DBAI

TECHNICAL REPORT

INSTITUT FÜR INFORMATIONSSYSTEME

ABTEILUNG DATENBANKEN UND ARTIFICIAL INTELLIGENCE

Hyperequivalence of Logic Programs with Respect to Supported Models

DBAI-TR-2008-58

Mirosław Truszczyński **Stefan Woltran**

Institut für Informationssysteme
Abteilung Datenbanken und
Artificial Intelligence
Technische Universität Wien
Favoritenstr. 9
A-1040 Vienna, Austria
Tel: +43-1-58801-18403
Fax: +43-1-58801-18492
sekret@dbai.tuwien.ac.at
www.dbai.tuwien.ac.at

DBAI TECHNICAL REPORT
2008

TECHNISCHE UNIVERSITÄT WIEN

DBAI TECHNICAL REPORT

DBAI TECHNICAL REPORT DBAI-TR-2008-58, 2008

Hyperequivalence of Logic Programs with Respect to Supported Models

Mirosław Truszczyński[1] **Stefan Woltran**[2]

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 with respect to supported (minimal) models.

[1]Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA. E-mail: mirek@cs.uky.edu

[2]Institute for Information Systems 184/2, Technische Universität Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria. E-mail: woltran@dbai.tuwien.ac.at

Acknowledgements: The authors acknowledge partial support by the NSF grant IIS-0325063, the KSEF grant KSEF-1036-RDE-008, and by the Austrian Science Fund (FWF) under grant P18019.

This is an extended version of a paper published in the Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI-08).

1 Introduction

The problem of the equivalence of logic programs with respect to the stable-model semantics has received substantial attention in the answer-set programming research community in the past several years [19, 21, 29, 5, 14, 9, 11, 7, 25, 22, 24, 31, 32, 12]. 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 of S and S' 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 [19]. If $\mathcal{C}$ is the collection of all sets of facts, we obtain *uniform* equivalence [7]. Another direction is to restrict the alphabet over which contexts are given. The resulting notions of hyperequivalence are called *relativized* (with respect to the context alphabet), and can be combined with strong and uniform equivalence [7]. 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 [31]. A yet different approach to hyperequivalence is to compare only some dedicated projected output atoms rather than entire stable models [9, 25, 24].

All those results concern the stable-model semantics[1]. In this paper, we address the problem of the hyperequivalence of programs with respect to the other major semantics, that of supported models [4]. We define several concepts of hyperequivalence, depending on the class of programs allowed as contexts. We obtain characterizations of hyperequivalence with respect to supported models in terms of semantic objects, similar to SE-models [29] or UE-models [7], 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 [16, 17] and then, under the name of *ground* $\mathcal{S}$-expansions, for an arbitrary nonmonotonic modal logic $\mathcal{S}$ [15, 27]. The complexity of reasoning with moderately grounded expansion was established in [8] to be complete for

[1]There is little work on other semantics, with [3] being a notable exception.

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 with respect to 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 with respect to 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 with respect to supported (minimal) models.

2 Preliminaries

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

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

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 $\{a_1, \dots, a_k\}$ the head of r and denote it by $hd(r)$. Similarly, we call the conjunction $b_1, \dots, b_m, not\, c_1, \dots, not\, c_n$ the body of r and denote it by $bd(r)$. We will also use $bd^+(r) = \{b_1, \dots, b_m,\}$ and $bd^-(r) = \{c_1, \dots, c_n\}$, as well as $bd^\pm(r) = bd^+(r) \cup bd^-(r)$ to denote the set of all atoms occurring in the body of r. Moreover, for a program P, let $hd(P) = \bigcup_{r \in P} hd(r)$, and $bd^\pm(P) = \bigcup_{r \in P} bd^\pm(r)$.

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 [2, 13].

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

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

where $i = 1, \dots, k$. 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 [30], 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 $T_P(M) = M$) [1].

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 $T_{sh(P)}(M) = M$.

3 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 with respect to models, stable models, and hyperequivalence with respect to stable models.

Example 3.1 *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 with respect to 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 with respect to classical models. However, P_2 and Q_2 do not have the same stable models and so, they are not equivalent with respect to stable models nor hyperequivalent with respect to 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 with respect to stable models relative to any class of programs nor equivalent with respect to 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 with respect to this class of programs (we will verify this claim later). As we will see, supp-equivalence with respect to all *programs implies equivalence with respect to models and so, it is not a coincidence that in the last example we used a restricted class of contexts. To see that P_3 and Q_3 are* not *supp-equivalence with respect to the class of all programs, one can consider $R = \{b\}$. Then, $\{a, b\}$ 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 with respect to 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 with respect to the stable-model semantics in [31].

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

Theorem 3.2 *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 $\mathcal{HB}^n(A,B)$.*

Proof. The assertion follows directly from the fact that models and supported models of programs are preserved under shifting. □

Theorem 3.2 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

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

An interpretation $Y \in Mod_A(P)$ can be understood as a *candidate* for becoming a supported model of an extension $P \cup R$, where $R \in \mathcal{HB}^n(A,B)$. Indeed, such a candidate has to be classical model of P (otherwise it cannot be a supported model, no matter how P is extended). Moreover, the elements from $Y \setminus T_P(Y)$ have to be contained in A. Otherwise programs from $\mathcal{HB}^n(A,B)$ cannot close this gap.

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

Theorem 3.3 *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 $Y \in Mod_A(P)$, $T_P(Y) = T_Q(Y)$.*

Proof. ($\Rightarrow$) Since $\mathcal{HB}^n(A,\emptyset) \subseteq \mathcal{C}$, P and Q are supp-equivalent relative to $\mathcal{HB}^n(A,\emptyset)$.

Let $Y \in Mod_A(P)$. It follows that $Y \models P$ and $Y \setminus T_P(Y) \subseteq A$. Let us consider $P \cup (Y \setminus T_P(Y))$ Then

$$T_{P\cup(Y\setminus T_P(Y))}(Y) = T_P(Y) \cup (Y \setminus T_P(Y)).$$

Since $Y \models P$, $T_P(Y) \subseteq Y$. Hence, $T_{P\cup(Y\setminus T_P(Y))}(Y) = Y$. It follows that Y is a supported model of $P \cup (Y \setminus T_P(Y))$. Since $Y \setminus T_P(Y) \subseteq A$, $Y \setminus T_P(Y) \in \mathcal{HB}^n(A,\emptyset)$. Thus, Y is a supported model of $Q \cup (Y \setminus T_P(Y))$ and, consequently,

$$Y = T_{Q\cup(Y\setminus T_P(Y))}(Y) = T_Q(Y) \cup (Y \setminus T_P(Y)).$$

It follows that $T_Q(Y) \subseteq Y$ and $T_P(Y) \subseteq T_Q(Y)$. Thus, $Y \setminus T_Q(Y) \subseteq Y \setminus T_P(Y) \subseteq A$ and so, $Y \in Mod_A(Q)$. The converse inclusion follows by the symmetry argument and so, we have $Mod_A(P) = Mod_A(Q)$

Next, let $Y \in Mod_A(P)$ (and so, $Y \in Mod_A(Q)$, too). We have seen that $T_P(Y) \subseteq T_Q(Y)$. By the symmetry, $T_Q(Y) \subseteq T_P(Y)$. Thus, $T_P(Y) = T_Q(Y)$.

