsf{dom}(\text{TAUT})$.

The instances of TAUT are then all replacements of the form $(\top, \{r\}, \emptyset)$, where $H(r) \cap B^+(r) \neq \emptyset$.

For example, let $P = \{p(X) \leftarrow p(X), q(Y);\ q(X) \leftarrow p(Y), q(X);\ p(a)\}$. Then, $\text{TAUT}[P]$ refers either to

$$P' = \{p(X) \leftarrow p(X), q(Y);\ p(a)\}$$

or to

$$P'' = \{q(X) \leftarrow p(Y), p(X);\ p(a)\}.$$

Hence, $\text{TAUT}^\star[P] = \{P', P''\}$. In any case, applying TAUT twice, we get $\text{TAUT}[\text{TAUT}[P]] = \{p(a)\}$.

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.* (2004).

The idea underlying local shifting has already been mentioned 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), a rule $r \in P$ 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:*

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

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

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

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.

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 & Dix (1999) and further investigated and developed by several authors (Osorio, Navarro, & Arrazola 2001; Lin & Chen 2005; Wang & Zhou 2005; Eiter *et al.* 2004).

We proceed 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 & Chen (2005). Finally, we consider replacement schemas which are not $\equiv_s$-preserving but $\equiv_u$- or $\equiv_o$-preserving.

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 & Dix (1999), and, with respect to equivalence notions, studied further by Eiter *et al.* (2004) and Osorio, Navarro, & Arrazola (2001).

Definition 9 *The replacement schema* CONTRA *is given by setting* $\mathsf{dom}(\text{CONTRA}) = \{(\{r\}, \emptyset) \mid r$ *is a rule with* $B^+(r) \cap B^-(r) \neq \emptyset\}$ *and* $\text{CONTRA}(M, N) = \top$*, for every* $(M, N) \in \mathsf{dom}(\text{CONTRA})$.

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

Definition 10 *The schemas* ϑ-TAUT *and* ϑ-CONTRA *are given as follows:*

- $\mathsf{dom}(\vartheta\text{-TAUT}) = \{(\{r\}, \emptyset) \mid r$ *is a rule such that, for each* $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $H(r\vartheta) \cap B^+(r\vartheta) \neq \emptyset\}$*;*
- $\mathsf{dom}(\vartheta\text{-CONTRA}) = \{(\{r\}, \emptyset) \mid r$ *is a rule such that, for each* $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $B^+(r\vartheta) \cap B^-(r\vartheta) \neq \emptyset\}$*; and*
- $\mathcal{R}(M, N) = \top$*, for every* $(M, N) \in \mathsf{dom}(\mathcal{R})$*, 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 & Dix (1999), the ground version of S-IMPL is due to Wang & Zhou (2005), and that of SUB is discussed by Lin & Chen (2005).

Definition 11 *The schemas* $\mathcal{R} \in \{\text{RED}^-, \text{S-IMPL}, \text{SUB}, \text{NONMIN}\}$ *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* $\mathcal{R} \in \{\text{S-IMPL}, \text{SUB}, \text{NONMIN}\}$*, there exists a* $\vartheta : \mathcal{V}_s \to \mathcal{V}_r \cup \mathcal{C}_r$ *such that* $B^+(s\vartheta) \subseteq B^+(r)$ *and*
    - *for* $\mathcal{R} = \text{S-IMPL}$*, there is some* $A \subseteq B^-(r)$ *with* $H(s\vartheta) \subseteq H(r) \cup A$ *and* $B^-(s\vartheta) \subseteq B^-(r) \setminus A$*,*
    - *for* $\mathcal{R} = \text{SUB}$, $H(s\vartheta) \subseteq H(r) \cup B^-(r)$ *and* $B^-(s\vartheta) \subseteq B^-(r)$*, and*
    - *for* $\mathcal{R} = \text{NONMIN}$, $H(s\vartheta) \subseteq H(r)$ *and* $B^-(s\vartheta) \subseteq B^-(r)$*; and*
- $\mathcal{R}(M, N) = \top$*, for every* $(M, N) \in \mathsf{dom}(\mathcal{R})$.

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 is 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, by explicitly referring to all groundings of the rules involved. We leave a further discussion of this point to a full version of this paper.

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 $\varrho = (\phi, M, N)$. 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, $\varrho' = (\top, M \cup P_0, N \cup (P_0 \setminus M))$ then satisfies $\varrho[P] = \varrho'[P]$ for every program P. It follows that the replacement schema $\mathcal{R}'$, defined by setting $\mathsf{dom}(\mathcal{R}') = \{(M \cup P_0, N \cup (P_0 \setminus M)) \mid (M, N) \in \mathsf{dom}(\mathcal{R})\}$ and $\mathcal{R}'(M, N) = \top$, for every $(M, N) \in \mathsf{dom}(\mathcal{R}')$, is equipollent to $\mathcal{R}$. Moreover, $\mathcal{R}'$ is clearly independent. □

In recent work, Lin & Chen (2005) 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 the program $\{r_1, \ldots, r_k, u_1, \ldots, u_m\}$ 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 & Chen (2005) for particular problem classes additionally enforce that the corresponding schema is closed under intersection. In fact, Theorem 5 constitutes a generalization of observations made by Lin & Chen (2005).

