In: Proc. 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7), I. Niemelä and V. Lifschitz, (eds), LNCS, © 2004 Springer.

Simplifying Logic Programs under Uniform and Strong Equivalence*

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

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

Abstract. We consider the simplification of logic programs under the stable-model semantics, with respect to the notions of strong and uniform equivalence between logic programs, respectively. Both notions have recently been considered for nonmonotonic logic programs (the latter dates back to the 1980s, though) and provide semantic foundations for optimizing programs with input. Extending previous work, we investigate syntactic and semantic rules for program transformation, based on proper notions of consequence. We furthermore provide encodings of these notions in answer-set programming, and give characterizations of programs which are semantically equivalent to positive and Horn programs, respectively. Finally, we investigate the complexity of program simplification and determining semantical equivalence, showing that the problems range between coNP and Π_2^P complexity, and we present some tractable cases.

1 Introduction

Implementations of answer-set solvers such as DLV [4], Smodels [21], or ASSAT [13] led to the consideration of practical applications of nonmonotonic logic programs in the last years, but also renewed interest in the study of foundational properties. In particular, semantical notions of equivalence between logic programs such as *strong equivalence* have been studied (cf. [11, 22, 23, 17, 12, 3]): Programs P_1 and P_2 are strongly equivalent, if, for any set R of rules, the programs $P_1 \cup R$ and $P_2 \cup R$ are equivalent under the stable semantics, i.e., have the same set of stable models. This can be used to simplify a logic program P [23, 16]: if a subprogram Q of P is strongly equivalent to a (simpler) program Q', then we can replace Q by Q'.

Since strong equivalence is rather strict, the more liberal notion of *uniform equivalence* [20, 14] has been considered in [5, 18], where R is restricted to sets of *facts*. A hierarchical component C within a program P may be replaced by a uniformly equivalent set of rules C', providing the global hierarchical component structure of P is not affected (which is a simple syntactic check). As recently discussed by Pearce and Valverde [18], uniform and strong equivalence are essentially the only concepts of equivalence obtained by varying the logical form of the program extensions.

* This work was partially supported by the Austrian Science Fund (FWF) under project Z29-N04, and the European Commission under projects FET-2001-37004 WASP and IST-2001-33570 INFOMIX.

This paper continues and extends the work in [5], which focused on semantical characterizations of uniform equivalence for disjunctive logic programs (DLPs). More specifically, we consider here the issue of simplifying DLPs under uniform and strong equivalence under different aspects. Our main contributions are briefly summarized as follows:

(1) We consider a method using *local transformation rules* to simplify programs, where we deal both with syntactic and semantic transformation rules. Syntactic transformation rules for strong equivalence have previously been considered by Osorio et al. [16]. Besides rules from there and from [2], we also examine the recent notion of *s-implication* [24], as well as a new transformation rule for head-cycle free rules, called *local shifting*. Both preserve uniform equivalence, and the former also strong equivalence. The semantic transformation rules employ logical consequence and remove redundant rules and literals. The method interleaves syntactic and semantic transformation rules, respecting that the former have much lower (polynomial) complexity compared to the intractability of the semantic rules.

(2) We provide answer-set programming (ASP) solutions for checking whether $P \models_s r$ resp. $P \models_u r$, where $\models_s$ denotes the consequence operator under strong-equivalence models (SE-models) [22, 23] and $\models_u$ refers to consequence under uniform-equivalence models (UE-models) [5]. Note that deciding $P \models_u r$ is Π_2^P-complete [5], and hence the full power of DLPs is needed to decide this problem, whilst deciding $P \models_s r$ is "only" coNP-complete. We remark that Pearce and Valverde [18] have provided a tableau system for deciding uniform equivalence, and that Janhunen and Oikarinen [9, 10] provided an alternative method in terms of ASP for testing strong and ordinary equivalence between normal logic programs.

(3) Beyond local transformations, we present general conditions under which programs possess equivalent programs belonging to syntactic subclasses of DLPs (which have special algorithms for ASP computation). In particular, we provide semantical characterizations (in terms of conditions on models) of programs which are uniformly resp. strongly equivalent to programs which are positive or Horn. Furthermore, we give similar conditions in terms of classical consequence for strong equivalence.

(4) We analyze the computational complexity of the problems encountered. In particular, we consider the cost of the semantic simplification rules, and the cost of deciding whether a given DLP is equivalent to a syntactically simpler program. While most of these problems are unsurprisingly intractable (coNP-complete, and in few cases for uniform equivalence Π_2^P-complete), we obtain a polynomial-time result for deciding whether a normal logic program without constraints of form $\leftarrow A_1, \ldots, A_n$ is strongly equivalent to some positive DLP or to some Horn program, respectively.

Given the availability of efficient answer-set solvers, such as the systems mentioned previously, enabling automated simplifications of logic programs has become an important issue. This is even more the case since with the development of applications using ASP solvers, an ever growing number of programs are automatically generated, leaving the burden of optimizations to the underlying ASP system. Our results might well be used for *offline optimizations* of application programs in ASP solvers.

For space reasons, proofs are omitted; they are given in an extended version of this paper.

2 Preliminaries

We deal with propositional disjunctive logic programs, containing rules r of form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n,$$

$n \geq m \geq l \geq 0$, where all a_i are atoms from a finite set of propositional atoms, At, and not denotes default negation. The *head* of r is the set $H(r) = \{a_1, \ldots, a_l\}$, and the *body* of r is the set $B(r) = \{a_{l+1}, \ldots, a_m, not\, a_{m+1}, \ldots, not\, a_n\}$. We also define $B^+(r) = \{a_{l+1}, \ldots, a_m\}$ and $B^-(r) = \{a_{m+1}, \ldots, a_n\}$. Moreover, for a set of atoms $A = \{a_1, \ldots, a_n\}$, $not\, A$ denotes the set $\{not\, a_1, \ldots, not\, a_n\}$.