($\Leftarrow$) Let R be a logic program from $\mathcal{C}$ and Y be a supported model of $P \cup R$. It follows that $Y = T_{P\cup R}(Y) = T_P(Y) \cup T_R(Y)$. Thus, $T_P(Y) \subseteq Y$ (that is, $Y \models P$) and $Y \setminus T_P(Y) \subseteq A$ (because $hd(R) \subseteq A$). We obtain $Y \in Mod_A(P)$ and, by the assumption, $T_Q(Y) = T_P(Y)$. Hence, $Y = T_Q(Y) \cup T_R(Y) = T_{Q\cup R}(Y)$. That is, Y is a supported model of $Q \cup R$. □

We note that our characterization for supp-equivalence relative to $\mathcal{HB}^n(A, B)$ does not depend on the body-alphabet B of the context. Thus, Theorem 3.3 applies, in particular, to $\mathcal{C} = \mathcal{HB}^n(At, \emptyset)$ and $\mathcal{C} = \mathcal{HB}^n(At, At)$. Consequently, it characterizes strong and uniform supp-equivalence of normal programs. It also has several corollaries concerned with special cases for A. The first one deals with the case when $A = At$, in which the characterizing condition simplifies.

Corollary 3.4 *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, $T_P(Y) = T_Q(Y)$.*

Proof. When $A = At$, $Mod_A(P)$ and $Mod_A(Q)$ consist of models of P and Q, respectively. Thus, the result follows directly from Theorem 3.3. □

At the other extreme, we have the case $A = \emptyset$. In that case, all context programs consist of constraints (rules with the empty head) only. For the case $A = \emptyset$, we have the following result.

Corollary 3.5 *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. *$Mod_\emptyset(P) = Mod_\emptyset(Q)$.*

Proof. [(1)implies (2)]: Since $\emptyset \in \mathcal{HB}^n(\emptyset, \emptyset)$, the assertion is obvious.

[(2) implies (3)]: Let $Y \in Mod_\emptyset(P)$. Then, $Y \models P$, that is, $T_P(Y) \subseteq Y$, and $Y \setminus T_P(Y) = \emptyset$. Thus, $Y = T_P(Y)$ and, consequently, Y is a supported model of P. By the assumption, Y is a supported model of Q, that is $Y = T_Q(Y)$. It follows that $Y \in Mod_\emptyset(Q)$. The converse inclusion follows by the symmetry argument.

[(3) implies (1)]: Let $R \in \mathcal{C}$ and let Y be a supported model of $P \cup R$. Then $Y \models P \cup R$ and $Y = T_{P\cup R}(Y) = T_P(Y) \cup T_R(Y) = T_P(Y)$ (indeed, as $Y \models R$ and every rule in R is a constraint, $T_R(Y) = \emptyset$). Thus $Y \in Mod_\emptyset(P)$ and so, also $Y \in Mod_\emptyset(Q)$. From the latter we obtain $Y = T_Q(Y)$. Since $T_R(Y) = \emptyset$, $Y = T_Q(Y) \cup T_R(Y) = T_{Q\cup R}(Y)$, that is, Y is a supported model of $Q \cup R$. Again, the other implication follows by the symmetry argument. □

We will now apply our results to some pairs of programs discussed in Example 3.1.

Example 3.6 *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 3.4).*

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 $Y \setminus T_{P_3}(Y) \subseteq At \setminus \{b\}$. Clearly, if $Y \models P_3$, then $T_{P_3}(Y)$ is defined and $b \notin Y$. Thus, $Y \setminus T_{P_3}(Y) \subseteq At \setminus \{b\}$. It follows that $Y \in Mod_{At\setminus\{b\}}(P_3)$ if and only if $Y \models P_3$, that is, if and only if $a \in Y$ and $b \notin Y$. One can check that this condition also characterizes $Y \in Mod_{At\setminus\{b\}}(Q_3)$. Indeed, $Y \models Q_3$ if and only if $a \in Y$ and $Y \setminus T_{Q_3}(Y) \subseteq At \setminus \{b\}$ if and only if $b \notin Y$. 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 case of disjunctive programs.

Corollary 3.7 *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 $Y \in Mod_A(sh(P))$, $T_{sh(P)}(Y) = T_{sh(Q)}(Y)$.*

Proof. The result follows from Theorems 3.3 and 3.2. $\square$

Corollary 3.7 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 an observation, already noted above, 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 with respect to stable models [31]. In particular, for the semantics of supported models, there is no difference between strong and uniform equivalence (even for disjunctive programs).

Finally, we note that Theorem 3.3 also implies a characterization of uniform hyperequivalence with respect to stable models for *tight* logic programs [10], as for such programs stable and supported models coincide (we refer to [18] for a more detailed discussion of tight disjunctive logic programs and relevant results).

Corollary 3.8 *Let P and Q be tight disjunctive programs, $A \subseteq At$, and $\mathcal{C}$ a class of programs such that $\mathcal{HB}^n(A, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}^d(A, \emptyset)$. Then, P and Q are uniformly equivalent (with respect to the stable-model semantics) relative to $\mathcal{C}$ if and only if $Mod_A(sh(P)) = Mod_A(sh(Q))$ and for every $Y \in Mod_A(sh(P))$, $T_{sh(P)}(Y) = T_{sh(Q)}(Y)$.*

Proof. Let $R \in \mathcal{HB}^d(A, \emptyset)$. Since R consists of rules with the empty body, both $P \cup R$ and $Q \cup R$ are tight. Thus, they have the same stable models if and only if they have the same supported models. The assertion follows now from Corollary 3.7. $\square$

The characterization given by Corollary 3.8 provides an alternative to the characterization given in [12].

4 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 4.1 *The programs P_2 and Q_2 from Example 3.1 are suppmin-equivalent with respect to 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 with respect to stable models nor hyperequivalent with respect to stable models relative to* any *class of programs.*

Programs $P_4 = P_2$ and $Q_4 = \{a \leftarrow not\, a\}$ have the same models, stable models, and are hyperequivalent with respect to 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 with respect to 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!

We start by observing that, as before, it suffices to focus on the case of normal programs.

