Hyperequivalence of Logic Programs with Respect to Supported Models

Mirosław Truszczyński
Department of Computer Science
University of Kentucky,
Lexington, KY 40506-0046, USA
mirek@cs.uky.edu

Stefan Woltran
Institut für Informationssysteme 184/2
Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
woltran@dbai.tuwien.ac.at

Abstract

Recent research in nonmonotonic logic programming has focused on program equivalence relevant for program optimization and modular programming. So far, most results concern the stable-model semantics. However, other semantics for logic programs are also of interest, especially the semantics of supported models which, when properly generalized, is closely related to the autoepistemic logic of Moore. In this paper, we consider a framework of equivalence notions for logic programs under the supported (minimal) model-semantics and provide characterizations for this framework in model-theoretic terms. We use these characterizations to derive complexity results concerning testing hyperequivalence of logic programs wrt supported (minimal) models.

Introduction

The problem of the equivalence of logic programs wrt the stable-model semantics has received substantial attention in the answer-set programming research community in the past several years (Lifschitz, Pearce, & Valverde 2001; Lin 2002; Turner 2003; Inoue & Sakama 2004; Eiter, Tompits, & Woltran 2005; Eiter, Fink, & Woltran 2007; Oikarinen & Janhunen 2006; Oetsch, Tompits, & Woltran 2007; Woltran 2008). The problem can be stated as follows. Given a class $\mathcal{C}$ of logic programs (we will refer to them as *contexts*), we say that programs P and Q are *equivalent relative to* $\mathcal{C}$ if for every program $R \in \mathcal{C}$, $P \cup R$ and $Q \cup R$ have the same *stable models*. Clearly, for every class $\mathcal{C}$, the equivalence relative to $\mathcal{C}$ implies the standard nonmonotonic equivalence of programs, where two programs P and Q are *nonmonotonically equivalent* if they have the same stable models. Therefore, we will refer to these stronger versions of equivalence collectively as *hyperequivalence*.

Understanding hyperequivalence is fundamental for the development of modular answer-set programs and knowledge bases. The problem is non-trivial due to the nonmonotonic nature of the stable-model semantics. If S is a module within a larger program T, replacing S with S' results in the program $T' = (T \setminus S) \cup S'$, which must have the same meaning (the same stable models) as T. The nonmonotonic equivalence of S and S' does not guarantee it. The hyperequivalence relative to the class of all programs does. However, the latter may be a too restrictive approach in certain application scenarios, in particular if properties of possible realizations for T are known in advance.

Thus, several interesting notions of hyperequivalence, imposing restrictions on the context class $\mathcal{C}$, have been studied. If $\mathcal{C}$ is unrestricted, that is, any program is a possible context, we obtain *strong* equivalence (Lifschitz, Pearce, & Valverde 2001). If $\mathcal{C}$ is the collection of all sets of facts, we obtain *uniform* equivalence (Eiter, Fink, & Woltran 2007). Another direction is to restrict the alphabet over which contexts are given. The resulting notions of hyperequivalence are called *relativized* (wrt the context alphabet), and can be combined with strong and uniform equivalence (Eiter, Fink, & Woltran 2007). Even more generally, we can specify different alphabets for bodies and heads of rules in contexts. This gives rise to a common view on strong and uniform equivalence (Woltran 2008). A yet different approach to hyperequivalence is to compare only some dedicated projected output atoms rather than entire stable models (Eiter, Tompits, & Woltran 2005; Oikarinen & Janhunen 2006; Oetsch, Tompits, & Woltran 2007).

All those results concern the stable-model semantics[1]. In this paper, we address the problem of the hyperequivalence of programs wrt the other major semantics, that of supported models (Clark 1978). We define several concepts of hyperequivalence, depending on the class of programs allowed as contexts. We obtain characterizations of hyperequivalence wrt supported models in terms of semantic objects, similar to se-models (Turner 2003) or ue-models (Eiter, Fink, & Woltran 2007), that one can attribute to programs.

Since the minimality property is fundamental from the perspective of knowledge representation, we also consider in the paper the semantics of supported models that are minimal (as models). While it seems to have received little attention in the area of logic programming, it has been studied extensively in a more general setting of modal nonmonotonic logics, first under the name of the semantics of *moderately grounded expansions* for autoepistemic logic (Konolige 1988) and then, under the name of *ground* $\mathcal{S}$-expansions, for an arbitrary nonmonotonic modal logic $\mathcal{S}$ (Kaminski 1991; Truszczyński 1991). The complexity of

[1]There is little work on other semantics, with (Cabalar *et al.* 2006) being a notable exception.

reasoning with moderately grounded expansion was established in (Eiter & Gottlob 1992) to be complete for classes at the third level of the polynomial hierarchy.