A rule r is *normal*, if $l \leq 1$; *definite*, if $l = 1$; *positive*, if $n = m$; and *Horn*, if it is normal and positive. If $H(r) = \emptyset$ and $B(r) \neq \emptyset$, then r is a *constraint*; and if moreover $B^-(r) = \emptyset$, then r is a *positive constraint*. If $B(r) = \emptyset$, r is a *fact*, written as $a_1 \vee \cdots \vee a_l$ if $l > 0$, and as $\perp$ otherwise.

A *disjunctive logic program* (DLP), P, is a finite set of rules. P is called a *normal logic program* (NLP) (resp., a *definite*, *positive*, *Horn* program), if every rule in P is normal (resp., definite, positive, Horn). We also define $P^+ = \{r \in P \mid B^-(r) = \emptyset\}$.

In a DLP P, an atom a *positively depends* on an atom b if $a \in H(r)$ for some rule $r \in P$, and either $b \in B^+(r)$, or some $c \in B^+(r)$ positively depends on b; $r \in P$ is *head-cycle free* (HCF) *in* P if no distinct atoms $a, b \in H(r)$ mutually positively depend on each other. A DLP P is HCF [1] if each $r \in P$ is HCF in P.

We recall the stable-model semantics for DLPs [8, 19]. Let I be an interpretation, i.e., a subset of At. Then, an atom a is *true under* I, symbolically $I \models a$, iff $a \in I$, and *false under* I otherwise. For a rule r, $I \models H(r)$ iff some $a \in H(r)$ is true under I, and $I \models B(r)$ iff (i) each $a \in B^+(r)$ is true under I, and (ii) each $a \in B^-(p)$ is false under I. I *satisfies* r, denoted $I \models r$, iff $I \models H(r)$ whenever $I \models B(r)$. Furthermore, I is a *model* of a program P, denoted $I \models P$, iff $I \models r$, for all $r \in P$. As usual, $P \models r$ iff $I \models r$, for each model I of P.

The *Gelfond-Lifschitz reduct* of a program P *relative to* a set of atoms I is the positive program $P^I = \{H(r) \leftarrow B^+(r) \mid r \in P,\, B^-(r) \cap I = \emptyset\}$. An interpretation I is a *stable model* of a program P iff I is a minimal model (under set inclusion) of P^I. The set of all stable models of P is denoted by $\mathcal{SM}(P)$.

Several notions of equivalence between logic programs have been considered in the literature (cf., e.g., [11, 14, 20]). Under stable semantics, two DLPs P and Q are regarded as equivalent, denoted $P \equiv Q$, iff $\mathcal{SM}(P) = \mathcal{SM}(Q)$. The more restrictive forms of *strong equivalence* and *uniform equivalence* are as follows:

Definition 1. *Let P and Q be two DLPs. Then,*

(*i*) *P and Q are* strongly equivalent, *or* s-equivalent, *denoted $P \equiv_s Q$, iff, for any set R of rules, the programs $P \cup R$ and $Q \cup R$ are equivalent, i.e., $P \cup R \equiv Q \cup R$.*
(*ii*) *P and Q are* uniformly equivalent, *or* u-equivalent, *denoted $P \equiv_u Q$, iff, for any set F of non-disjunctive facts, $P \cup F$ and $Q \cup F$ are equivalent, i.e., $P \cup F \equiv Q \cup F$.*

Obviously, $P \equiv_s Q$ implies $P \equiv_u Q$ but not vice versa (see Example 1 below). Both notions of equivalence, however, enjoy interesting semantical characterizations. As shown in [11], strong equivalence is closely related to the non-classical logic of here-and-there, which was adapted to logic-programming terms by Turner [22, 23]:

Definition 2. *Let P be a DLP, and let $X, Y \subseteq At$ such that $X \subseteq Y$. The pair (X, Y) is an* SE-model *of P (over At), if $Y \models P$ and $X \models P^Y$. By $M_s^{At}(P)$ we denote the set of all SE-models over At of P.*

In what follows, we usually leave the set At implicit and write $M_s(P)$ *simpliciter* instead of $M_s^{At}(P)$.

Proposition 1 ([22,23]). *For any DLP P and Q, $P \equiv_s Q$ iff $M_s(P) = M_s(Q)$.*

Recently, the following pendant to SE-models, characterizing uniform equivalence for (finite) logic programs, has been defined [5].

Definition 3. *Let P be a DLP and $(X, Y) \in M_s(P)$. Then, (X, Y) is an* UE-model *of P iff, for every $(X', Y) \in M_s(P)$, it holds that $X \subset X'$ implies $X' = Y$. By $M_u(P)$ we denote the set of all UE-models of P.*

Proposition 2 ([5]; cf. also [18]). *For any DLP P and Q, $P \equiv_u Q$ iff $M_u(P) = M_u(Q)$.*

Example 1. Consider $P = \{a \leftarrow\}$ and $Q = \{a \leftarrow not\, b;\ a \leftarrow b\}$ over $At = \{a, b\}$. As easily checked, we have $M_s(P) = \{(\{a, b\}, \{a, b\}), (\{a\}, \{a, b\}), (\{a\}, \{a\})\}$ and $M_s(Q) = M_s(P) \cup \{(\emptyset, \{a, b\})\}$. Thus, $P \not\equiv_s Q$. However, $(\emptyset, \{a, b\}) \notin M_u(Q)$. In fact, it holds that $M_u(P) = M_s(P)$ and $M_u(Q) = M_u(P)$. Therefore, $P \equiv_u Q$.