Theorem 4.2 *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 $\mathcal{HB}^n(A, B)$.*

Proof. Since models are preserved under shifting, minimal models are preserved under shifting, too. We already noted in the proof of Theorem 3.2 that shifting preserves supported models. Thus, the result follows. □

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 $Y \setminus T_P(Y) \subseteq X$.

The characterization is much more involved than the one for supp-equivalence. While the intuition for elements in $Mod_A(P)$ is the same as before, we now need an additional handle to ensure that minimality affects the two programs under comparison in the same way. Loosely speaking, for each $Y \in Mod_A(P)$, we have to keep track of those smaller models $X \subset Y$ which can be "influenced" differently by programs from $\mathcal{HB}^n(A, B)$. This is the task of items (2)-(4) in the definition above. The final condition, informally speaking again, states that such smaller models are of "no danger", as long as they cannot fill the gap for making Y supported. The forthcoming proof makes this intuition more concrete.

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

Theorem 4.3 *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 $(X, Y) \in Mod_A^B(P)$, $T_P(Y) \setminus (A \setminus B) = T_Q(Y) \setminus (A \setminus B)$.*

Proof. [(1) implies (2)]: Let $(X, Y) \in Mod_A^B(P)$. By the definition, $X \subseteq Y|_{A\cup B}$ and so, $X|_B \subseteq Y|_B$. If $X|_B = Y|_B$, we set

$$R' = (Y \setminus T_P(Y))|_B \cup X|_{A\setminus B}.$$

Otherwise, we fix an element $t \in Y|_B \setminus X|_B$ and define

$$R' = \{y \leftarrow t \mid y \in (Y \setminus T_P(Y)) \cup Y|_{A\setminus B}\} \cup \{x \leftarrow not\, t \mid x \in X|_A\}.$$

Finally, we set

$$R = R' \cup \{ \leftarrow not\, x \mid \ x \in X|_B\} \cup \{\leftarrow u, not\, z \mid u, z \in Y|_B \setminus X|_B\}.$$

We note that $R \in \mathcal{HB}^n(A,B)$. Indeed, since $(X,Y) \in Mod_A^B(P)$, $Y \in Mod_A(P)$. Thus, $Y \setminus T_P(Y) \subseteq A$.

It is easy to see that both Y and X satisfy the constraint rules in R. In addition, it is evident that $Y \models R'$. Thus, $Y \models R$. If $X|_B = Y|_B$, $(Y \setminus T_P(Y))|_B \subseteq Y|_B \subseteq X$ and, consequently, $X \models R'$. If $X|_B \subset Y|_B$, then $X \models R'$, as $t \notin X$. Thus, $X \models R$, as well.

Moreover, for each $Z \subseteq Y$ with $Z|_B = X|_B$ and $Z|_A \supseteq X|_A$, $Z \models R$ holds. Indeed, each such Z satisfies the constrains of R. In addition, if $X|_B = Y|_B$, then $Z|_B = Y|_B$ and so, $Z \models (Y \setminus T_P(Y))|_B$. Also, since $X|_A \subseteq Z|_A$, $Z \models X|_{A\setminus B}$. Thus, $Z \models R'$. If $X|_B \subset Y|_B$, then $Z \models R'$ by the fact that $t \notin Z$ (since $t \notin X|_B$, $t \notin Z|_B$ and, as $t \in B$, $t \notin Z$ follows).

Next, we show that Y is a supported model of $P \cup R$. Since $Y \in Mod_A(P)$ (it follows from the fact that $(X,Y) \in Mod_A^B(P)$), $Y \models P$. We already proved that $Y \models R$. Thus, $Y \models P \cup R$, that is, $T_{P\cup R}(Y) \subseteq Y$.

To prove the converse inclusion, we consider two cases. If $X|_B \subset Y|_B$, $T_R(Y) = (Y \setminus T_P(Y)) \cup Y|_{A\setminus B}$. Hence, $Y \subseteq T_P(Y) \cup T_R(Y) = T_{P\cup R}(Y)$. If $X|_B = Y|_B$, we have $T_R(Y) = (Y \setminus T_P(Y))|_B \cup X|_{A\setminus B}$. Let $y \in Y \setminus T_P(Y)$. It follows that $y \in X$ (condition (5) of the definition of $Mod_A^B(P)$) and $y \in A$ (by the fact that $Y \in Mod_A(P)$). If $y \notin B$, then $y \in X|_{A\setminus B}$ and so, $y \in T_R(Y)$. If $y \in B$, then $y \in (Y \setminus T_P(Y))|_B$. Thus, $y \in T_R(Y)$ in this case, too. It follows that if $y \in Y$ then $y \in T_P(Y) \cup T_R(Y)$ and consequently, $Y \subseteq T_P(Y) \cup T_R(Y) = T_{P\cup R}(Y)$.

We will now show that Y is a minimal model of $P \cup R$. To this end, let us consider $Z \subseteq Y$ such that $Z \models P \cup R$. It follows that $Z|_B \subseteq Y|_B$. Since $Z \models \{ \leftarrow not\, x \mid \ x \in X|_B\}$, $X|_B \subseteq Z|_B$. Thus, since $Z \models \{\leftarrow u, not\, z \mid u, z \in Y|_B \setminus X|_B\}$, we obtain that $Z|_B = Y|_B$ or $Z|_B = X|_B$.

Let us assume that $X|_B = Y|_B$. Since $Z \models R'$, $X|_{A\setminus B} \subseteq Z$. If $y \in X|_{A\cap B}$, then $y \in X|_B$ and $y \in Y|_B$. Thus, $y \in Z|_B$. It follows that $X|_A \subseteq Z$ and, consequently, $X|_A \subseteq Z|_A$. Moreover, $X|_B = Y|_B$ implies that $Z|_B = X|_B$. Since $Z \models P$, by the definition of $Mod_A^B(P)$ (condition (3)), $Z = Y$.

Thus, let us assume that $X|_B \subset Y|_B$. First, we consider the case $Z|_B = Y|_B$. Since $Z \models R$, we have $Y|_{A\setminus B} \subseteq Z$ (thanks to the rules $y \leftarrow t$, $y \in Y|_{A\setminus B}$, in R). it follows that $Y|_{A\cup B} \subseteq Z$ and, consequently, $Y|_{A\cup B} \subseteq Z|_{A\cup B}$. On the other hand $Z \subseteq Y$ and so, $Z|_{A\cup B} \subseteq Y|_{A\cup B}$. Thus, $Y|_{A\cup B} = Z|_{A\cup B}$. Since $Z \models P$, the definition of $Mod_A^B(P)$ implies $Y = Z$. Next, we consider the case $Z|_B = X|_B$. We have $X|_A \subseteq Z$ (thanks to the rules $x \leftarrow not\, t$, $x \in X|_A$, in R). Thus, as before, $Z = Y$ follows.