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

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

- $\mathsf{dom}(\mathrm{LC}_{0\text{-}1\text{-}0}) = \{(\{r\},\emptyset) \mid$ *r is a rule with* $B^+(r) \cap (H(r) \cup B^-(r)) \neq \emptyset\}$*;*
- $\mathsf{dom}(\vartheta\text{-}\mathrm{LC}_{0\text{-}1\text{-}0}) = \{(\{r\},\emptyset) \mid$ *r is a rule such that, for each* $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $B^+(r\vartheta) \cap (H(r\vartheta) \cup B^-(r\vartheta)) \neq \emptyset\}$*; and*
- $\mathcal{R}(M,N) = \top$*, for every* $(M,N) \in \mathsf{dom}(\mathcal{R})$*, with* $\mathcal{R} \in \{\mathrm{LC}_{0\text{-}1\text{-}0}, \vartheta\text{-}\mathrm{LC}_{0\text{-}1\text{-}0}\}$.

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

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

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

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

As mentioned above, Lin & Chen (2005) are concerned with conditions making k-m-n problems true, for small k, m, n. The following proposition rephrases a result of that endeavor:

Proposition 1 (Lin & Chen 2005) *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 & Chen (2005) for propositional programs.

Non-Monotone Replacement Schemas

We already have introduced a non-monotone replacement schema in Definition 8. It has the following properties.

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

We note that the fact that LSH is not $\equiv_s$-preserving already follows from an analogous result in the propositional case (Eiter *et al.* 2004). However, to illustrate this property, consider the following example in the non-ground setting: Take P as consisting of the single rule $r = p(X) \vee q(X) \leftarrow o(X,Y)$. Clearly, r is HCF in $P = \{r\}$, 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.,

$$P'' = \{p(Y) \leftarrow q(Y);\ q(X) \leftarrow p(X);\ o(a,b)\},$$

for which we get that $\mathcal{AS}(P \cup P'') = \{p(a), q(a), o(a,b)\}$ 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* $(M,N) \in \mathsf{dom}(\mathrm{FOLD})$, $\mathrm{FOLD}(M,N) = \phi$*, where* $\phi(P)$ *holds iff, for each head atom b in P, each* $\vartheta_a : \mathcal{V}_a \to \mathcal{C}$ *and each* $\vartheta_b : \mathcal{V}_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$*, with a as above.*

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

For illustration, consider

$$P = \{p(X,X) \leftarrow q(X), \mathrm{not}\ o(X);\\ p(Y,Y) \leftarrow q(Y), o(Y)\}.$$

We can apply FOLD to P since no atom of form $o(\cdot)$ occurs in a head of P. The result of the replacement is $P' = \mathrm{FOLD}[P] = \{p(Y,Y) \leftarrow q(Y)\}$. By the theorem above, $P \equiv_u P'$. For instance, adding $Q = \{q(a)\}$ yields $\mathcal{AS}(P \cup Q) = \mathcal{AS}(P' \cup Q) = \{p(a,a), q(a)\}$. On the other hand, adding $Q' = \{q(a);\ o(X) \leftarrow p(X,Y)\}$ results in $\mathcal{AS}(P \cup Q') = \emptyset$, while $\mathcal{AS}(P' \cup Q') = \{p(a,a), q(a), o(a)\}$. This shows that FOLD is not $\equiv_s$-preserving; a corresponding counterexample can also be constructed for the propositional setting. Furthermore, FOLD is applicable to the program $P \cup Q$ as well, but it is not applicable to $P \cup Q'$. Since $Q' \supset Q$, 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 & Dix (1999).

Definition 14 *Schema* RED^+ *is given as follows:*

- $\mathsf{dom}(\mathrm{RED}^+) = \{(\{r\},\{t\}) \mid$ *r, t are rules such that* $H(r) = H(t)$ *and* $B(r) = B(t) \cup \{\mathrm{not}\ a\}\}$*; and*
- *for every* $(M,N) \in \mathsf{dom}(\mathrm{RED}^+)$, $\mathrm{RED}^+(M,N) = \phi$*, where* $\phi(P)$ *holds iff, for each head atom b in P, each* $\vartheta_a : \mathcal{V}_a \to \mathcal{C}$*, and each* $\vartheta_b : \mathcal{V}_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$*, with a as above.*

Note that RED^+ is, to some extent, a simplification of FOLD, where the second rule, s, of M 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 it is 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 (Eiter *et al.* 2004).

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. For space reasons, we omit involved proofs in Cavour of intuitive explanations.

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

Theorem 12 *The applicability problem for each of the replacement schemas* TAUT, CONTRA, RED^- *is in* LOGSPACE.

This result is easily established by observing that applicability of these schemas can be checked for each rule (resp., pair of rules in case of RED^-) independently by purely syntactical checks. Each syntactic check can be realized in a straightforward manner using tow pointers to the input, which require logarithmic space.

The remaining independent replacement schemas, involving two rules, are more complex, however. Preparatory for our result, a special case of the subsumption problem is useful:

Lemma 1 *Given two sets, A and B, of atoms (without function symbols), the problem of deciding whether A subsumes B, i.e., whether there exists a substitution ϑ from the variables of A into the variables and constants of B such that $A\vartheta \subseteq B$, is* NP*-complete.*

Proof. Membership is by guessing a substitution ϑ from the variables of A into the variables and constants of B and by checking in polynomial time whether $A\vartheta \subseteq B$.