Finally, we define consequence relations associated to SE- and UE-models in the usual manner.

Definition 4. *Let P be a DLP, r a rule, and $e \in \{s, u\}$. Then, $P \models_e r$ iff, for each $(X, Y) \in M_e(P)$, $(X, Y) \models_e r$, i.e., $Y \models r$ and $X \models \{r\}^Y$. Furthermore, we write $P \models_e Q$ iff $P \models_e r$, for every $r \in Q$.*

Proposition 3. *For any DLP P and Q, $P \equiv_e Q$ iff $P \models_e Q$ and $Q \models_e P$, $e \in \{s, u\}$.*

3 Local Program Transformations

Given a logic program P and a notion of equivalence $\equiv_e$ ($e \in \{s, u\}$), we aim at a procedure to systematically simplify P to a program P' which is e-equivalent to P. As a starting point, we consider *local modifications*, i.e., iterative modifications of P on a rule-by-rule basis. Later on, in Section 5, we present results for replacing—under certain conditions—the *entire program* by a simpler, e-equivalent program.

Figure 1 gives the general structure of such a simplification algorithm, using different kinds of simplifications: A first simplification can be obtained by analyzing the syntax of the program and by applying appropriate syntactic transformation rules. After this "static" analysis, the notion of consequence induced by the given type of equivalence can be used for further semantic simplifications by means of consequence tests (see also next section). Systematic consequence testing can be applied to simplify rules by identifying redundant literals for removal, as well as for program simplification by identifying redundant rules.

Algorithm $\mathit{Simplify}(P, \equiv_e)$

Input: A DLP P and a notion of equivalence $\equiv_e$.
Output: A simplified DLP P' such that $P' \equiv_e P$.

var P' : DLP, I : changeInfo ;
$P' := P$; $\quad I := \mathit{all}$;
while $I \neq \emptyset$ **do**
  $P' := \mathit{Simplify\_Syntactic}(P', \equiv_e, I)$;
  $P' := \mathit{Remove\_Redundant\_Rule}(P', \equiv_e, I)$;
  if $(I = \emptyset)$ **then** $P' := \mathit{Remove\_Redundant\_Literal}(P', \equiv_e, I)$; **fi**
od
return P';

Fig. 1. Generic simplification algorithm.

The algorithm depicted in Figure 1 interleaves syntactic with semantic simplifications, giving preference to syntactic ones since they are easier to accomplish. Furthermore, it uses a data structure for recording latest changes (changeInfo I) in order to restrict the set of applicable transformation rules as well as to restrict their range of application: By means of this information, in each iteration, only a particular set of syntactic transformation rules has to be tested for a specific rule. Of course, this basic algorithm can be further optimized to fit particular needs. Finally, we remark that $\mathit{Simplify}$ is sound and that it is an anytime algorithm, since we use rewriting rules for a transformation-based calculus, where $P = P_0$ is transformed by applying rules T_i iteratively: $P = P_0 \xrightarrow{T_{i_1}} P_1 \xrightarrow{T_{i_2}} P_2 \cdots P_{k-1} \xrightarrow{T_{i_k}} P_k$, such that $P_j \equiv_e P_{j+1}$, $0 \leq j < k$.

3.1 Syntactic transformation rules

Basic transformation rules. A set of basic syntactic transformation rules has been introduced and initially studied by Brass and Dix [2]. Table 1 briefly summarizes these rules, called *elimination of tautologies* (TAUT), *positive-* and *negative reduction* (RED$^+$ and RED$^-$, respectively), *partial evaluation* (GPPE), *weak partial evaluation* (WGPPE), *elimination of non-minimal rules* (NONMIN), and *elimination of contradictions* (CONTRA).

Osorio et al. [16] reported results for TAUT, RED$^+$, RED$^-$, GPPE, and NONMIN under strong equivalence. The positive results for TAUT, RED$^-$, and NONMIN directly carry over to uniform equivalence. The programs $P = \{a \leftarrow \mathit{not}\, b\}$ and $Q = \{a \leftarrow b;\ b \leftarrow c\}$ show that RED$^+$ and GPPE also fail to preserve uniform equivalence. Moreover, it is easily verified that CONTRA preserves both notions of equivalence.

While GPPE does neither preserve s-equivalence nor u-equivalence, both are preserved, however, by the weak variant WGPPE.

Proposition 4. *Let P be a DLP and consider $r_1, r_2 \in P$ such that $a \in B^+(r_1)$ and $a \in H(r_2)$, for some atom a. Then, $P \cup \{r'\} \equiv_s P$, where r' is given by $H(r_1) \cup (H(r_2) \setminus \{a\}) \leftarrow (B^+(r_1) \setminus \{a\}) \cup \mathit{not}\, B^-(r_1) \cup B(r_2)$.*

Table 1. Syntactic transformation rules.