In each of the possible cases, $Y = Z$. Thus, Y is a minimal model of $P \cup R$.

Since P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A,B)$ and $R \in \mathcal{HB}^n(A,B)$, Y is a supported minimal model of $Q \cup R$. It follows that Y is a model of Q. In particular, $T_Q(Y) \subseteq Y$. Moreover, since $R \in \mathcal{HB}^n(A,B)$, and $Y = T_{Q\cup R}(Y) = T_Q(Y) \cup T_R(Y)$, $Y \setminus T_Q(Y) \subseteq A$. Thus, $Y \in Mod_A(Q)$, that is, the condition (1) for $(X,Y) \in Mod_A^B(Q)$ holds. The condition (2) holds simply because $(X,Y) \in Mod_A^B(P)$.

Since $Y \models R$ and $R \in \mathcal{HB}^n(A,B)$, for every $Z \subset Y$ such that $Z|_{A\cup B} = Y|_{A\cup B}$, $Z \models R$ and so, $Z \not\models Q$ (otherwise, we would have $Z \models Q \cup R$, contrary to the minimality of Y). If $Z \subset Y$,

$Z|_B = X|_B$ and $Z|_A \supseteq X|_A$ then, according to our earlier observation, $Z \models R$. Thus, for the same reason as before, $Z \not\models Q$ holds in this case, too. It follows that the conditions (3) and (4) for $(X,Y) \in Mod_A^B(Q)$ hold.

Finally, let us assume that $X|_B = Y|_B$. We recall that $T_R(Y) = (Y \setminus T_P(Y))|_B \cup X|_{A\setminus B}$. We also have $Y = T_Q(Y) \cup T_R(Y)$. Let $y \in Y \setminus T_Q(Y)$. Then $y \in (Y \setminus T_P(Y))|_B$ or $y \in X|_{A\setminus B}$. In the first case, $y \in X$, by the condition (5) for $(X,Y) \in Mod_A^B(P)$. In the second case, $y \in X$ is evident. Thus, the condition (5) for $(X,Y) \in Mod_A^B(Q)$ holds, too. Consequently, $(X,Y) \in Mod_A^B(Q)$ and $Mod_A^B(P) \subseteq Mod_A^B(Q)$.

We show next $T_P(Y) \setminus (A \setminus B) \subseteq T_Q(Y) \setminus (A \setminus B)$. To this end, let us consider $y \in T_P(Y) \setminus (A \setminus B)$. Clearly, $y \notin (A \setminus B)$ and it suffices to show that $y \in T_Q(Y)$. Since Y is a supported model of $Q \cup R$, we have $Y = T_Q(Y) \cup T_R(Y)$. Let us assume that $y \in T_R(Y)$. It follows that $y \in A$. Since $y \notin (A \setminus B)$, $y \in B$.

If $X|_B = Y|_B$, then $T_R(Y) = (Y \setminus T_P(Y))|_B \cup X|_{A\setminus B}$. Since $y \in B$, $y \notin X|_{A\setminus B}$. Thus, $y \in (Y \setminus T_P(Y))|_B$, and consequently, $y \notin T_P(Y)|_B$. We recall that $y \in B$ and so, $y \notin T_P(Y)$, a contradiction. It follows that $X|_B \subset Y|_B$. Then, we have $T_R(Y) = (Y \setminus T_P(Y)) \cup Y|_{A\setminus B}$. Since $y \in B$, $y \in Y \setminus T_P(Y)$, and so, $y \notin T_P(Y)$, a contradiction again. Thus, $y \notin T_R(Y)$. It follows that $y \in T_Q(Y)$, as needed.

We have shown that $Mod_A^B(P) \subseteq Mod_A^B(Q)$, and $T_P(Y) \setminus (A \setminus B) \subseteq T_Q(Y) \setminus (A \setminus B)$. By the symmetry argument, $Mod_A^B(Q) \subseteq Mod_A^B(P)$ holds, as well as, $T_Q(Y) \setminus (A \setminus B) \subseteq T_P(Y) \setminus (A \setminus B)$. Thus, (2) follows.

[(2) implies (3)]: Let $y \in T_P(Y)|_B$. Then $y \in B$ and so, $y \notin A \setminus B$. Since $y \in T_P(Y)$, $y \in T_P(Y) \setminus (A \setminus B)$. It follows that $y \in T_Q(Y) \setminus (A \setminus B)$. In particular, $y \in T_Q(Y)$ and, consequently, $y \in T_Q(Y)|_B$. The equality $T_P(Y)|_B = T_Q(Y)|_B$ follows by the symmetry argument.

[(3) implies (1)]: Let R be a logic program from $\mathcal{HB}^n(A,B)$, and let Y be a supported minimal model of $P \cup R$. Next, let $X = T_R(Y) \cup Y|_B$. We note that since $Y \models R$, $T_R(Y) \subseteq Y$. Thus, $X|_B = Y|_B$.

We will show that $(X,Y) \in Mod_A^B(P)$. Since Y is a suppmin model of $P \cup R$, it follows that $Y \models P$, $Y \models R$, and $Y = T_P(Y) \cup T_R(Y)$. We have $R \in \mathcal{HB}^n(A,B)$. Thus, the latter identity shows that $Y \setminus T_P(Y) \subseteq A$. Since $Y \models P$, we obtain that $Y \in Mod_A(P)$, that is, the condition (1) for $(X,Y) \in Mod_A^B(P)$ holds.

Since $T_R(Y) \subseteq Y$ and $T_R(Y) \subseteq A$ (we recall that $R \in \mathcal{HB}^n(A,B)$), $T_R(Y) \subseteq Y|_A \subseteq Y|_{A\cup B}$. Clearly, $Y|_B \subseteq Y|_{A\cup B}$. Thus, $X \subseteq Y|_{A\cup B}$. This proves that the condition (2) for $(X,Y) \in Mod_A^B(P)$ holds.

We also have $T_R(Y) \subseteq X$ (by the definition of X). Since $Y \setminus T_P(Y) \subseteq T_R(Y)$, $Y \setminus T_P(Y) \subseteq X$ follows. Consequently, the condition (5) for $(X,Y) \in Mod_A^B(P)$ holds, too.

Next, let $Z \subset Y$ and $Z|_{A\cup B} = Y|_{A\cup B}$. Since $R \in \mathcal{HB}^n(A,B)$ and $Y \models R$, $Z \models R$. We have that Y is a minimal model of $P \cup R$. Thus, $Z \not\models P$ and, consequently, the condition (3) for $(X,Y) \in Mod_A^B(P)$ follows.