Here, we study this semantics in the form tailored to logic programming. By refining techniques we develop for the case of supported models, we characterize the concept of hyperequivalence wrt supported minimal models relative to several classes of contexts.

The characterizations allow us to derive results on the complexity of problems to decide whether two programs are hyperequivalent wrt supported (minimal) models. They are especially useful in establishing upper bounds which, typically, are easy to derive but in the context of hyperequivalence are not obvious. Our results paint a detailed picture of the complexity landscape for relativized hyperequivalence wrt supported (minimal) models. For proofs and additional discussion we refer to (Truszczyński & Woltran 2008).

Preliminaries

We fix a countable set At of atoms (possibly infinite). All programs we consider here consist of *rules* of the form

$$a_1|\ldots|a_k \leftarrow b_1,\ldots,b_m, not\, c_1,\ldots,not\, c_n,$$

where a_i, b_i and c_i are atoms in At, '|' stands for the disjunction, ',' stands for the conjunction, and not is the *default* negation. If $k = 0$, the rule is a *constraint*. If $k \leq 1$, the rule is *normal*.

For a rule r of the form given above, we call the set $\{a_1,\ldots,a_k\}$ the head of r and denote it by $hd(r)$. Similarly, we call the conjunction $b_1,\ldots,b_m, not\, c_1,\ldots,not\, c_n$ the body of r and denote it by $bd(r)$. We will also use $bd^{\pm}(r)$ to denote the set $\{b_1,\ldots,b_m,c_1,\ldots,c_n\}$ of atoms in the body of r.

An interpretation $M \subseteq At$ is a *model* of a rule r, written $M \models r$, if whenever M satisfies every literal in $bd(r)$, written $M \models bd(r)$, we have that $hd(r) \cap M \neq \emptyset$, written $M \models hd(r)$.

An interpretation $M \subseteq At$ is a *model* of a program P, written $M \models P$, if $M \models r$ for every $r \in P$. If, in addition, M is a minimal hitting set of $\{hd(r) \mid\ r \in P \text{ and } M \models bd(r)\}$, then M is a *supported* model of P (Brass & Dix 1997; Inoue & Sakama 1998).

For a rule $r = a_1|\ldots|a_k \leftarrow bd$, where $k \geq 1$, a *shift* of r is a normal program rule of the form

$$a_i \leftarrow bd, not\, a_1,\ldots,not\, a_{i-1}, not\, a_{i+1},\ldots,not\, a_k,$$

where $i = 1,\ldots,k$. If r is a constraint, the only *shift* of r is r itself. A program consisting of all shifts of rules in a program P is the *shift* of P. We denote it by $sh(P)$. It is evident that a set Y of atoms is a (minimal) model of P if and only if Y is a (minimal) model of $sh(P)$. It is easy to check that Y is a supported model of P if and only if it is a supported model of $sh(P)$.

Supported models of a *normal* logic program P have a useful characterization in terms of the (partial) one-step provability operator T_P (van Emden & Kowalski 1976), defined as follows. For $M \subseteq At$, if there is a constraint $r \in P$ such that $M \models bd(r)$ (that is, $M \not\models r$), then $T_P(M)$ is undefined. Otherwise,

$$T_P(M) = \{hd(r) \mid\ r \in P \text{ and } M \models bd(r)\}.$$

Whenever we use $T_P(M)$ in a relation such as (proper) inclusion, equality or inequality, we always implicitly assume that $T_P(M)$ is defined.

It is well known that M is a model of P if and only if $T_P(M) \subseteq M$ (that is, T_P is defined for M and satisfies $T_P(M) \subseteq M$). Similarly, M is a *supported* model of P if $T_P(M) = M$ (that is, T_P is defined for M and satisfies $T_P(M) = M$) (Apt 1990).

It follows that M is a model of a disjunctive program P if and only if $T_{sh(P)}(M) \subseteq M$. Moreover, M is a supported model of P if and only if $T_{sh(P)}(M) = M$.

Hyperequivalence with Respect to Supported Models

Disjunctive programs P and Q are *supp-equivalent* relative to a class $\mathcal{C}$ of disjunctive programs if for every $R \in \mathcal{C}$, $P \cup R$ and $Q \cup R$ have the same supported models.

Supp-equivalence is a non-trivial concept, different than equivalence wrt models, stable models, and hyperequivalence wrt stable models.