Name	Condition	Transformation
TAUT	$H(r) \cap B^+(r) \neq \emptyset$	$P' = P \setminus \{r\}$
RED$^+$	$a \in B^-(r_1),\ \nexists r_2 \in P:\ a \in H(r_2)$	$P' = P \setminus \{r_1\} \cup \{r'\}^\dagger$
RED$^-$	$H(r_2) \subseteq B^-(r_1), B(r_2) = \emptyset$	$P' = P \setminus \{r_1\}$
NONMIN	$H(r_2) \subseteq H(r_1), B(r_2) \subseteq B(r_1)$	$P' = P \setminus \{r_1\}$
GPPE	$a \in B^+(r_1), G_a \neq \emptyset,$ for $G_a = \{r_2 \in P \mid a \in H(r_2)\}$	$P' = P \setminus \{r_1\} \cup G_a'^{\ddagger}$
WGPPE	same condition as for GPPE	$P' = P \cup G_a'^{\ddagger}$
CONTRA	$B^+(r) \cap B^-(r) \neq \emptyset$	$P' = P \setminus \{r\}$
S-IMP	$r, r' \in P, r \lhd r'$	$P' = P \setminus \{r'\}$
LSH	$r \in P, \|H(r)\| > 1$, r head-cycle free in P	$P' = P \setminus \{r\} \cup r^{\rightarrow}$

† $r' : H(r_1) \ \leftarrow\ B^+(r_1) \cup not\,(B^-(r_1) \setminus \{a\})$.
‡ $G'_a = \{H(r_1) \cup (H(r_2) \setminus \{a\}) \ \leftarrow\ (B^+(r_1) \setminus \{a\}) \cup not\, B^-(r_1) \cup B(r_2) \mid r_2 \in G_a\}$.

Note that WGPPE creates new rules and may make other transformations applicable. Furthermore, it may be applied exponentially often during iteration steps, creating further (even minimal) rules. Hence, to avoid an exponential explosion in the simplification algorithm, the number of applications of WGPPE may be polynomially bounded.

S-implication (*S-IMP*). Recently, Wang and Zhou [24] introduced the notion of s-implication:

Definition 5. *A rule r' is an* s-implication *of a rule $r \neq r'$, symbolically $r \lhd r'$, iff there exists a set $A \subseteq B^-(r')$ such that (i) $H(r) \subseteq H(r') \cup A$, (ii) $B^-(r) \subseteq B^-(r') \setminus A$, and (iii) $B^+(r) \subseteq B^+(r')$.*

For example, if $r = a \vee b$ and $r' = a \leftarrow not\, b$, then $r \lhd r'$ (choose $A = \{b\}$).

Proposition 5. *Let P be a DLP, and $r, r' \in P$ such that $r \lhd r'$. Then, $P \setminus \{r'\} \equiv_s P$.*

Example 2. Consider the program $P = \{r_1 : a \leftarrow not\, b;\ r_2 : a \vee b \leftarrow\}$. Since $r_2 \lhd r_1$, P can be simplified to the strongly equivalent program $P' = \{a \vee b \leftarrow\}$.

Local shifting (*LSH*). Finally, we present a new transformation rule, called *local shifting*. It relies on the concept of head-cycle freeness and turns out to preserve uniform equivalence, but not strong equivalence.

For any rule r, define $r^{\rightarrow} = \{a \leftarrow B(r), not\,(H(r) \setminus \{a\}) \mid a \in H(r)\}$ if $H(r) \neq \emptyset$, and $r^{\rightarrow} = \{r\}$ otherwise. Furthermore, for any DLP P, let $P^{\rightarrow} = \bigcup_{r \in P} r^{\rightarrow}$.

Definition 6. *The rule* local shifting (*LSH*) *allows replacing a rule r in a DLP P by $r^{\rightarrow}$, providing r is head-cycle free in P.*

Theorem 1. *Let P be a DLP and $r \in P$ HCF in P. Then, $P \equiv_u (P \setminus \{r\}) \cup r^{\rightarrow}$.*

Example 3. Consider $P = \{a \vee b \leftarrow;\ c \vee d \leftarrow b;\ c \leftarrow a, d;\ d \leftarrow b, c\}$. Here, $a \vee b \leftarrow$ is HCF in P, whilst $c \vee d \leftarrow b$ is not. Hence, under $\equiv_u$, program P can be simplified to $P' = (P \setminus \{a \vee b \leftarrow\}) \cup \{a \leftarrow not\, b; b \leftarrow not\, a\}$.

Table 2 summarizes the results for syntactic transformation rules under both uniform and strong equivalence.

Table 2. Syntactic transformations preserving e-equivalence ($e \in \{s, u\}$).

Eq.	TAUT	RED$^+$	RED$^-$	NONMIN	GPPE	WGPPE	CONTRA	S-IMP	LSH
$\equiv_s$	yes*	no*	yes*	yes*	no*	yes	yes	yes	no
$\equiv_u$	yes	no	yes	yes	no	yes	yes	yes	yes

*Results due to Osorio et al. [16].

3.2 Semantic transformation rules

More powerful than the above syntactic transformation rules for program simplification are semantic ones based on $\models_e$, in the light of Propositions 1–3. However, checking the applicability of these rules is intractable in general, while checking the applicability of the above syntactic rules is tractable.

Brass and Dix [2] considered *supraclassicality* (SUPRA): If $P \models a$, then $P \equiv P \cup \{a \leftarrow\}$. However, SUPRA fails under $\equiv_u$ (and thus also under under $\equiv_s$). A simple counterexample is $P = \{a \leftarrow \mathit{not}\, b;\ a \leftarrow b;\ a \leftarrow b, c;\ b \leftarrow a, c\}$. Clearly, $P \models a$, but P has a UE-model $(\{c\}, \{a, b, c\})$ which is not a UE-model of $P \cup \{a \leftarrow\}$.

Rule simplification. One way is minimizing individual rules r, based on property NONMIN: replace r by a subsumer r' (with $H(r') \subseteq H(r)$ and $B(r') \subseteq B(r)$) such that $(P \setminus \{r\}) \cup \{r'\} \equiv_e P$. Rule simplification (or *condensation*) thus amounts to the systematic search for such r', which must satisfy $P \models_e r'$. The easiest way is literalwise removal, i.e., where r' is r with some literal in r removed.