Finally, let $Z \subset Y$, $Z|_B = X|_B$ and $Z|_A \supseteq X|_A$. Since $X|_B = Y|_B$, $Z|_B = Y|_B$. We have $R \in \mathcal{HB}^n(A,B)$. Thus, $T_R(Z) = T_R(Y) \subseteq X$ (the inclusion holds by the definition of X). Moreover, $T_R(Y) \subseteq A$ and so, $T_R(Z) \subseteq X|_A \subseteq Z|_A \subseteq Z$. Consequently, $Z \models R$ in this case, too. As before, we obtain that $Z \not\models P$. This shows the condition (4) for $(X,Y) \in Mod_A^B(P)$.

Thus, we have established that $(X, Y) \in Mod_A^B(P)$. By the assumption, $(X, Y) \in Mod_A^B(Q)$ and also $T_P(Y)|_B = T_Q(Y)|_B$. We will now show that $Y = T_Q(Y) \cup T_R(Y)$, that is, that Y is a supported model of $Q \cup R$. Since $Y \in Mod_A(Q)$, $Y \models Q$. Hence, $Y \models Q \cup R$ (we recall that $Y \models R$) and so, $T_Q(Y) \cup T_R(Y) \subseteq Y$.

To show $Y \subseteq T_Q(Y) \cup T_R(Y)$, let $y \in Y$. We recall that $T_R(Y) \subseteq A$. We distinguish three cases:

(i) $y \notin A$: Since $(X, Y) \in Mod_A^B(Q)$, $Y \in Mod_A(Q)$ and so, $Y \setminus T_Q(Y) \subseteq A$. Thus, $y \in T_Q(Y)$ follows.

(ii) $y \in B$: If $y \in T_R(Y)$ we are done; otherwise (since $Y = T_P(Y) \cup T_R(Y)$), we obtain $y \in T_P(Y)$. It follows that $y \in T_P(Y)|_B$. Thus, $y \in T_Q(Y)|_B$ and, consequently, $y \in T_Q(Y)$.

(iii) $y \in A \setminus B$: If $y \in X$, then $y \in T_R(Y)$ (we recall that $X = T_R(Y) \cup Y|_B$); if $y \notin X$, then $y \notin Y \setminus T_Q(Y)$ (indeed, since $(X, Y) \in Mod_A^B(Q)$ and $X|_B = Y|_B$, $Y \setminus T_Q(Y) \subseteq X$), and thus, $y \in T_Q(Y)$.

It follows that $Y = T_Q(Y) \cup T_R(Y)$, that is, Y is a supported model of $Q \cup R$.

It remains to show that Y is a minimal model of $Q \cup R$. Let $Y' \subset Y$ be a model of $Q \cup R$. Since $Y' \models Q$, $(Y'|_{A\cup B}, Y) \notin Mod_A^B(Q)$ (it violates condition (4) of the definition of $Mod_A^B(Q)$). By the assumption, $(Y'|_{A\cup B}, Y) \notin Mod_A^B(P)$. Since $(X, Y) \in Mod_A^B(P)$, $(Y'|_{A\cup B}, Y)$ satisfies condition (1) of the definition of $Mod_A^B(P)$. Moreover, $Y' \subset Y$ implies $Y'|_{A\cup B} \subseteq Y|_{A\cup B}$. Thus, condition (2) holds, too.

Let $Y|_B = (Y'|_{A\cup B})|_B$. Then, $Y|_B = Y'|_B$. Since $Y' \models R$ and $R \in \mathcal{HB}^n(A, B)$, $T_R(Y') \subseteq Y'$ and $T_R(Y') = T_R(Y)$. Thus, $T_R(Y) \subseteq Y'$ and, consequently, $X \subseteq Y'$. We proved above that $Y \setminus T_P(Y) \subseteq X$. Consequently, condition (5) for $(Y'|_{A\cup B}, Y) \in Mod_A^B(P)$ holds, as well. It follows that at least one of the conditions (3) and (4) is violated. That is, there is $U \subset Y$, such that $U \models P$, and either $U|_{A\cup B} = Y|_{A\cup B}$ or $U|_B = Y'|_B$ and $U|_A \supseteq Y'|_A$. Since both $Y \models R$ and $Y' \models R$, $U \models R$ (we recall here that $R \in \mathcal{HB}^n(A, B)$). Thus, $U \models P \cup R$ and Y is not a minimal model of $P \cup R$, a contradiction. It follows that there is no $Y' \subset Y$ such that $Y' \models Q \cup R$. That is, Y is a supported minimal model of $Q \cup R$. □

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 4.3 implies the following result.

Corollary 4.4 *Let $A \subseteq At$. Normal programs P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, \emptyset)$ if and only if $Mod_A^\emptyset(P) = Mod_A^\emptyset(Q)$.*

Proof. The result follows by the equivalence of the conditions (1) and (3) in Theorem 4.3. □

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. $Y \setminus T_P(Y) \subseteq X$.

We will discuss additional special-cases for instantiating $Mod_A^B(P)$ as part of our complexity analysis in Lemma 6.1.

When $A = B = At$ (strong suppmin-equivalence), it turns out that supp-equivalence and suppmin-equivalence coincide (cf. comments at the end of Example 4.1).

Corollary 4.5 *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 $\mathcal{HB}^n(At, At)$.*

Proof. We note that $(X,Y) \in Mod_{At}^{At}(P)$ if and only if $Y \in Mod_{At}(P)$, and either $X = Y$, or $X \subset Y$ and $X \not\models P$. Thus, $Mod_{At}^{At}(P) = Mod_{At}^{At}(Q)$ if and only if $Mod_{At}(P) = Mod_{At}(Q)$. Moreover, $(Y,Y) \in Mod_{At}^{At}(P)$ if and only if $Y \in Mod_{At}(P)$. Thus, the result follows from Corollary 3.4 and Theorem 4.3. □

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

Example 4.6 *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 4.4, 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 3.6) and so, not suppmin-equivalent (by Corollary 4.5).*

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 4.4, P_5 and Q_5 are not suppmin-equivalent relative to $\mathcal{HB}^n(\emptyset, \emptyset)$.

Thanks to Theorem 4.2, all results concerning normal programs lift to the disjunctive case. To illustrate it, we give two such results below.

Corollary 4.7 *Let $A, B \subseteq At$. The following conditions are equivalent.*

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

2. *$Mod_A^B(sh(P)) = Mod_A^B(sh(Q))$ and for every $(X,Y) \in Mod_A^B(sh(P))$, $T_{sh(P)}(Y) \setminus (A \setminus B) = T_{sh(Q)}(Y) \setminus (A \setminus B)$*

3. *$Mod_A^B(sh(P)) = Mod_A^B(sh(Q))$ and for every $(X,Y) \in Mod_A^B(sh(P))$, $T_{sh(P)}(Y)|_B = T_{sh(Q)}(Y)|_B$.*

Proof. This result follows by Theorem 4.2 from Theorem 4.3. □

Corollary 4.8 *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 $\mathcal{HB}^d(At, At)$.*

Proof. This result follows by Theorems 4.2 and 3.2, and Corollary 4.5. □

5 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:

1. SUPP: given programs P, Q (over At) and $A, B \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A, B)$
2. SUPP$_A$: given programs P, Q (over At) and $B \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A, B)$
3. SUPP^B: given programs P, Q (over At) and $A \subseteq At$, decide whether P and Q are supp-equivalent relative to $\mathcal{HB}^n(A, B)$
4. 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 5.1 *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* $B \subseteq At$.