As for NP-hardness, consider a directed graph G over the vertices $\{1, ..., n\}$. Define $B = \{e(r,g), e(g,r), e(r,b), e(b,r), e(g,b), e(b,g)\}$ and $e(X_i, X_j) \in A$ iff (i,j) is an edge in G. This establishes a polynomial-time constructible reduction from the NP-hard 3-colorability problem into the subsumption problem such that G is 3-colorable iff A subsumes B. □

Theorem 13 *The applicability problem for each of the replacement schemas* NONMIN, S-IMPL, *and* SUB *is* NP*-complete.* NP*-hardness holds even if the arities of the predicates in the given program are bounded by a constant.*

Proof. We first remark that it is sufficient to show membership in NP for SUB and NP-hardness for NONMIN in order to establish the result.

For showing the membership of checking SUB-eligibility in NP, guess any pair of rules r, s in the given program P, as well as a corresponding substitution $\vartheta : \mathcal{V}_s \to \mathcal{V}_r \cup \mathcal{C}_r$. Checking the conditions of SUB-eligibility then amounts to simple syntactical checks which can be done in polynomial time.

For the hardness part, consider a program $P = \{r, s\}$ consisting of two rules, where both rules are positive constraints, i.e., $H(r) = H(s) = B^-(r) = B^-(s) = \emptyset$. In this particular case, P is NONMIN-eligible iff there exists a substitution ϑ such that $B^+(s\vartheta) \subseteq B^+(r)$, i.e., iff $B^+(s)$ subsumes $B^+(r)$. Hence, even in this simplified case the applicability problem amounts to the subsumption problem which is NP-complete, as shown above. □

We now turn to non-monotone replacements.

Theorem 14 *The applicability problem for the replacement schema* 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. It can be shown that this test can be performed by taking only a restricted number of groundings into account such that the resulting dependency graph has a polynomial number of vertices. Thus, the HCF test amounts to test reachability in an implicitly represented graph, which is PSPACE-complete. Taking also the LSH domain check into account, we remark that this check can be performed independently and essentially amounts to the problem of finding a most general unifier for $H(r)$. The latter problem is well-known to be (at least) linear time solvable. Hence, we obtain PSPACE-completeness for LSH-eligibility in the general case. Also note that, in the practical relevant setting of programs having bounded predicate arities, LSH-eligibility can be tested in NLOGSPACE, and thus in polynomial time. Here, the implicit graph can be effectively constructed using logarithmic workspace. Hardness follows by reducing the reachability problem in directed graphs to deciding whether a rule r is HCF in P for propositional P, i.e., when all predicates in $\mathcal{A}_P$ have arity 0.

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 in 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-eligibility can be decided by a polynomial number of tests for graph isomorphism. The graph-isomorphism problem 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.*

RED^+-eligibility for a program P can be decided by checking whether there exists a negative atom a in P such that for each atom b in a head of P, any of the substitutions $\vartheta_a : \mathcal{V}_a \rightarrow \mathcal{C}$ and $\vartheta_b : \mathcal{V}_b \rightarrow \mathcal{C}$ yields $a\vartheta_a \neq b\vartheta_b$. These checks can be shown to be LOGSPACE-complete by reducing undirected graph reachability to it, which very recently has been shown to be LOGSPACE-complete (Reingold 2005). Since there is a polynomial number of candidate atoms a, and each check can be performed independently, the result for RED^+-eligibility follows.

Discussion and 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 context. 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 complexity. Note that for all schemas considered in this paper, the tests for applicability are cheaper than the complexity of computing an answer set (which is $\mathrm{NEXP}^{\mathrm{NP}}$-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*.

An implementation of such a scan for replacements in non-ground logic programs is currently under development, and a first prototype system is already available at

```
http://www.kr.tuwien.ac.at/research/
eq/simpl/.
```

This tool scans an input program and outputs those rules which are SUB-eligible (recall that SUB is the most general independent replacement schema over two rules we have considered here) or LSH-eligible. The concrete checks for applicability are realized via polynomial-time computable reductions to reasoning problems in ASP itself. Those reductions are implemented in PERL and the resulting ASP-tasks are relegated to DLV (Leone *et al.* 2002), which is thus used as a black-box solver within this system. In particular, we reduced the applicability problem for SUB to a conjunctive query problem, matching the NP-completeness of the implemented task. Furthermore, the applicability problem for LSH is reduced to brave reasoning over Horn programs (which is EXPTIME-complete, and thus mildly harder than the encoded problem).

Finally, we mention some related and future work. Transformations of logic programs have been extensively studied by Pettorossi & Proietti (1994; 1996; 1998), however in the setting of (positive) logic programs *including function symbols*. Hence, the focus in that work is different to ours in the sense that the expressivity of functions is additionally exploited in order to simplify the structure of programs. Still, there are some issues (e.g., subsumption) common with the ASP-setting we studied here, and future work calls for a closer comparison between techniques used in (functional) logic programming and ASP.

Further work also includes the extension of replacement schemas to the framework of first-order equilibrium logic (Pearce & Valverde 2004a), which generalizes logic programs under the answer-set semantics to full logical theories. Another direction of research is to investigate relations to recent work by Ferraris (2005), who showed that strong equivalence is implicit with modular rewritings of (propositional) programs that preserve equivalence.

Appendix: Proofs

Preparatory for the proofs below, we need some additional concepts and characterizations.