Example 1 *Let $P_1 = \{a \leftarrow a\}$ and $Q_1 = \emptyset$. Clearly, P_1 and Q_1 have the same models and the same stable models. Moreover, for every program R, $P_1 \cup R$ and $Q_1 \cup R$ have the same stable models, that is, P_1 and Q_1 are strongly (and so, also uniformly) equivalent wrt stable models. However, P_1 and Q_1 have different supported models. Thus, they are not supp-equivalent relative to* any *class of programs.*

Next, let $P_2 = \{a \leftarrow a; a \leftarrow not\, a\}$ and $Q_2 = \{a\}$. One can check that for every program R, $P_2 \cup R$ and $Q_2 \cup R$ have the same supported models, that is, P_2 and Q_2 are supp-equivalent relative to any *class of programs. They are also equivalent wrt classical models However, P_2 and Q_2 do not have the same stable models and so, they are not equivalent wrt stable models nor hyperequivalent wrt stable models relative to* any *class of programs.*

Finally, let $P_3 = \{\leftarrow b\} \cup P_2$ and $Q_3 = Q_2$. Then, P_3 and Q_3 are neither hyperequivalent wrt stable models relative to any class of programs nor equivalent wrt classical models. Still P_3 and Q_3 have the same supported models, and for any program R, such that b does not appear in rule heads of R, $P_3 \cup R$ and $Q_3 \cup R$ have the same supported models, that is, P_3 and Q_3 are supp-equivalent wrt this class of programs (we will verify this claim later). As we will see, supp-equivalence wrt all *programs implies equivalence wrt models and so, it is not a coincidence that in the last example we used a restricted class of contexts. In fact, P_3 and Q_3 are* not *supp-equivalence wrt all programs. For $R = \{b\}$, $\{a, b\}$ is a supported model of $Q_3 \cup R$, but not of $P_3 \cup R$.*

We observe that supp-equivalence relative to $\mathcal{C}$ implies supp-equivalence relative to any $\mathcal{C}'$, such that $\mathcal{C}' \subseteq \mathcal{C}$ (in particular, for $\mathcal{C}' = \{\emptyset\}$, this implies standard equivalence wrt supported models), but the converse is not true in general as illustrated by programs P_3 and Q_3.

In this section we characterize supp-equivalence relative to classes of programs defined in terms of atoms that can appear in the heads and in the bodies of rules. Let $A, B \subseteq At$. By $\mathcal{HB}^d(A, B)$ we denote the class of all disjunctive programs P such that $hd(P) \subseteq A$ (atoms in the heads of rules in P must be from A) and $bd^{\pm}(P) \subseteq B$ (atoms in the bodies of rules in P must be from B). We denote by $\mathcal{HB}^n(A, B)$ the class of all normal programs in $\mathcal{HB}^d(A, B)$ (possibly with constraints). These classes of programs were considered in the context of hyperequivalence of programs wrt the stable-model semantics in (Woltran 2008).

We start with an observation implied by the fact that models and supported models are preserved under shifting.

Theorem 2 *Let P and Q be disjunctive logic programs, and let $A, B \subseteq At$. The following conditions are equivalent*

1. *P and Q are supp-equivalent relative to $\mathcal{HB}^d(A, B)$*
2. *P and Q are supp-equivalent relative to $\mathcal{HB}^n(A, B)$*
3. *$sh(P)$ and $sh(Q)$ are supp-equivalent relative to $\mathcal{HB}^n(A, B)$*

Theorem 2 allows us to focus on normal programs and normal contexts and, then, obtain characterizations of the general disjunctive case as corollaries. It is important, as in the normal program case, we can take advantage of the one-step provability operator.

Given a normal program P, and a set $A \subseteq At$, we define

$$Mod_A(P) = \{Y \subseteq At \mid\ Y \models P \text{ and } Y \setminus T_P(Y) \subseteq A\}.$$

We have the following characterization of the supp-equivalence relative to $\mathcal{HB}^n(A, B)$.

Theorem 3 *Let P and Q be normal programs, $A \subseteq At$, and $\mathcal{C}$ a class of programs such that $\mathcal{HB}^n(A, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}^n(A, At)$. Then, P and Q are supp-equivalent relative to $\mathcal{C}$ if and only if $Mod_A(P) = Mod_A(Q)$ and for every $Y \in Mod_A(P)$, $T_P(Y) = T_Q(Y)$.*

Theorem 3 has several corollaries. The first one deals with the case when $A = At$, in which the characterizing condition simplifies.

Corollary 4 *Let P and Q be normal programs and $\mathcal{C}$ a class of programs such that $\mathcal{HB}^n(At, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}^n(At, At)$. Then, P and Q are supp-equivalent relative to $\mathcal{C}$ if and only if P and Q have the same models and for every model Y of P, $T_P(Y) = T_Q(Y)$.*

This result applies, in particular, to $\mathcal{C} = \mathcal{HB}^n(At, \emptyset)$ and $\mathcal{C} = \mathcal{HB}^n(At, At)$ and, consequently, characterizes strong and uniform supp-equivalence of normal programs.

Next, for the case $A = \emptyset$ we have the following result.