Proof. (1) Theorem 3.3 implies that the complement of the problem is in the class NP. Indeed, given two programs P and Q and sets $A, B \subseteq At$, if there is a set Y such that Y belongs to exactly one of $Mod_A(P)$ and $Mod_A(Q)$, or $Y \in Mod_A(P) \cap Mod_A(Q)$ and $T_P(Y) \neq T_Q(Y)$, then $Y \setminus T_P(Y) \subseteq A$ or $Y \setminus T_Q(Y) \subseteq A$. It follows that $Y \subseteq At(P \cup Q) \cup A$. Since for such Y verifying the membership in $Mod_A(P)$ and $Mod_A(Q)$, and testing $T_P(Y) \neq T_Q(Y)$ can be done in polynomial time in the size of $At(P \cup Q) \cup A$, our claim and, consequently, the assertion, follows.
(2) Each of these problems reduces to the problem (1) (extend an instance of a problem in (2) with A, the set that defines the problem, to specify an instance of the problem (1)). Thus, the bound follows.
(3) and (4) Each of these problems is equivalent to the problem with $B = \emptyset$ and so, can be reduced to the problem (1).
(5) Corollary 3.4 implies that the complement of the problem is in the class NP. Indeed, given two normal programs P and Q, if there is a set Y such that (a) Y is a model of exactly one of P and Q, or (b) Y is a model of both P and Q, and $T_P(Y) \neq T_Q(Y)$, then there is $Y' \subseteq At(P \cup Q)$ with the same property. Since verifying conditions (a) and (b) for $Y' \subseteq At(P \cup Q)$ can be done in polynomial time in the size of $P \cup Q$, our claim and, consequently, the assertion, follows. $\Box$

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). In several proofs in this and the next sections, we use the following concepts and notation. We consider a CNF formula φ over a set of atoms Y, or a QBF formula $\forall Y \exists X \varphi$, where φ is a CNF formula over the set of atoms $X \cup Y$. For every such atom $z \in Y$ or $z \in X \cup Y$, respectively, we denote by z' a new atom not appearing anywhere in φ, possibly also different from some other atoms that might be named explicitly, and different from other "primed" atoms. Given a set of "non-primed" atoms Z, we define $Z' = \{z' \mid z \in Z\}$. Finally, for a clause $c = z_1 \vee \cdots \vee z_k \vee \neg z_{k+1} \vee \cdots \vee \neg z_m$, we denote by $\hat{c}$ the sequence $z'_1, \ldots, z'_k, z_{k+1}, \ldots, z_m$.

Theorem 5.2 *For every finite $A \subseteq At$ and $A = At$, and for every $B \subseteq At$, the problem* SUPP_A^B *is coNP-hard.*

Proof. Let us consider a CNF φ and let Y be the set of atoms in φ. We define

$$\begin{array}{rcl} P(\varphi) & = & \{y \leftarrow not\, y';\ y' \leftarrow not\, y \mid y \in Y\} \cup \\ & & \{\leftarrow y, y' \mid y \in Y\} \cup \\ & & \{\leftarrow \hat{c} \mid c \text{ is a clause in } \varphi\} \end{array}$$

To simplify the notation, we write P for $P(\varphi)$. One can check that φ has a model if and only if P has a model. Moreover, for every model M of P such that $M \subseteq At(P)$, M is a *stable* model of P. Thus, each such model of P is also a *supported* model of P and, consequently, satisfies $M = T_P(M)$.

Next, we define Q to consist of two rules: f and $\leftarrow f$. Clearly, Q has no models. By Theorem 3.3, Q is supp-equivalent to P relative to $\mathcal{HB}^n(A, B)$ if and only if $Mod_A(Q) = Mod_A(P)$ and for every $M \in Mod_A(Q)$, $T_Q(M) = T_P(M)$. Since $Mod_A(Q) = \emptyset$, we have that Q is supp-equivalent to P relative to $\mathcal{HB}^n(A, B)$ if and only if $Mod_A(P) = \emptyset$. If $M \in Mod_A(P)$, then there is $M' \subseteq At(P)$ such that $M' \in Mod_A(P)$. Since every model M' of P such that $M' \subseteq At(P)$ satisfies $M' = T_P(M')$, it follows that $Mod_A(P) = \emptyset$ if and only if P has no models.

Thus, φ is unsatisfiable if and only if Q is supp-equivalent to P relative to $\mathcal{HB}^n(A, B)$, and the assertion follows. □

We observe that for the result to hold we do not need to know B. Putting together Theorems 5.1 and 5.2 we obtain the following result.

Corollary 5.3 *The problems listed in Theorem 5.1 are coNP-complete.*

Proof. The hardness of problems in Theorem 5.1 follows from Theorem 5.2. Thus, the coNP-completeness follows. □

Problems we considered so far do not impose restrictions on input programs P and Q. In particular, they contain instances, in which $Mod_A(P) \neq Mod_A(Q)$, a property exploited by the proof we presented above. 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.

We start with an auxiliary result.

Lemma 5.4 *Let $A \subseteq At$ be a fixed finite non-empty set or $A = At$. The following problem is coNP-complete. Given a normal logic program P, decide whether every $M \in Mod_A(P)$ such that $M \subseteq At(P)$ is a supported model of P.*

Proof. Let us select and fix an element in A, say g, and let φ be a CNF formula over Y. Wlog we may assume that φ does not contain g. We define

$$\begin{array}{rcl} S(\varphi) & = & \{y \leftarrow not\, y';\ y' \leftarrow not\, y \mid y \in Y\} \cup \\ & & \{\leftarrow y, y' \mid\ y \in Y\} \\ & & \{g \leftarrow \hat{c} \mid c \text{ is a clause in } \varphi\} \end{array}$$

In the remainder of the proof, we write S for $S(\varphi)$.

We note that for every $M \subseteq At(S)$, if $M \models S$ then $T_S(M) = M$ or $T_S(M) = M \setminus \{g\}$. In particular, if $M \subseteq At(S)$ and $M \models S$, then $M \setminus T_S(M) \subseteq \{g\} \subseteq A$. Thus, for $M \subseteq At(S)$, $M \in Mod_A(S)$ if and only if $M \models S$.