In analogy to the Herbrand base of a program, we call the set of all ground atoms over language $\mathcal{L}$ the Herbrand base of $\mathcal{L}$, denoted $B_{\mathcal{L}}$. For a set $A \subseteq \mathcal{A}$ of predicate symbols and a set $C \subseteq \mathcal{C}$ of constants, we write $B_{A,C}$ to denote the set of all ground atoms constructed from the predicate symbols from A and the constants from C.

Following Eiter *et al.* (2005), we briefly recall some important characterizations for deciding different kinds of equivalence.

Let P be a program, $I, J \subseteq B_{\mathcal{A},C}$ interpretations, and $C \subseteq \mathcal{C}$. Then, the triple $(J, I)_C$ is an *SE-model* of P iff (i) $J \subseteq I$, (ii) $I \models grd(P, C)$, and (iii) $J \models grd(P, C)^I$. Furthermore, given $C \subseteq \mathcal{C}$, we define

$$SE_C(P) = \{(J, I)_C \mid (J, I)_C \text{ is an SE-model of } P\},$$

and $SE(P) = \bigcup_{C \subseteq \mathcal{C}} SE_C(P)$.

The following result captures the key property of SE-models and extends a related characterization for propositional programs due to Turner (2003):

Proposition 2 (Eiter *et al.* 2005) *Given programs P and P', the following statements are equivalent:*

- $P \equiv_s P'$.
- $SE(P) = SE(P')$.
- *For each finite* $C \subseteq \mathcal{C}$, $SE_C(P) = SE_C(P')$.

In order to capture uniform equivalence, we introduce the concept of UE-models, again following Eiter *et al.* (2005).

Let P be a program and $(J, I)_C \in SE(P)$. Then, $(J, I)_C$ is a *UE-model* of P iff, for every SE-model $(J', I)_C$ of P, $J \subset J'$ implies $J' = I$. We write $UE_C(P)$ to refer to the set $\{(J, I)_C \mid (J, I)_C \text{ is an UE-model of } P\}$.

Proposition 3 (Eiter *et al.* 2005) *For all programs P and P', $P \equiv_u P'$ iff, for each finite C, $UE_C(P) = UE_C(P')$.*

To ease notion, we usually write $SE_C(r)$ (resp., $UE_C(r)$) instead of $SE_C(\{r\})$ (resp., $UE_C(\{r\})$).

A rule or a program is called *propositional* iff it contains only predicate symbols of arity 0 (i.e., propositional atoms). Let $\mathcal{A}_0 \subseteq \mathcal{A}$ be a set of such atoms. SE-models for propositional programs reduce simply to *pairs* (J, I) with $J \subseteq I \subseteq \mathcal{A}_0$ (Turner 2003). Accordingly, UE-models for propositional programs are then obtained from SE-models as in the non-ground case above. With a slight abuse of notation, we denote by $SE(P)$ the set of all SE-models of a propositional program P, and $UE(P)$ refers to the set of all UE-models of P.

In what follows, we shall implicitly consider a bijective mapping between ground atoms over $\mathcal{L}$ and propositional atoms $\mathcal{A}_0$. We extend this mapping to rules, programs, and interpretations. This allows us to make use of known results for propositional programs by applying them to ground programs. In fact, most of the following proofs on properties of replacements rely on reducing these properties to the groundings of the considered programs.

In order to avoid a re-definition of specific replacement schemas whose domains are intended to comprise propositional programs only, but to distinguish between the ground setting (which amounts to propositional programs) and the non-ground setting, we shall make use of the following notion:

Definition 15 *Let $\equiv$ be an equivalence notion. A replacement ϱ is called $\equiv^{prp}$-*preserving *if $P \equiv \varrho(P)$, for each ground ϱ-eligible program P. Analogously, a replacement schema $\mathcal{R}$ is $\equiv^{prp}$-*preserving *if each of its instances is $\equiv^{prp}$-preserving.*

We sometimes refer to a replacement schema $\mathcal{R}$ as being in its *ground setting*, or we talk about a *ground restriction* of $\mathcal{R}$, in case we consider $\mathcal{R}$ applied to ground programs only. This allows us to ease the definition of replacement schemas in such situations—in particular, avoiding the handling of variables.

Proof of Theorem 3

Since both TAUT and CONTRA are independent, we use Theorem 2 to show the two replacement schemas are $\equiv_s$-preserving.

For each $(M, N) \in \mathit{inst}(\text{TAUT}) \cup \mathit{inst}(\text{CONTRA})$, it holds that $N = \emptyset$ and $M = \{r\}$, and thus we have to show, for each such M, $M \equiv_s \emptyset$. Note that any $(J, I)_C$ with $J \subseteq I \subseteq B_{\mathcal{A},C}$ is an SE-model of the empty program.

Towards a contradiction, suppose there is some $(\{r\}, \emptyset) \in \mathit{inst}(\text{TAUT}) \cup \mathit{inst}(\text{CONTRA})$ such that $\{r\} \not\equiv_s \emptyset$. By Proposition 2, there exists some finite $C \subseteq \mathcal{C}$ and some $I, J \subseteq B_{\mathcal{A},C}$ such that $J \subseteq I$ and $(J, I)_C \notin SE_C(r)$.