Corollary 5 *Let P and Q be normal programs and $\mathcal{C}$ a class of programs such that $\mathcal{HB}^n(\emptyset, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}^n(\emptyset, At)$. The following conditions are equivalent:*

1. *P and Q are supp-equivalent relative to $\mathcal{C}$*
2. *P and Q have the same supported models*
3. *$Mod_\emptyset(P) = Mod_\emptyset(Q)$*

We will now apply our results to some pairs of programs discussed in Example 1.

Example 6 *First, we note that P_1 and Q_1 have the same models. In particular, $\{a\}$ is a model of both programs. However, $T_{P_1}(\{a\}) = \{a\}$ and $T_{Q_1}(\{a\}) = \emptyset$. Thus, $T_{P_1}(\{a\}) \neq T_{Q_1}(\{a\})$ and so, P_1 and Q_1 are not supp-equivalent relative to $\mathcal{HB}^n(At, \emptyset)$ (by Corollary 4).*

On the other hand, P_2 and Q_2 have the same models and for every Y (in particular, for every model *Y of P_2 and Q_2), $T_{P_2}(Y) = \{a\} = T_{Q_2}(Y)$. Thus, P_2 and Q_2 are supp-equivalent relative to $\mathcal{HB}^n(At, At)$.*

Finally, $Y \in Mod_{At\setminus\{b\}}(P_3)$ if and only if $Y \models P_3$ and $Y \setminus T_{P_3}(Y) \subseteq At \setminus \{b\}$. Since the former implies the latter, $Y \in Mod_{At\setminus\{b\}}(P_3)$ if and only if $a \in Y$ and $b \notin Y$. One can check that this condition also characterizes $Y \in Mod_{At\setminus\{b\}}(Q_3)$. Thus, $Mod_{At\setminus\{b\}}(P_3) = Mod_{At\setminus\{b\}}(Q_3)$. Moreover, if $Y \in Mod_{At\setminus\{b\}}(P_3)$ ($a \in Y$ and $b \notin Y$), $T_{P_3}(Y) = \{a\} = T_{Q_3}(Y)$. Consequently, P_3 and Q_3 are supp-equivalent relative to $\mathcal{HB}^n(At \setminus \{b\}, At)$.

The last corollary concerns the disjunctive case and depends also on Theorem 2.

Corollary 7 *Let P and Q be disjunctive programs, $A \subseteq At$, and $\mathcal{C}$ a class of programs such that $\mathcal{HB}^n(A, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}^d(A, At)$. Then, P and Q are supp-equivalent relative to $\mathcal{C}$ if and only if $Mod_A(sh(P)) = Mod_A(sh(Q))$ and for every $Y \in Mod_A(sh(P))$, $T_{sh(P)}(Y) = T_{sh(Q)}(Y)$.*

Corollary 7 applies, in particular, to the cases when $\mathcal{C}$ is any of the following classes: $\mathcal{HB}^d(A, At)$, $\mathcal{HB}^n(A, At)$, $\mathcal{HB}^d(A, \emptyset)$, and $\mathcal{HB}^n(A, \emptyset)$. It also implies that the alphabet allowed for the bodies of context programs plays no role in the case of supp-equivalence, unlike in the case of hyperequivalence wrt stable models (Woltran 2008). In particular, for the semantics of supported models, there is no difference between strong and uniform equivalence.

Finally, we note that Theorem 3 also implies characterizations of uniform hyperequivalence wrt stable models for *tight* logic programs, as for such programs stable and supported models coincide. The approach yields an alternative to a characterization given in (Gebser *et al.* 2008).

Hyperequivalence with Respect to Supported Minimal Models

A set M of atoms is a *supported minimal model* (*suppmin* model, for short) of a logic program P if it is a supported model of P and a minimal model of P.

Disjunctive programs P and Q are *suppmin-equivalent* relative to a class $\mathcal{C}$ of disjunctive programs if for every $R \in \mathcal{C}$, $P \cup R$ and $Q \cup R$ have the same suppmin models. Suppmin-equivalence is a different concept than other types of equivalence we considered.

Example 8 *The programs P_2 and Q_2 from Example 1 are suppmin-equivalent wrt any class of programs, as for every program R, programs $P_2 \cup R$ and $Q_2 \cup R$ have the same models and the same supported models. However, as we pointed out earlier, they are not equivalent wrt stable models nor hyperequivalent wrt stable models relative to* any *class of programs.*

Then $P_4 = P_2$ and $Q_4 = \{a \leftarrow not\, a\}$ have the same models, stable models, and are hyperequivalent wrt stable models relative to an arbitrary class of programs. However, P_4 and Q_4 are not suppmin-equivalent (they have different suppmin models).