If φ is unsatisfiable then, for every $M \subseteq At(S)$ such that $M \models S$, we have $g \in M$. Consequently, each such M is a supported model of S. It follows that for every $M \in Mod_A(S)$ such that $M \subseteq At(S)$, M is a supported model of S.

If φ is satisfiable, every model of φ gives rise to a supported model, say X, of S, such that $g \notin X$. It is easy to see that $M = X \cup \{g\}$ is a model of S but not a supported one. Since $M \subseteq At(S)$ and M is a model of S, $M \in Mod_A(S)$.

Thus, φ is unsatisfiable if and only if every $M \in Mod_A(S)$ such that $M \subseteq At(S)$ is a supported model of S. Consequently, the hardness follows.

The membership part is evident. Indeed, the complementary problem can be decided by the following algorithm: nondeterministically guess $M \subseteq At(S)$; verify that (1) $M \in Mod_A(S)$ and that (2) M is not supported model of S. Clearly both (1) and (2) can be done in polynomial time (both for A finite and nonempty, and for $A = At$, where the condition $M \setminus T_S(M) \subseteq A$ trivializes). Thus, the complementary problem is in NP and the assertion follows. $\square$

Applying Lemma 5.4 to the case $A = At$, we obtain the following result of some interest in its own right.

Theorem 5.5 *The following problem is coNP-complete: given a finite normal logic program P, decide whether every model Y of P such that $Y \subseteq At(P)$ is supported.*

However, the primary application of Lemma 5.4 is in determining the complexity of hyper-equivalence for programs P and Q with $Mod_A(P) = Mod_A(Q)$, the problem we already mentioned above.

Theorem 5.6 *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 $\mathcal{HB}^n(A, B)$.*

Proof. We restrict to the case $B = \emptyset$ (we recall that supp-equivalence does not depend on B).

The membership part follows from Theorem 5.1. For the hardness part, let P be a normal logic program. Let us consider $P' = P \cup \{g \leftarrow g \mid g \in At(P) \cap A\}$. We note that for every $M \subseteq At(P)$, $T_{P'}(M) = T_P(M) \cup (M \cap A)$.

We will first prove that $Mod_A(P) = Mod_A(P')$. Let $M \in Mod_A(P)$. Then $M \models P$ (and so, $T_P(M) \subseteq M$) and $M \setminus T_P(M) \subseteq A$. One can verify that $T_{P'}(M) = M \cap At(P)$. Thus, $M \models P'$. Moreover, as $T_P(M) \subseteq T_{P'}(M)$, $M \setminus T_{P'}(M) \subseteq A$. Consequently, $M \in Mod_A(P')$. Conversely, let $M \in Mod_A(P')$. It follows that $M \models P$. Next, let $y \in M \setminus T_P(M)$. If $y \in M \setminus T_{P'}(M)$, then $y \in A$ (as $M \setminus T_{P'}(M) \subseteq A$). Otherwise, $y \in T_{P'}(M) \setminus T_P(M)$. It follows that $y \in A \cap At(P)$ and so, $y \in A$ in this case, too. Thus, we obtain $M \setminus T_P(M) \subseteq A$ and, consequently, $M \in Mod_A(P)$.

Let us assume that for every $M \in Mod_A(P)$ such that $M \subseteq At(P)$, M is a supported model for P. Let $M \in Mod_A(P)$. Then, $M \cap At(P) \in Mod_A(P)$ and, by the assumption, $T_P(M \cap At(P)) = M \cap At(P)$. Since $T_P(M) = T_P(M \cap At(P))$, $T_P(M) = M \cap At(P)$. We also have $T_{P'}(M) = T_{P'}(M \cap At(P)) = T_P(M \cap At(P)) \cup [(M \cap At(P)) \cap A] = M \cap At(P)$. Thus, $T_P(M) = T_{P'}(M)$. By Theorem 3.3, P and P' are supp-equivalent relative to $\mathcal{HB}^n(A, B)$.

Conversely, let P and P' be supp-equivalent relative to $\mathcal{HB}^n(A, B)$ and let $M \in Mod_A(P)$ be such that $M \subseteq At(P)$. We proved earlier that for $M \in Mod_A(P)$, $T_{P'}(M) = M \cap At(P)$. Since $M \subseteq At(P)$, $T_{P'}(M) = M$. Thus, M is a supported model of P' and so (as $\emptyset \in \mathcal{HB}^n(A, B)$), of P, too.

It follows that for every $M \in Mod_A(P)$ such that $M \subseteq At(P)$, M is a supported model for P if and only if P and P' are supp-equivalent relative to $\mathcal{HB}^n(A, B)$. Since $Mod_A(P) = Mod_A(P')$, the assertion follows from Lemma 5.4. $\square$

There seems to be no simple reduction from any problem considered in Corollary 5.3 to the problem from Theorem 5.6 and so, a direct proof is needed. The requirement that $A \neq \emptyset$ is necessary for the complexity result of Theorem 5.6. Indeed, by Corollary 3.5, if $A = \emptyset$, programs P and Q with $Mod_A(P) = Mod_A(Q)$ are necessarily supp-equivalent.

6 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 and turns out to be coNP-complete. To prove that, we first simplify the conditions for $(X, Y) \in Mod_A^B$ if $A = At$ or $B = At$.

Lemma 6.1 *Let P be a normal logic program and $A, B \subseteq At$. Then,*

1. $(X, Y) \in Mod_{At}^B(P)$ *if and only if the following conditions hold:*
   - *(a)* $Y \models P$
   - *(b)* $X \subseteq Y$
   - *(c) for every* $Z \subset Y$*, such that* $Z|_B = X|_B$ *and* $Z \supseteq X$, $Z \not\models P$
   - *(d) if* $X|_B = Y|_B$*, then* $Y \setminus T_P(Y) \subseteq X$.
2. $(X, Y) \in Mod_A^{At}(P)$ *if and only if the following conditions hold:*
   - *(a)* $Y \in Mod_A(P)$
   - *(b)* $X \subseteq Y$
   - *(c) If* $X \subset Y$ *then* $X \not\models P$.

Proof. If $A = At$, the condition (1) for $(X, Y) \in Mod_A^B(P)$ specializes to (1a) as $Y \in Mod_{At}(P)$ if and only if $Y \models P$. Assuming $B = At$ has no effect on the condition (1). Thus, it appears without any change as the condition (2a).