First, suppose that I is not a model of $grd(r, C)$, i.e., there exists a substitution $\vartheta : \mathcal{V}_r \to C$ such that $I \not\models r\vartheta$. Thus, we have that (a) $B^+(r\vartheta) \subseteq I$, (b) $I \cap B^-(r\vartheta) = \emptyset$, and (c) $I \cap H(r\vartheta) = \emptyset$. If $(\{r\}, \emptyset) \in \mathit{inst}(\text{TAUT})$, then there exists an atom $a \in B^+(r) \cup H(r)$. But from this, $a\vartheta \in B^+(r\vartheta) \cap H(r\vartheta)$ follows, and we get that Conditions (a) and (c) cannot jointly hold. If $(\{r\}, \emptyset) \in \mathit{inst}(\text{CONTRA})$, we have an atom $a \in B^+(r) \cap B^-(r)$, and $a\vartheta$ occurs in both $B^+(r\vartheta)$ and $B^-(r\vartheta)$. It follows that (a) and (b) cannot jointly hold.

Second, suppose J is not a model of $grd(r, C)^I$. The argumentation follows a similar pattern as before, i.e., there exists a $\vartheta : \mathcal{V}_r \to C$ such that J is not a model of $\{r\vartheta\}^I$, and thus $\{r\vartheta\}^I \neq \emptyset$. Moreover, it holds that (d) $B^+(r\vartheta) \subseteq J$ and (e) $J \cap H(r\vartheta) = \emptyset$. For the case that $(\{r\}, \emptyset) \in \mathit{inst}(\text{TAUT})$ we get a contradiction as above. If $(\{r\}, \emptyset) \in \mathit{inst}(\text{CONTRA})$, there is some $a \in B^+(r) \cap B^-(r)$, and thus $a\vartheta \in B^+(r\vartheta) \cap B^-(r\vartheta)$. But $I \cap B^-(r\vartheta) = \emptyset$ and $J \subseteq I$, so $B^+(r\vartheta) \not\subseteq J$, contradicting Condition (d).

We thus proved Part 1 of the theorem. It remains to show Parts 2 and 3. We show only Part 2 in what follows; the proof of Part 3 is similar.

We have to show that TAUT and ϑ-TAUT are equipollent, i.e., that $\text{TAUT}^\star[P] = \vartheta\text{-TAUT}^\star[P]$, for every program P.

Fix some program P. That $\text{TAUT}^\star[P] \subseteq \vartheta\text{-TAUT}^\star[P]$ holds is an immediate consequence of the fact that for every $(\{r\}, \emptyset) \in \mathit{inst}(\text{TAUT})$, $a \in H(r) \cap B^+(r)$ implies $a\vartheta \in H(r\vartheta) \cap B^+(r\vartheta)$, for all $\vartheta : \mathcal{V}_r \to \mathcal{C}$.

Now suppose that P is not TAUT-eligible. Then, $H(r) \cap B^+(r) = \emptyset$, for each $r \in P$. Take some $r \in P$ and a ground substitution ϑ mapping each variable X in r to a new distinct constant c_X (this is possible since we assume an infinite universe). Then, for any $a \in H(r)$ and any $b \in B^+(r)$, we get $a\vartheta \neq b\vartheta$, since, by assumption, either a and b have different predicate symbols or differ at some variable occurrence. This shows that there exists a substitution such that $H(r\vartheta) \cap B^+(r\vartheta) = \emptyset$. Since this holds for all $r \in P$, we get that P is not ϑ-TAUT-eligible as well. Consequently, $\vartheta\text{-TAUT}^\star[P] \subseteq \text{TAUT}^\star[P]$.

Proof of Theorem 4

We only show that SUB is $\equiv_s$-preserving; the corresponding results for RED$^-$, NONMIN, and S-IMPL are then trivial consequences, since for any program P and any $\mathcal{R} \in \{\text{RED}^-, \text{NONMIN}, \text{S-IMPL}\}$, it holds that $\mathcal{R}^\star(P) \subseteq \text{SUB}^\star(P)$, whenever P is applicable to $\mathcal{R}$.

We use a different strategy here than in the proof of Theorem 3; however, we make use of Theorem 2 again: We show that for each $(M, N) \in \mathit{inst}(\text{SUB})$, $M \equiv_s N$ holds, by considering the relations $grd(M, C) \equiv_s grd(N, C)$, for any finite $C \subseteq \mathcal{C}$, in view of Proposition 2. For the latter, we show that a sequence of applications of the ground restriction of SUB to the program $grd(M, C)$ eventually results in $grd(N, C)$. To this end, we make use of the following result:

Proposition 4 (Lin & Chen 2005) SUB *is $\equiv_s^{prp}$-preserving.*

So let $(M, N) = (\{r, s\}, \{s\})$ be an instance of SUB, and fix some finite $C \subseteq \mathcal{C}$. We show $grd(M, C) \equiv_s grd(N, C)$.

By definition of SUB, there exists some $\vartheta : \mathcal{V}_s \to \mathcal{V}_r \cup \mathcal{C}_r$ such that $H(s\vartheta) \subseteq H(r) \cup B^-(r)$ and $B(s\vartheta) \subseteq B(r)$. We write $grd(M, C)$ as follows:

$$grd(M, C) = grd(s, C) \cup \{r\theta, s\vartheta\theta \mid \theta : \mathcal{V}_r \to C\},$$

with ϑ from above. Now, the ground restriction of SUB is applicable to $\{r\theta, s\vartheta\theta\}$, for each substitution θ. By Proposition 4, we are allowed to apply SUB step-by-step to each set $\{r\vartheta, s\vartheta\theta\} \subseteq grd(M, C)$ and replace it by $\{s\vartheta\theta\}$. The resulting program is (i) strongly equivalent to $grd(M, C)$, and (ii) exactly matches $grd(N, C) = grd(\{s\}, C)$, since, for each θ, $s\vartheta\theta \in grd(\{s\}, C)$.