Rule redundancy. In addition, we can simplify P by removing redundant rules. Proposition 3 implies that $r \in P$ is redundant in P iff $P \setminus \{r\} \models_e r$.

Notice that rule simplification might create redundant rules, as shown by the following example, which is well-known in the context of computing a minimal cover of a set of functional dependencies in databases.

Example 4. Consider the program $P = \{c \leftarrow a, b;\ a \leftarrow b, c;\ b \leftarrow a;\ b \leftarrow c\}$. None of the rules in P is redundant, neither under strong nor under uniform consequence. However, b can be removed in both $c \leftarrow a, b$ and $a \leftarrow b, c$, yielding $P' = \{c \leftarrow a;\ a \leftarrow c;\ b \leftarrow a;\ b \leftarrow c\}$. Here, $b \leftarrow c$ or $b \leftarrow a$ can be eliminated.

We remark that some of the syntactic transformation rules, such as TAUT, RED$^-$, NONMIN, and CONTRA, can be regarded as special cases of semantic transformation rules. However, others such as LSH, cannot be viewed this way.

We may bypass e-consequence and stay with relation $\models$ in some cases. This is true, e.g., for positive programs, which is an easy corollary of the next result, which generalizes an analogous result of [5] for uniform equivalence to e-equivalence.

Theorem 2. *Let P be a DLP and $e \in \{s, u\}$. Then, the following conditions are equivalent:*

(i) $P \models_e r$ *iff* $P \models r$, *for every rule* r.

(*ii*) *For every* $(X, Y) \in M_e(P)$*, it holds that* $X \models P$.

Corollary 1. *For any positive DLP* P *and any rule* r, $P \models_e r$ *iff* $P \models r$, $e \in \{s, u\}$.

Thus, for every positive program, a UE- and SE-canonical form is given by any irredundant prime CNF, written in positive rule form. Hence, minimization of any program which is semantically equivalent to a positive one is possible in polynomial time with an NP-oracle; detecting this property is considered in Section 5.

3.3 Computational cost of $Simplify$ and incomplete simplification

As pointed out above, testing the applicability and execution of each syntactic rule in Section 3.1 valid for $\equiv_e$ is clearly polynomial. Since all transformation rules except WGPPE shrink or remove program rules, $\mathit{Simplify}(P, \equiv_e)$ is polynomial with an oracle for $\equiv_e$ if WGPPE is applied at most polynomially often.

As discussed in Section 6, following from Corollary 1, deciding rule and literal redundancy is intractable in general. Interestingly, the following extension of Corollary 1 yields sound inference in terms of $\models$ for arbitrary DLPs.

Proposition 6. *For any DLP* P *and rule* r, $P^+ \models r$ *implies* $P \models_e r$*, for* $e \in \{s, u\}$.

However, the condition is not necessary; e.g., for $P = \{a \leftarrow b;\ a \leftarrow \mathit{not}\, b\}$ and $r = a \leftarrow \mathit{not}\, b$, we have $P \models_s r$ but $P^+ \not\models r$. Note that $P \models r$ for positive r does not imply $P \models_e r$, as shown by $P = \{a \leftarrow \mathit{not}\, a\}$ and $r = a \leftarrow$.

By exploiting Proposition 6, we can simplify any DLP P using its positive part P^+, without guaranteeing completeness. Notice that if P^+ is Horn (e.g., if P is a NLP), this sound redundancy test is polynomial, and is in coNP for general P^+. Thus, for P^+ being Horn it is cheaper than the sound and complete test for NLPs (which is coNP-complete for both $\models_s$ and $\models_u$), and for arbitrary P^+ for $\models_u$ (which is Π_2^P-complete; cf. Section 6).

4 Consequence Testing

In order to enable the systematic tests from the previous section by means of answer-set solvers, we provide here efficient translations of checking SE- and UE-consequence into (disjunctive) logic programs under the stable-model semantics.

We employ the following notation. For any set I of atoms, we define $I' = \{v' \mid v \in I\}$ and $\bar{I} = \{\bar{v} \mid v \in I\}$, where v' and $\bar{v}$ are globally new atoms, for each atom v. Furthermore, r' (resp., $\bar{r}$) denotes the result of replacing in r each occurrence of an atom v by v' (resp., $\bar{v}$). Informally, primed atoms will be used to characterize X in a UE- or SE-model (X, Y); whilst atoms of form $\bar{v}$ will be used to encode the negation of v.

4.1 Encoding SE-consequence

Based on results in [12, 17, 23], a SAT encoding of $\models_s$ is easily obtained. In what follows, we describe an analogous encoding in terms of normal logic programs.

Definition 7. *For any DLP P and atom set $V = \{v_1, \ldots, v_n\}$, define $S_{P,V}$ as:*

$$
\begin{array}{lll}
(1) & v_i \leftarrow not\, \bar{v}_i; \quad \bar{v}_i \leftarrow not\, v_i, & 1 \leq i \leq n, \\
(2) & v'_i \leftarrow v_i, not\, \bar{v}'_i; \quad \bar{v}'_i \leftarrow not\, v'_i, & 1 \leq i \leq n, \\
(3) & \leftarrow B^+(r), not\, B^-(r), not\, H(r), & r \in P, \\
(4) & \leftarrow B^+(r'), not\, B^-(r), not\, H(r'), & r \in P.
\end{array}
$$

Intuitively, (1) and (2) guess sets $X, Y \subseteq V$ such that $X \subseteq Y$, where atoms in X are represented by primed variables; (3) checks $Y \models P$; and (4) $X \models P^Y$.