Next, one can show that for every set U of atoms, *programs $P_1 \cup U$ and $Q_1 \cup U$ have the same suppmin models, but the programs themselves have different supported models. Thus, P_1 and Q_1 are suppmin-equivalent relative to the class of all programs consisting of atoms ($\mathcal{HB}^n(At, \emptyset)$) but they are not supp-equivalent relative to the same class. We note that P_1 and Q_1 are* not *suppmin-equivalent relative to $\mathcal{HB}^n(At, At)$, as witnessed by the context $R = \{\leftarrow not\, a\}$.*

Finally, $P_5 = \{a \leftarrow b;\ b \leftarrow b;\ \leftarrow not\, a, not\, b\}$ and $Q_5 = \{a \leftarrow b;\ b \leftarrow a;\ \leftarrow not\, a, not\, b\}$ have the same supported models but different suppmin models ($\{a, b\}$ is the only supported model of P_5 and Q_5, and a suppmin model for Q_5 but not for P_5). Thus, the programs are supp-equivalent relative to $\mathcal{HB}^n(\emptyset, \emptyset)$ (which contains the empty program only) but not suppmin-equivalent wrt that class.

Our examples distinguishing between supp- and suppmin-equivalence refer to restricted classes of contexts. As we show later, it is not coincidental. The two types of equivalence are the same if all programs are allowed as contexts.

A refinement of the method used in the previous section provides a characterization of suppmin-equivalence relative to contexts from $\mathcal{HB}^n(A, B)$ and $\mathcal{HB}^d(A, B)$, where $A, B \subseteq At$. Compared to supp-equivalence the second alphabet, B, has now to be taken into consideration!

As before, since models, minimal models and supported models are preserved under shifting, it suffices to focus on the case of normal programs.

Theorem 9 *Let P and Q be disjunctive programs, and $A, B \subseteq At$. The following conditions are equivalent*

1. *P and Q are suppmin-equivalent relative to $\mathcal{HB}^d(A, B)$*
2. *P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, B)$*
3. *$sh(P)$ and $sh(Q)$ are suppmin-equivalent relative to $\mathcal{HB}^n(A, B)$*

To characterize suppmin-equivalence for *normal* programs P and Q, we define $Mod_A^B(P)$ to be the set of all pairs (X, Y) such that

1. $Y \in Mod_A(P)$
2. $X \subseteq Y|_{A \cup B}$
3. for each $Z \subset Y$ such that $Z|_{A \cup B} = Y|_{A \cup B}$, $Z \not\models P$
4. for each $Z \subset Y$ such that $Z|_B = X|_B$ and $Z|_A \supseteq X|_A$, $Z \not\models P$
5. if $X|_B = Y|_B$, then $Y \setminus T_P(Y) \subseteq X$

Suppmin-equivalence of normal logic programs P and Q depends on sets $Mod_A^B(P)$ and $Mod_A^B(Q)$ as follows.

Theorem 10 *Let $A, B \subseteq At$ and let P, Q be normal programs. The following conditions are equivalent*

1. *P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, B)$*
2. *$Mod_A^B(P) = Mod_A^B(Q)$ and for every $(X, Y) \in Mod_A^B(P)$, $T_P(Y)|_B = T_Q(Y)|_B$*
3. *$Mod_A^B(P) = Mod_A^B(Q)$ and for every $(X, Y) \in Mod_A^B(P)$, $T_P(Y) \setminus (A \setminus B) = T_Q(Y) \setminus (A \setminus B)$*

We have several corollaries for some special choices of A and B. The first one concerns the case when $B = \emptyset$, that is, the case of relativized uniform suppmin-equivalence. Since the condition $T_P(Y)|_B = T_Q(Y)|_B$ is now trivially satisfied, Theorem 10 implies the following result.

Corollary 11 *Let $A \subseteq At$. Normal programs P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, \emptyset)$ if and only if $Mod_A^\emptyset(P) = Mod_A^\emptyset(Q)$.*

Moreover, the description of $Mod_A^B(P)$, when $B = \emptyset$ simplifies. In fact, $(X, Y) \in Mod_A^\emptyset(P)$ if and only if

1. $Y \in Mod_A(P)$
2. $X \subseteq Y|_A$
3. for each Z with $X \subseteq Z \subset Y$, $Z \not\models P$
4. $Y \setminus T_P(Y) \subseteq X$

When $A = B = At$ (strong suppmin-equivalence), it turns out that supp-equivalence and suppmin-equivalence coincide (cf. comments at the end of Example 8).

Corollary 12 *Normal programs P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(At, At)$ if and only if P and Q are supp-equivalent relative to $\mathcal{HB}^n(At, At)$.*

We will now use our results to resolve the issue of suppmin-equivalence of programs discussed earlier.