Proof of Theorem 8

The only-if direction is an immediate consequence of our results about TAUT and CONTRA, as well as of Theorem 6.

For the if-direction, we first show that, for $P = \{r\}$, $P \equiv_s \emptyset$ implies $\{r\vartheta\} \equiv_s \emptyset$, for every $\vartheta : \mathcal{V}_r \to \mathcal{C}$.

Towards a contradiction, assume $P \equiv_s \emptyset$ but $\{r\vartheta\} \not\equiv_s \emptyset$, for some $\vartheta : \mathcal{V}_r \to C, C \subseteq \mathcal{C}$. Then, there exist interpretations $I, J \subseteq B_{\mathcal{A},C}$ such that $(J, I)_C \in SE(\emptyset)$ and $(J, I)_C \notin SE(\{r\vartheta\})$. By Proposition 2, we can assume that C is finite. Hence, since $(J, I)_C \in SE(\emptyset)$, the hypothesis that $\{r\} \equiv_s \emptyset$ implies that $(J, I)_C \in SE(\{r\})$. However, since $r\vartheta \in grd(r, C)$, we have that $I \models r\vartheta$ and $J \models r\vartheta^I$, i.e. $(J, I)_C \in SE(\{r\vartheta\})$, a contradiction.

So, we have shown that $\{r\} \equiv_s \emptyset$ implies $\{r\vartheta\} \equiv_s \emptyset$, for every $\vartheta : \mathcal{V}_r \to \mathcal{C}$. Therefore, by Proposition 1, $\{r\vartheta\}$ is LC$_{0\text{-}1\text{-}0}$-applicable for every $\vartheta : \mathcal{V}_r \to \mathcal{C}$. By definition, this implies that $\{r\}$ is ϑ-LC$_{0\text{-}1\text{-}0}$-eligible, and thus LC$_{0\text{-}1\text{-}0}$-eligible as well.

Proof of Theorem 9

First, we need an auxiliary result which deals with a *weakening* of replacements and which is also used for further proofs below.

Lemma 2 *Let $\varrho = (\phi, M, N)$ be an $\equiv$-preserving replacement with $SE(M) \subseteq SE(N)$ and M being classically equivalent to N. Then, for any $M' \subseteq M$, $\varrho' = (\phi, M, N \cup M')$ is $\equiv$-preserving. Furthermore, for $M' = M$, ϱ' is $\equiv_s$-preserving.*

We also exploit the following result for the ground case:

Proposition 5 (Eiter *et al.* 2004) LSH *is $\equiv_u^{prp}$-preserving.*

We now commence with the proof of Theorem 9. We show that for any instance ϱ of LSH and each ϱ-eligible program P, $P \equiv_u \varrho[P]$ holds.

Consider some $\varrho \in inst(\text{LSH})$ and some ϱ-eligible program P. In view of Proposition 3, we show that, for each finite $C \subseteq \mathcal{C}$, $UE_C(P) = UE_C(\varrho[P])$, i.e., $grd(P, C) \equiv_u grd(\varrho[P], C)$ (note that $grd(P, C)$ and $grd(\varrho[P], C)$ are *finite* programs). From this, we conclude that each $\varrho \in inst(\text{LSH})$ is $\equiv_u$-preserving. Consequently, LSH is $\equiv_u$-preserving.

So, let $C \subseteq \mathcal{C}$ be finite, $\varrho \in inst(\text{LSH})$, and P ϱ-eligible. By definition, ϱ is of the form $(\phi, \{r\}, N_r)$. We show that $grd(P, C) \equiv_u grd(\varrho[P], C)$.

Let $P' = P \setminus M$. Then,

$$grd(P, C) = grd(P', C) \cup \{r\theta \mid \theta : \mathcal{V}_r \to C\}.$$

Let $r\theta_0, \ldots, r\theta_n$ be all groundings of r with respect to C, and define, for each $0 \leq i \leq n+1$,

$$P_i = grd(P', C) \cup \{r\theta_j \mid i \leq j \leq n\} \cup \bigcup_{k=0}^{i-1} N_r\theta_k.$$

Then, the ground restriction of LSH is applicable to P_i, for any $0 \leq i \leq n$, by choosing $\varrho_i = (\phi, \{r\theta_i\}, N_{r\theta_i})$. Indeed, in view of the condition that for each $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $|H(r\vartheta)| > 1$ must hold, *a fortiori* it holds for θ_i, and thus $r\theta_i$ is properly disjunctive. Moreover, by hypothesis that r is HCF in P, we have, for any finite $C \subseteq \mathcal{C}$ and each $r' \in grd(r, C)$, that r' is HCF in $grd(P, C)$. Hence, $r\theta_i$ is HCF in $grd(P, C)$. Note that $r\theta_i$ is then HCF in P_i as well, since shifting does not change the positive dependency graph. We have two cases: First, if $r\theta_i \in grd(P', C)$, i.e., if the rule is also contained in the remaining grounding, then we just add $N_r\theta_i$ to P_i. The resulting program is then P_{i+1} and since it can be verified that, for any disjunctive rule r, $SE(r) \subseteq SE(N_r)$, we get by Lemma 2, $P_i \equiv_s P_{i+1}$, which implies $P_i \equiv_u P_{i+1}$. Second, if $r\theta_i \notin grd(P', C)$, we in fact apply ϱ_i. By Proposition 5, $P_i \equiv_u \varrho_i[P_i]$. Moreover, $\varrho_i[P_i] = P_{i+1}$. Hence, we have shown for each $0 \leq i \leq n$, that $P_i \equiv_u P_{i+1}$. Observing finally that $P_0 = grd(P, C)$ and $P_{n+1} = grd(\varrho[P], C)$, we arrive at $grd(P, C) \equiv_u grd(\varrho[P], C)$.