The next result describes the desired encoding of $\models_s$.

Theorem 3. *Let P, Q be DLPs, V the set of atoms occurring in $P \cup Q$, and w a fresh atom. Then, $P \models_s Q$ iff the following NLP has no stable model:*

$$
\begin{aligned}
S_{P,V} \cup \{\leftarrow not\, w\} \cup \{w \leftarrow B^+(r), not\, B^-(r), not\, H(r);& \\
w \leftarrow B^+(r'), not\, B^-(r), not\, H(r') \mid r \in Q\}.&
\end{aligned}
$$

4.2 Encoding UE-consequence

Definition 8. *Given a DLP P, a set of atoms $V = \{v_1, \ldots, v_n\}$, and sets of new atoms $V_i = \{v_{i,1}, \ldots, v_{i,n}\}$, for $i \in \{1, \ldots, n\}$, $U_{P,V}$ is defined as follows:*

$$
\begin{array}{lll}
(1) & S_{P,V}, & \\
(2) & v_{i,j} \leftarrow v_i, \bar{v}'_i, v'_j; \quad v_{i,i} \leftarrow v_i, \bar{v}'_i, & 1 \leq i, j \leq n, \\
(3) & \leftarrow v_i, \bar{v}'_i, v_j, \bar{v}'_j, not\, v_{i,j}; \quad \leftarrow v_i, \bar{v}'_i, \bar{v}_j, v_{i,j}, & 1 \leq i, j \leq n, \\
(4) & H(r_i) \leftarrow v_i, \bar{v}'_i, B^+(r_i), not\, B^-(r), & r \in P, 1 \leq i \leq n,
\end{array}
$$

where rule r_i is the result of replacing in r each atom v_j by $v_{i,j}$.

Intuitively, $U_{P,V}$ works as follows. First, $S_{P,V}$ yields all SE-models (X, Y) of P. Then, (2)–(4) check if no Z with $X \subset Z \subset Y$ satisfies $Z \models P^Y$. This is done as follows: For each $v_i \in Y \setminus X$, (2) initializes sets $X_i = X \cup \{v_i\}$ via atoms V_i. Then, (4) yields, for each $v_i \in Y \setminus X$, minimal sets $Z_i \supseteq X_i$ satisfying $Z_i \models P^Y$. Finally, constraints (3) apply for each such $Z_i \neq Y$. Hence, we get a stable model iff, for each i such that $v_i \in Y \setminus X$, the minimal set Z_i matches Y. But then, $(X, Y) \in M_u(P)$. Formally, we have the following result:

Lemma 1. *Let P be a DLP over atoms $V = \{v_1, \ldots, v_n\}$. Then, $\mathcal{SM}(U_{P,V})$ is given by $\{X' \cup (\bar{V}' \setminus \bar{X}') \cup Y \cup (\bar{V} \setminus \bar{Y}) \cup J_{X,Y} \mid X, Y \subseteq V, (X, Y) \in M_u(P)\}$, where $J_{X,Y} = \{v_{i,j} \mid v_i \in Y \setminus X, v_j \in Y\}$.*

Theorem 4. *Let P,Q be DLPs, V the set of atoms in $P \cup Q$, and u, w, s new atoms. Then, $P \models_u Q$ iff the program $C^U_{P,Q}$, given as follows, has no stable model:*

$$
\begin{array}{lll}
(1) & U_{P,V}, & \\
(2) & u \leftarrow B^+(r), not\, B^-(r), not\, H(r), & r \in Q, \\
(3) & u \leftarrow B^+(r'), not\, B^-(r), not\, H(r'), & r \in Q, \\
(4) & v''_i \leftarrow v'_i; \quad v''_i \leftarrow v_i, not\, \bar{v}''_i; \quad \bar{v}''_i \leftarrow not\, v''_i, & 1 \leq i \leq n, \\
(5) & w \leftarrow not\, v''_i, v_i; \quad s \leftarrow v''_i, not\, v'_i, & 1 \leq i \leq n, \\
(6) & \leftarrow not\, w, not\, u; \quad \leftarrow not\, s, not\, u, & \\
(7) & \leftarrow B^+(r''), not\, B^-(r), not\, H(r''), not\, u, & r \in Q.
\end{array}
$$

The intuition behind $C^U_{P,Q}$ is as follows. (1) computes all UE-models (X, Y) of P via atoms V' and V (characterizing X' resp. Y). We must check that each of them is also a UE-model of Q, and if so, the program should have no stable model. Therefore, (2) checks $Y \models Q$ and (3) $X \models Q^Y$. If this fails, $(X, Y) \notin M_s(Q)$, and thus $(X, Y) \notin M_u(Q)$. Whenever this is detected, we derive u, which is used to disable the constraints (6)–(7). Finally, we must check whether no Z exists such that $X \subset Z \subset Y$ and $(Z, Y) \in M_s(Q)$. To this end, (4) guesses a set Z such that $X \subseteq Z \subseteq Y$, and (5) derives w if some $p \in Y \setminus Z$ exists, and s if some $q \in Z \setminus X$ exists. Using (6), we remain with those Z which are in between X and Y. Finally, (7) checks $(Z, Y) \notin M_s(Q)$.

Now let I be any stable model of $C^U_{P,Q}$. If $u \in I$, then the members of V,V' in I give an UE-model of P which is not an SE-model of Q; and if $u \notin I$, then V, V'' give an SE-model (Z, Y) of Q where $(X, Y) \in M_u(P)$ with $X \subset Z$.