Example 13 *If P is a program such that every set of atoms is a model of P, then $Mod_{At}^\emptyset(P) = \{(Y, Y) \mid Y \subseteq At\}$. This observation applies both to P_1 and Q_1. Thus, by Corollary 11, P_1 and Q_1 are suppmin-equivalent relative to $\mathcal{HB}^n(At, \emptyset)$. We note that P_1 and Q_1 are not suppmin-equivalent relative to $\mathcal{HB}^n(At, At)$. Indeed, they are not supp-equivalent (cf. Example 6) and so, not suppmin-equivalent (by Corollary 12).*

Next, we consider programs P_5 and Q_5. We note that for every program P, $Mod_\emptyset^\emptyset(P)$ consists of pairs $(\emptyset, Y)$, where Y is a suppmin model of P. Thus, $Mod_\emptyset^\emptyset(P_5) = \emptyset$ and $Mod_\emptyset^\emptyset(Q_5) = \{(\emptyset, \{a, b\})\}$. By Corollary 11, P_5 and Q_5 are not suppmin-equivalent relative to $\mathcal{HB}^n(\emptyset, \emptyset)$.

Thanks to Theorem 9, all results concerning normal programs lift to the disjunctive case. To illustrate it, we give two such results below.

Corollary 14 *Let $A, B \subseteq At$. Disjunctive programs P and Q are suppmin-equivalent relative to $\mathcal{HB}^d(A, B)$ if and only if $Mod_A^B(sh(P)) = Mod_A^B(sh(Q))$ and for every $(X, Y) \in Mod_A^B(sh(P))$, $T_{sh(P)}(Y)|_B = T_{sh(Q)}(Y)|_B$.*

Corollary 15 *Disjunctive programs P and Q are suppmin-equivalent relative to $\mathcal{HB}^d(At, At)$ if and only if P and Q are supp-equivalent relative to $\mathcal{HB}^d(At, At)$.*

Complexity of Supp-Equivalence

We focus entirely on the case of normal programs and normal contexts. As we noted, it is not an essential restriction, and all results we obtain hold without it. We will study deciding hyperequivalence relative to classes $\mathcal{HB}^n(A,B)$. Specifically, we will consider the following problems:

SUPP: given programs P, Q (over At) and $A, B \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A,B)$

SUPP$_A$: given programs P, Q (over At) and $B \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A,B)$

SUPP^B: given programs P, Q (over At) and $A \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A,B)$

SUPP^B_A: given programs P, Q (over At), decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A,B)$.

We emphasize the changing roles of the sets A and B. In some cases, they are used to specify a problem (A in SUPP^B_A and SUPP$_A$); in others, they belong to the specification of an instance (A in SUPP^B and SUPP). In the first role, they can be finite or infinite. For instance, SUPP$_{At}$ denotes the problem to decide, given programs P, Q (over At) and $B \subseteq At$, whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(At,B)$. In the second role, they need to have finite representations.

To establish the complexity of a problem, we derive an upper and a lower bound (membership and hardness). We start by pointing out that establishing an upper bound is not entirely straightforward. A natural witness against supp-equivalence is a pair (R,Y), where R is a finite program in $\mathcal{C}$ and Y is finite set of atoms such that Y is a supported model of exactly one of $P \cup R$ and $Q \cup R$. The problem is that the size of such a program R might not be bounded by a polynomial in the size of P, Q, and possibly also A and B, depending on the problem. Thus, the most direct attempt to prove the membership of the problem in the class coNP fails. The bound can, however, be derived from our characterization theorem for several classes of context programs.

Theorem 16 *The following problems are in the class coNP: (1)* SUPP*; (2)* SUPP$_A$*, for every finite $A \subseteq At$; (3)* SUPP^B_A*, for every finite $A \subseteq At$, and for every $B \subseteq At$; (4)* SUPP^B*, for every $B \subseteq At$; (5)* SUPP$^B_{At}$*, for every $B \subseteq At$.*

In problems (3) - (5) we do not need any explicit or implicit representation of B, as the supp-equivalence relative to $\mathcal{HB}^n(A,B)$ depends on A only.

We move on to the lower bound (hardness). We have the following result.

Theorem 17 *For every finite $A \subseteq At$ and $A = At$, and for every $B \subseteq At$, the problem* SUPP^B_A *is coNP-hard.*

We observe that for the result to hold we do not need to know B. Putting together Theorems 16 and 17 we obtain the following result.

Corollary 18 *The problems listed in Theorem 16 are coNP-complete.*

Problems we considered so far do not impose restrictions on input programs P and Q. In particular, they contain instances with $Mod_A(P) \neq Mod_A(Q)$. We will now consider the problem to decide whether normal programs P and Q such that $Mod_A(P) = Mod_A(Q)$ are supp-equivalent relative to $\mathcal{HB}^n(A,B)$. It turns out that this additional information is of no help as the complexity does not go down.