Proof of Theorem 10

As in the proofs before, we first start with a corresponding result for the ground case.

Lemma 3 FOLD *is $\equiv_u^{prp}$-preserving.*

Proof. Let $(\{r, s\}, \{t\}) \in \mathsf{dom}(\text{FOLD})$ such that r, s, t are propositional rules and let a be an atom with $a \in B^-(r) \cap B^+(s)$. We show that, for any propositional program P such that a does not occur in rule heads of P, the programs $P' = P \cup \{r, s\}$ and $P'' = P \cup \{t\}$ satisfy $UE(P') = UE(P'')$. We have the following observations:

1. $\{r, s\}$ and $\{t\}$ have the same classical models; from this we immediately get that, for each propositional interpretation I, $(I, I) \in UE(P')$ iff $(I, I) \in UE(P'')$;
2. for any program Q without a in its head and for any propositional interpretation I with $a \in I$, it holds that $(I, I) \in UE(Q)$ implies $(I \setminus \{a\}, I) \in UE(Q)$ (this observation thus applies to both P and P'');
3. for each I with $a \notin I$, it holds that $(J, I) \in SE(\{r, s\})$ iff $(J, I) \in SE(\{t\})$; basically, this follows from the following facts: (i) each $J \subseteq I$ is a model of $\{r\}^I$, since $a \notin J$ but $a \in B^+(r)$, and (ii) $\{s\}^I = \{t\}^I$, by definition.

It follows that $UE(P') = UE(P'')$, and thus $P' \equiv_u P''$, in view of Proposition 3. By the construction of P' and $P'' = \varrho[P']$, which is obtained from any replacement $\varrho = (\phi, \{r, s\}, \{t\}) \in inst(\text{FOLD})$, we conclude that FOLD is $\equiv_u^{prp}$-preserving. □

Also observe that $SE(\{r, s\}) \subseteq SE(t)$ holds for any pair $(\{r, s\}, \{t\}) \in \mathsf{dom}(\text{FOLD})$.

We proceed with the proof of Theorem 10. Having now Lemma 3 at hand, we can use a similar strategy as done in the previous results. That is, we show that, for any instance ϱ of FOLD and each ϱ-eligible program P, $P \equiv_u \varrho[P]$.

So, let $\varrho = (\phi, \{r, s\}, \{t\})$ be an instance of FOLD, along with a renaming δ and an atom a as in Definition 13. Then, $\phi(P)$ holds for any P such that, for each head atom b in P, each $\vartheta_a : \mathcal{V}_a \to \mathcal{C}$, and each $\vartheta_b : \mathcal{V}_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$.

Let P be a ϱ-eligible program and $P' = P \setminus \{r, s\}$. We show, for each finite $C \subseteq \mathcal{C}$, $grd(P, C) \equiv_u grd(\varrho[P], C)$, i.e., $UE_C(P) = UE_C(\varrho[P])$. By Proposition 3, $P \equiv_u \varrho[P]$ follows, and thus, by definition, ϱ is $\equiv_u$-preserving.

Taking the renaming δ into account, we have that

$$\begin{aligned} grd(P, C) &= grd(P', C) \cup \{r\theta, s\theta \mid \theta : \mathcal{V}_P \to C\} \\ &= grd(P', C) \cup \{r\delta\theta, s\theta \mid \theta : \mathcal{V}_s \to C\}. \end{aligned}$$

Let $\theta_0, \ldots, \theta_n$ be all substitutions from $\mathcal{V}_s$ to C as used above and define, for each $0 \leq i \leq n+1$,

$$P_i = grd(P', C) \cup \{r\delta\theta_j, s\theta_j \mid i \leq j \leq n\} \cup \{t\theta_k \mid 0 \leq k < i\}.$$

Note that the ground restriction of FOLD is applicable to P_i, for each $0 \leq i \leq n$, and P_{i+1} is among the possible results of FOLD applied to P_i. More precisely, for each P_i we replace the two rules $r\delta\theta_i$ and $s\theta_i$ by $t\theta_i$, and if $\{r\delta\theta_i, s\theta_i\} \cap grd(P', C) \neq \emptyset$, we take the corresponding weakened replacement, as in Lemma 2 (using the fact that $\{r\delta\theta_i, s\theta_i\}$ is classically equivalent to $\{t\theta_i\}$). Hence, in either case we end up with P_{i+1}. By definition, $a\theta_i \in B^-(r\delta\theta_i) \cap B^+(s\theta_i)$, $H(r\delta\theta_i) = H(s\theta_i) = H(t\theta_i)$, and $B(r\delta\theta_i) \setminus \{\text{not } a\theta_i\} = B(s\theta_i) \setminus \{a\theta_i\} = B(t\theta_i)$. Now, $a\theta_i$ does not occur in any head of P_i, which is guaranteed by the condition for $\phi(P)$. By Lemma 3, we then get, for each $0 \leq i \leq n$, $P_i \equiv_u P_{i+1}$. This concludes the proof, since $P_0 = P$ and $P_{n+1} = \varrho[P]$.