Notice that both $U_{P,V}$ and $C^U_{P,Q}$ are normal (resp., HCF) if P is normal (resp., HCF). Hence, the test for $P \models_u Q$ yields a normal (resp., HCF) program if P is normal (resp., HCF), which is in coNP compared to Π^P_2-completeness in general [5]. $U_{P,V}$ is thus appealing from this point of view. However, the size of $U_{P,V}$ is quadratic in the size of P. In the extended paper, we give a linear-size program replacing $U_{P,V}$, but it is neither normal nor HCF regardless of P.

5 Global Simplifications

In this section, we consider the following issue: Given a DLP P from some class $\mathcal{C}$, under what conditions is there a program Q from a class $\mathcal{C}'$ such that $P \equiv_e Q$? This question is crucial for program transformation and simplification from a more general perspective than local transformations. We succeed here providing characterizations where $\mathcal{C}'$ is the class of positive or Horn programs.

Theorem 5. *Let P be a DLP. Then, there exists a positive program Q such that $P \equiv_e Q$ iff, for all $(X, Y) \in M_e(P)$, it holds that $X \models P$, where $e \in \{s, u\}$.*

Theorem 6. *For any DLP P, let $M^1_e(P) = \{X \mid (X, Y) \in M_e(P)\}$, for $e \in \{s, u\}$. Then, there exists some Horn program Q such that $P \equiv_e Q$ iff (i) $M^1_e(P) \models P$, and (ii) $M^1_e(P)$ is closed under intersection.*

Note that Theorem 5 implies that if P is e-equivalent to some positive program Q, then we can obtain Q by shifting *not*-literals to the head. We can exploit this observation for a simple test. Let, for any rule r, r^{ls} be the rule such that $H(r^{ls}) = H(r) \cup B^-(r)$, and $B(r^{ls}) = B^+(r)$ (i.e., the "left shift" of r).

Theorem 7. *Let P be any DLP and $e \in \{s, u\}$. Then, there exists a positive program Q such that $P \equiv_e Q$ iff $P \models_e r^{ls}$, for every $r \in P$ such that $B^-(r) \neq \emptyset$.*

To test for the existence of an e-equivalent Horn program Q, we just need to add a check whether the models of P are intersection closed (which is in coNP).

In the case of $\models_s$, we can even get rid of SE-consequence and come up with a test using $\models$ only. In the following, we characterize strong equivalence of any program P,

Table 3. Complexity of deciding rule and literal redundancy.

$\equiv_s$ / $\equiv_u$	Horn LP	Normal LP	Positive LP	HCF LP	DLP
$r \in P$ redundant?	P	coNP	coNP	coNP	coNP/Π_2^P
ℓ in $r \in P$ redundant?	P	coNP	coNP	coNP	coNP/Π_2^P

possessing a classical model, to some positive program Q in terms of $\models$; if P has no classical model, then $P \equiv_s \{\bot\}$. Note that $\{\bot\}$ is positive.

Recall that $(X, Y) \in M_e$ requires $X \subseteq Y$, but the reduct P^Y of program P may have models X' such that $X' \not\subseteq Y$. To select exactly those X' such that $X' \subseteq Y$, let, for any DLP P and interpretation I, $P^I_{\leq}$ be the positive logic program $P^I \cup \{\leftarrow a \mid I \not\models a\}$.

Proposition 7. *Let P be a DLP having a classical model, and let r be a positive rule. Then, $P \models_s r$ iff $P^M_{\leq} \models r$, for every maximal model M of P.*

Note that the proposition fails for non-positive r. Indeed, consider the program $P = \{a \leftarrow b;\ a \leftarrow \mathit{not}\, b\}$ and $r = a \leftarrow \mathit{not}\, b$. The unique maximal model of P is $At = \{a, b\}$, and $P^{At}_{\leq} = \{a \leftarrow b\}$, but $P^{At}_{\leq} \not\models a \leftarrow \mathit{not}\, b$.

Theorem 8. *Let P, Q be DLPs, where P is positive having a classical model. Then, $P \equiv_s Q$ iff (i) $P \models Q$, and (ii) $Q^M_{\leq} \models P$, for every maximal model M of P.*

We finally arrive at the desired characterization of semantic equivalence to some positive program, by exploiting that P^+ is a singular candidate for Q.

Theorem 9. *Let P be any DLP without positive constraints. Then, there exists some positive program Q such that $P \equiv_s Q$ iff $P^+ \models P \setminus P^+$.*

For strong equivalence of P to some Horn theory Q, we have in the right-hand side the additional condition that the models of P^+ are closed under intersection.

6 Complexity

In this section, we address complexity issues related to the problems above. Recall that we deal with propositional programs only.

The complexity of deciding redundant literals and rules is given in Table 3, for various classes of programs, where entries represent completeness for the respective complexity classes. The upper complexity bounds are easily obtained from the complexity of deciding $P \models_s r$ resp. $P \models_u r$ for the respective program class $\mathcal{C}$ in Table 3 (cf. [5]), since $P \in \mathcal{C}$ implies $P \setminus \{r\} \in \mathcal{C}$ in each case. The hardness parts for positive programs, HCF programs, and DLPs under $\equiv_s$ are easily obtained from the coNP-completeness of deciding $P \models v$ for an atom v and a positive HCF P. For Horn programs, the problem is polynomial since $P \models_s r$ and $P \models_u r$ coincide with $P \models r$, which is polynomial for Horn P. The Π_2^P-hardness of DLPs under $\equiv_u$ can be shown by suitable adaptions of proofs in [5]. However, redundancy checking is tractable in certain cases:

Table 4. Complexity of deciding if for $P \in \mathcal{C}$ (row) some $Q \in \mathcal{C}'$ (column) exists with $P \equiv_e Q$.