Theorem 19 *Let A be a fixed finite non-empty subset of At or let $A = At$. For every set $B \subseteq At$, the following problem is coNP-complete: given two normal programs P and Q such that $Mod_A(P) = Mod_A(Q)$, decide whether they are supp-equivalent relative $\mathcal{HB}^n(A,B)$.*

The requirement that $A \neq \emptyset$ is necessary for the complexity result of Theorem 19. Indeed, by Corollary 5, if $A = \emptyset$, programs P and Q with $Mod_A(P) = Mod_A(Q)$ are necessarily supp-equivalent.

Complexity of Suppmin-Equivalence

We will use here the same notational schema as in the previous section, but replace supp-equivalence with suppmin-equivalence and write SUPPMIN instead of SUPP. For instance, we write SUPPMIN^B (for B fixed and not part of the input) to denote the following problem: given normal programs P and Q, and $A \subseteq At$, decide whether P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A,B)$.

Deciding suppmin-equivalence relative to $\mathcal{HB}^n(A,B)$, where $A = At$ or $B = At$, remains in the class coNP.

Theorem 20 *The following problems are coNP-complete:* SUPPMIN$^B_{At}$*, for every finite $B \subseteq At$,* SUPPMIN$^{At}_A$*, for every finite $A \subseteq At$,* SUPPMIN^{At}*,* SUPPMIN$_{At}$*, and* SUPPMIN$^{At}_{At}$*.*

The general problem is of higher complexity.

Theorem 21 *The problem* SUPPMIN *is in Π^P_2.*

We now have the following result. The membership follows from the fact that problems SUPPMIN^B, SUPPMIN$_A$, SUPPMIN^B_A (for finite $A, B \subseteq At$) reduce to SUPPMIN. The hardness can be established by showing that already the problem SUPPMIN^B_A is Π^P_2-hard. The proof exploits the characterization given by Theorem 10, and depends on a reduction from the problem to decide whether a QBF $\forall Y \exists X \varphi$ is true, known to be Π^P_2-complete.

Theorem 22 *The following problems are Π^P_2-complete:* SUPPMIN, SUPPMIN^B, SUPPMIN$_A$, SUPPMIN^B_A*, for every finite $A, B \subseteq At$.*

As for supp-equivalence, having additional information that sets $Mod^B_A(P)$ and $Mod^B_A(Q)$ coincide does not make the problem of deciding suppmin-equivalence easier.

Theorem 23 *Let $A, B \subseteq At$ be finite and such that $A \cap B \neq \emptyset$. The following problem is Π^P_2-complete: given normal programs P, Q such that $Mod^B_A(P) = Mod^B_A(Q)$, decide whether P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A,B)$.*

This theorem cannot be extended to a wider class of finite sets A and B. Let $A \cap B = \emptyset$ and P, Q two normal programs such that $Mod^B_A(P) = Mod^B_A(Q)$. Let $(X,Y) \in Mod^B_A(P)$ and $b \in T_P(Y)|_B$. Then $b \in Y$ and $b \notin A$. Since $Y \in Mod_A(Q)$, $Y \setminus T_Q(Y) \subseteq A$. It follows that $b \in T_Q(Y)$ and, as $b \in B$, $b \in T_Q(Y)|_B$. Thus, $T_P(Y)|_B \subseteq T_Q(Y)|_B$ and, by symmetry, $T_P(Y)|_B = T_Q(Y)|_B$. Consequently, P and Q are suppmin-equivalent.

Conclusions

In this paper we extended the concept of hyperequivalence to two other major semantics of logic programs: the supported-model semantics and the suppmin-model semantics. We characterized these concepts of hyperequivalence and derived several complexity results.

There are similarities between hyperequivalence wrt supported (minimal) models and hyperequivalence wrt stable models. However, strong and uniform equivalence coincide for supp-equivalence, even for disjunctive programs, while for stable semantics strong and uniform equivalence are different, as long as negation as failure is permitted.

As concerns the complexity, the picture is uniform in the case of hyperequivalence wrt supported models — problems that arise naturally turn out to be coNP-complete. The situation is different for hyperequivalence wrt suppmin models. When at least one of A and B consists of all atoms, the corresponding problems of deciding hyperequivalence are coNP-complete. As soon as this is not the case, the complexity goes up to Π_2^P-completeness.[2] The results we presented demonstrate that with problems in which the departure from At is major: At is replaced with a finite set (either a parameter of the problem, or a part of the input). However, much less drastic change has the same effect. One can show that for every finite $A, B \subseteq At$ such that $A \neq \emptyset$, the following problem is Π_2^P-complete: given normal programs P and Q, decide whether P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(At \setminus A, B)$. Thus, even if just one atom from At is forbidden from appearing in heads of rules in context programs, the complexity jumps one level up.