Proof of Theorem 11

We again make use of a corresponding result for the ground case.

Proposition 6 (Brass & Dix 1999) *The schema* RED^+ *is* $\equiv_o^{prp}$*-preserving.*

Also observe that, for each $(\{r\}, \{t\}) \in \mathsf{dom}(\mathrm{RED}^+)$, $SE(r) \subseteq SE(t)$ holds.

We show that for any instance ϱ of RED^+ and any ϱ-eligible program P, $P \equiv_o \varrho[P]$. Consider an instance $\varrho = (\phi, \{r\}, \{t\})$ of RED^+ and some atom a such that $a \in B^-(r)$ and $B(t) = B(r) \setminus \{\text{not } a\}$. By definition, $\phi(P)$ holds if, for each head atom b in P, each $\vartheta_a : \mathcal{V}_a \to \mathcal{C}$, and each $\vartheta_b : \mathcal{V}_b \to \mathcal{C}$, $a\vartheta_a \neq b\vartheta_b$.

Let $P' = P \setminus \{r\}$, and consider

$$grd(P) = grd(P', \mathcal{C}_P) \cup \{r\theta \mid \theta : \mathcal{V}_r \to \mathcal{C}_P\}$$

As before, we successively replace each rule from $\{r\theta \mid \theta : \mathcal{V}_r \to \mathcal{C}_P\}$ by the corresponding rule $t\theta$ (possibly keeping $r\theta$ within the program, whenever $r\theta \in grd(P', \mathcal{C}_P)$). By hypothesis that P is ϱ-eligible, the ground restriction of RED^+ is applicable to each program in this sequence. By Lemma 2 and Proposition 6, one shows that each of this steps retains ordinary equivalence. Therefore, we get that $P \equiv_o \varrho[P]$.

References

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press.

Brass, S., and Dix, J. 1999. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming* 38(3):167–213.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2004. Simplifying Logic Programs Under Uniform and Strong Equivalence. In Lifschitz, V., and Niemelä, I., eds., *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-04*), volume 2923 of *LNCS*, 87–99. Springer Verlag.

Eiter, T.; Fink, M.; Tompits, H.; and Woltran, S. 2005. Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. In Veloso, M., and Kambhampati, S., eds., *Proceedings of the 20th National Conference on Artificial Intelligence* (*AAAI-05*), 695–700. AAAI Press.

Eiter, T.; Fink, M.; and Woltran, S. 2005. Semantical Characterizations and Complexity of Equivalences in Stable Logic Programming. Technical Report INFSYS RR-1843-05-01, Institut für Informationssysteme, Technische Universität Wien, Austria. Accepted for publication in *ACM Transactions on Computational Logic*.

Eiter, T.; Tompits, H.; and Woltran, S. 2005. On Solution Correspondences in Answer Set Programming. In Kaelbling, L., and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI-05*), 97–102.

Ferraris, P. 2005. On Modular Translations and Strong Equivalence. In Baral, C.; Greco, G.; Leone, N.; and Terracina, G., eds., *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR-05*), volume 3552 of *LNCS*, 79–91. Springer Verlag.

Gelfond, M., and Leone, N. 2002. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2002. The DLV System for Knowledge Representation and Reasoning. Technical Report cs.AI/0211004, arXiv.org. Accepted for publication in *ACM Transactions on Computational Logic*.

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

Lin, F., and Chen, Y. 2005. Discovering Classes of Strongly Equivalent Logic Programs. In Kaelbling, L., and Saffiotti, A., eds., *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI-05*), 516–521.

Osorio, M.; Navarro, J. A.; and Arrazola, J. 2001. Equivalence in Answer Set Programming. In Pettorossi, A., ed., *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation* (*LOPSTR-01*), volume 2372 of *LNCS*, 57–75. Springer Verlag.

Pearce, D., and Valverde, A. 2004a. Towards a First Order Equilibrium Logic for Nonmonotonic Reasoning. In Alferes, J. J., and Leite, J. A., eds., *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA-04*), volume 3229 of *LNCS*, 147–160. Springer Verlag.

Pearce, D., and Valverde, A. 2004b. Synonymous Theories in Answer Set Programming and Equilibrium Logic. In de Mántaras, R. L., and Saitta, L., eds., *Proceedings of the 16th European Conference on Artificial Intelligence* (*ECAI-04*), 388–392. IOS Press.

Pettorossi, A., and Proietti, M. 1994. Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming* 19/20:261–320.

Pettorossi, A., and Proietti, M. 1996. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys* 28(2):360–414.

Pettorossi, A., and Proietti, M. 1998. Transformation of Logic Programs. In Gabbay, D. M.; Hogger, C. J.; and Robinson, J. A., eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, 697–787. Oxford University Press.

Reingold, O. 2005. Undirected ST-Connectivity in Log-Space. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing* (*STOC-05*), 376–385. ACM Press.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence* 138:181–234.

Turner, H. 2003. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *Theory and Practice of Logic Programming* 3(4-5):602–622.

Wang, K., and Zhou, L. 2005. Comparisons and Computation of Well-founded Semantics for Disjunctive Logic Programs. *ACM Transactions on Computational Logic* 6(2):295–327.

Woltran, S. 2005. 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`.