$\equiv_s$ / $\equiv_u$	Positive LP	Horn LP	constraint-free positive LP	Definite Horn
Normal LP	coNP	coNP	P/coNP	P/coNP
HCF LP	coNP	coNP	coNP	coNP
Disjunctive LP	coNP/Π_2^P	coNP/in Π_2^P	coNP/Π_2^P	coNP/in Π_2^P

Theorem 10. *Let P be a DLP without positive constraints such that P^+ is normal. Then, checking redundancy of a positive rule $r \in P$ under $\equiv_s$ is decidable in polynomial time, and likewise checking redundancy of a literal ℓ in a positive rule $r \in P$.*

We next consider the complexity of deciding whether a given program is equivalent to some program belonging to a specific syntactic subclass of DLPs. The generic problem is as follows:

Instance: A program P (possibly from a restricted class of programs, $\mathcal{C}$).
Question: Does there exist some program Q from a fixed class of programs, $\mathcal{C}'$, such that $P \equiv_e Q$, where $e \in \{s, u\}$ is fixed?

Our results concerning this task are summarized in Table 4, where entries stand for completeness for the respective complexity classes unless stated otherwise. Notice that definite Horn programs are constraint-free Horn programs.

Unsurprisingly, the problem is intractable in general. The upper bounds for the non-polynomial cases can be easily derived from the results in the previous sections and the complexity of $\models_s$, which is coNP-complete for general DLPs (as follows from [22, 23]), and of $\models_u$, which is Π_2^P-complete in general but coNP-complete for HCF (and, in particular, normal) logic programs.

Note that the rows for NLPs and HCF programs coincide under $\equiv_u$, since $P \equiv_u P^{\rightarrow}$ holds for each HCF program P, and the shifting program $P^{\rightarrow}$ is constraint-free iff P is constraint-free. However, under $\equiv_s$, we have a different picture, and we can find some interesting tractable cases from Theorem 9:

Theorem 11. *Let P be a positive-constraint free DLP such that P^+ is normal. Then, deciding if there is some positive program Q such that $P \equiv_s Q$ is feasible in polynomial time, and likewise deciding whether there is some Horn program Q with $P \equiv_s Q$.*

7 Conclusion

We have pushed further the work on nonmonotonic logic-program simplification under the notions of strong and uniform equivalence (cf. [11, 22, 23, 20, 5, 16]) by providing syntactic and semantic rules for program transformation and giving characterizations of programs which are semantically equivalent to positive and Horn programs. Similar characterizations for equivalence to *normal* programs are considered in a companion paper [6]. Future work concerns implementing the methods presented, as well as extending the results to programs with two kinds of negation and to the case of programs with variables. The current results provide a solid basis, though, for lifting them using common techniques.

References

1. R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12:53–87, 1994.
2. S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 40(1):1–46, 1999.
3. D. J. de Jongh and L. Hendriks. Characterizations of Strongly Equivalent Logic Programs in Intermediate Logics. *Theory and Practice of Logic Programming*, 3(3):259–270, 2003.
4. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving Using the DLV System. In *Logic-Based Artificial Intelligence*, pp. 79–103. Kluwer, 2000.
5. T. Eiter and M. Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proc. ICLP-03*. To appear.
6. T. Eiter, M. Fink, H. Tompits, and S. Woltran. Eliminating Disjunction from Propositional Logic Programs under Stable Model Preservation. In *Proc. ASP-03*. Available at: http://CEUR-WS.org/Vol-78/, 2003.
7. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22:364–418, 1997.
8. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
9. T. Janhunen and E. Oikarinen. Testing the Equivalence of Logic Programs under Stable Model Semantics. In *Proc. JELIA-02*, LNCS 2424, pp. 493–504. Springer, 2002.
10. T. Janhunen and E. Oikarinen. LPEQ and DLPEQ – Translators for Automated Equivalence Testing of Logic Programs. In *Proc. LPNMR-03*. To appear.
11. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
12. F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. KR-02*, pp. 170–176. Morgan Kaufmann, 2002.
13. F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. AAAI-2002*, pp. 112–117. AAAI Press / MIT Press, 2002.
14. M. J. Maher. Equivalences of Logic Programs. In Minker [15], pp. 627–658.
15. J. Minker, editor. *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufman, Washington DC, 1988.
16. M. Osorio, J. Navarro, and J. Arrazola. Equivalence in Answer Set Programming. In *Proc. LOPSTR 2001*, LNCS 2372, pp. 57–75. Springer, 2001.
17. D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA 2001*, LNCS 2258, pp. 306–320. Springer, 2001.
18. D. Pearce and A. Valverde. Some Types of Equivalence for Logic Programs and Equilibrium Logic. In *Proc. APPIA-GULP-PRODE 2003*, 2003.
19. T. Przymusinski. Stable Semantics for Disjunctive Programs. *New Generation Computing*, 9:401–424, 1991.
20. Y. Sagiv. Optimizing Datalog Programs. In Minker [15], pp. 659–698.
21. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, 2002.
22. H. Turner. Strong Equivalence for Logic Programs and Default Theories (Made Easy). In *Proc. LPNMR-01*, LNCS 2173, pp. 81–92. Springer, 2001.
23. H. Turner. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *Theory and Practice of Logic Programming*, 3(4–5):609–622, 2003.
24. K. Wang and L. Zhou. Comparisons and Computation of Well-founded Semantics for Disjunctive Logic Programs. *ACM Transactions on Computational Logic*. To appear. Available at: http://arxiv.org/abs/cs.AI/0301010, 2003.