Finally, we note that our characterizations and techniques behind proofs are algebraic and suggest generalizations to the language of partial operators on boolean algebras (cf. (Truszczyński 2006) for algebraic generalizations of hyperequivalence wrt stable models). Thus, it may be possible to extend the results on supp- and suppmin-equivalence to other nonmonotonic logics. One direction is to study hyperequivalence in autoepistemic logic wrt expansions and moderately grounded expansions. Indeed, autoepistemic logic with the semantics of (moderately grounded) expansions, when restricted to theories in which the modal operator is applied to atoms, can be regarded as a modal variant of logic programming with the semantics of supported (minimal) models. This is the subject of our future work.

Acknowledgments

This work was partially supported by the NSF grant IIS-0325063, the KSEF grant KSEF-1036-RDE-008, and by the Austrian Science Fund (FWF) under grant P18019.

References

Apt, K. 1990. Logic Programming. In van Leeuven, J., ed., *Handbook of Theoretical Computer Science*. Elsevier. 493–574.

Brass, S., and Dix, J. 1997. Characterizations of the Disjunctive Stable Semantics by Partial Evaluation. *J. Logic Programm.* 32(3):207–228.

Cabalar, P.; Odintsov, S. P.; Pearce, D.; and Valverde, A. 2006. Analysing and Extending Well-Founded and Partial Stable Semantics Using Partial Equilibrium Logic. In *Proc. of ICLP 2006*, *LNCS* 4079, 346–360. Springer.

Clark, K. 1978. Negation as Failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. Plenum Press. 293–322.

Eiter, T., and Gottlob, G. 1992. Reasoning with Parsimonious and Moderately Grounded Expansions. *Fund. Inform.* 17(1-2):31–53.

Eiter, T.; Fink, M.; and Woltran, S. 2007. Semantical Characterizations and Complexity of Equivalences in Answer Set Programming. *ACM TOCL* 8(3), article 17.

Eiter, T.; Tompits, H.; and Woltran, S. 2005. On Solution Correspondences in Answer-Set Programming. In *Proc. of IJCAI 2005*, 97–102. Professional Book Center.

Gebser, M.; Schaub, T.; Tompits, H.; and Woltran, S. Alternative Characterizations for Program Equivalence under Answer-Set Semantics based on Unfounded Sets. *Proc. of FoIKS 2008*, *LNCS* 4932, 24–41. Springer.

Inoue, K., and Sakama, C. 1998. Negation as Failure in the Head. *J. Logic Programm.* 35:39–78.

Inoue, K., and Sakama, C. 2004. Equivalence of Logic Programs under Updates. In *Proc. of JELIA 2004*, *LNCS* 3229, 174–186. Springer.

Kaminski, M. 1991. Embedding a Default System into Nonmonotonic Logic. *Fund. Inform.* 14(3):345–353.

Konolige, K. 1988. On the Relation between Default and Autoepistemic Logic. *Artif. Intell.* 35(3):343–382 (and errata: ibid. 41(1):115).

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

Lin, F. 2002. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. of KR 2002*, 170–176. Morgan Kaufmann.

Oetsch, J.; Tompits, H.; and Woltran, S. 2007. Facts do not Cease to Exist Because They are Ignored: Relativised Uniform Equivalence with Answer-Set Projection. In *Proc. of AAAI 2007*, 458–464. AAAI Press.

Oikarinen, E., and Janhunen, T. 2006. Modular Equivalence for Normal Logic Programs. In *Proc. of ECAI 2006*, 412–416. IOS Press.

Truszczyński, M. 1991. Modal Nonmonotonic Logic with Restricted Application of the Negation as Failure to Prove Rule. *Fund. Inform.* 14(3):355–366.

Truszczyński, M. 2006. Strong and Uniform Equivalence of Nonmonotonic Theories — an Algebraic Approach. In *Proc. of KR 2006*, 389–399. AAAI Press.

Truszczyński, M., and Woltran, S. 2008. Hyperequivalence of Logic Programs with respect to Supported Models. Technical Report DBAI-TR-2008-58, Technische Universität Wien.

Turner, H. 2003. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *TPLP* 3:609–622.

van Emden, M., and Kowalski, R. 1976. The Semantics of Predicate Logic as a Programming Language. *JACM* 23(4):733–742.

Woltran, S. 2008. A Common View on Strong, Uniform, and Other Notions of Equivalence in Answer-Set Programming. *TPLP*, 8:217–234.

[2] We recall that Π_2^P-hardness holds already for normal programs, while for stable models such problems remain in coNP (higher complexity requires disjunctive programs) (Eiter, Fink, & Woltran 2007; Woltran 2008). On the other hand uniform equivalence wrt stable models is Π_2^P-complete for disjunctive programs, while uniform suppmin-equivalence, as we proved, is in coNP.