The condition (2) for $(X, Y) \in Mod_A^B(P)$ specializes to (1b) or (2b), respectively, since $A \cup B = At$. For the same reason, the condition (3) for $(X, Y) \in Mod_A^B(P)$ is trivially true, in both cases. The conditions (4) and (5) for $(X, Y) \in Mod_A^B(P)$ specialize to the conditions (1c) and (1d) in case $A = At$. If $B = At$, the condition (4) for $(X, Y) \in Mod_A^B(P)$ specializes to (2c). The condition (5) for $(X, Y) \in Mod_A^B(P)$ holds true (if $X = Y$ then, trivially, $Y \setminus T_P(Y) \subseteq Y$) and can be dropped. $\square$

Lemma 6.1 has a corollary that plays a key role in establishing the membership in the class coNP of the relativized suppmin-equivalence problems for which $A = At$ or $B = At$. For the case $A = At$ we need one more important property.

Corollary 6.2 *Let P, Q be normal programs and $B \subseteq At$. If $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$, then there is $Y \subseteq At(P \cup Q) \cup B$ such that Y is a model of exactly one of P and Q, or there is $a \in Y$ such that $(Y \setminus \{a\}, Y)$ belongs to exactly one of $Mod^B_{At}(P)$ and $Mod^B_{At}(Q)$.*

Proof. Let us assume that P and Q have the same models (otherwise, there is $Y \subseteq At(P \cup Q)$ that is a model of exactly one of P and Q, and the assertion follows). Wlog we can assume that there is $(X, Y) \in Mod^B_{At}(P) \setminus Mod^B_{At}(Q)$. Moreover, we can assume that $Y \subseteq At(P \cup Q) \cup B$. It follows that (X, Y) satisfies the conditions (1a)-(1d) from Lemma 6.1 for $(X, Y) \in Mod^B_{At}(P)$. Moreover, since P and Q have the same models, (X, Y) already satisfies conditions (1a)–(1c) for $(X, Y) \in Mod^B_{At}(Q)$. Hence $X|_B = Y|_B$ and $Y \setminus T_Q(Y) \not\subseteq X$ have to hold. Thus, there is $a \in (Y \setminus T_Q(Y)) \setminus X$. We will show that $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$ and $(Y \setminus \{a\}, Y) \notin Mod^B_{At}(Q)$.

Since $(X, Y) \in Mod^B_{At}(P)$, Y is a model of P. Next, obviously, $Y \setminus \{a\} \subseteq Y$. Thus, the conditions (1a) and (1b) of Lemma 6.1 hold. Let $Z \subset Y$ be such that $Z \supseteq Y \setminus \{a\}$. Then $Z = Y \setminus \{a\}$. We have $Y|_B = X|_B$, $a \in Y$, and $a \notin X$. Thus, $a \notin B$. It follows that $(Y \setminus \{a\})|_B = X|_B$ and $X \subseteq Y \setminus \{a\}$. Since $(X, Y) \in Mod^B_{At}(P)$, $Y \setminus \{a\} \not\models P$, that is, $Z \not\models P$. Thus, the condition (1c) of Lemma 6.1 holds.

Since $a \notin B$, $(Y \setminus \{a\})|_B = Y|_B$. Thus, we also have to verify the condition (1d) of Lemma 6.1. We have $Y \setminus T_P(Y) \subseteq X$ (we recall that $Y|_B = X|_B$) and so, $Y \setminus T_P(Y) \subseteq Y \setminus \{a\}$. Hence, the condition (1d) of Lemma 6.1 holds, for P and, consequently, $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$. On the other hand, $a \in Y \setminus T_Q(Y)$ and $a \notin Y \setminus \{a\}$. Thus, the condition (1d) of Lemma 6.1 does not hold for Q and so, $(Y \setminus \{a\}, Y) \notin Mod^B_{At}(Q)$. □

We are now ready to show the promised coNP-completeness results.

Theorem 6.3 *The following problems are coNP-complete:*

1. SUPPMIN$^B_{At}$, *for every finite $B \subseteq At$,*
2. SUPPMIN$^{At}_A$, *for every finite $A \subseteq At$,*
3. SUPPMIN^{At}, SUPPMIN$_{At}$, *and* SUPPMIN$^{At}_{At}$.

Proof. The case of SUPPMIN$^{At}_{At}$ is already clear by Corollary 4.5 and Theorem 5.2.

To establish the other cases, we will show coNP-membership for SUPPMIN^{At} and SUPPMIN$_{At}$. From these two results, the membership results for SUPPMIN$^B_{At}$ and SUPPMIN$^{At}_A$ (for finite $A, B \subseteq At$) follow easily.

Likewise, we will show coNP-hardness for SUPPMIN$^B_{At}$ and SUPPMIN$^{At}_A$ (for finite $A, B \subseteq At$). That implies the corresponding lower bounds for SUPPMIN^{At} and SUPPMIN$_{At}$.

We start with coNP-membership for SUPPMIN$_{At}$. The following nondeterministic algorithm verifies, given program P, Q and $B \subseteq At$, that P and Q are not suppmin-equivalent relative

$\mathcal{HB}^n(At, B)$. We guess a pair (a, Y), where $Y \subseteq At(P \cup Q) \cup B$, and $a \in At$ such that (1) Y is a model of exactly one of P and Q; or (2) $a \in Y$ and $(Y \setminus \{a\}, Y)$ belongs to exactly one of $Mod^B_{At}(P)$ and $Mod^B_{At}(Q)$; or (3) Y is model of both P and Q and $T_P(Y)|_B \neq T_Q(Y)|_B$.

Such a pair exists if and only if P and Q are not suppmin-equivalent relative $\mathcal{HB}^n(At, B)$. Indeed, let us assume that such a pair (a, Y) exists. If (1) holds for (a, Y), say Y is a model of P but not Q, then $(Y, Y) \in Mod^B_{At}(P) \setminus Mod^B_{At}(Q)$ (easy to verify by means of Lemma 6.1). Thus, $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$ and, by Theorem 4.3, P and Q are not suppmin-equivalent relative $\mathcal{HB}^n(At, B)$. If (2) holds for (a, Y), $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$ again, and we reason as above. Finally, if neither (1) nor (2) holds, Corollary 6.2 implies $Mod^B_{At}(P) = Mod^B_{At}(Q)$. In this case, (3) holds for (a, Y). Since $Y \models P$, we have $(Y, Y) \in Mod^B_{At}(P)$. Moreover, $T_P(Y)|_B \neq T_Q(Y)|_B$. Thus, again by Theorem 4.3, P and Q are not suppmin-equivalent relative $\mathcal{HB}^n(At, B)$.

Conversely, if P and Q are not suppmin-equivalent relative $\mathcal{HB}^n(At, B)$, then $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$ or there is $(X, Y) \in Mod^B_{At}(P)$ such that $T_P(Y)|_B \neq T_Q(Y)|_B$. The former implies (by Corollary 6.2) that there is (a, Y), with $Y \subseteq At(P \cup Q) \cup B)$, that satisfies (1) or (2). Thus, let us assume that $Mod^B_{At}(P) = Mod^B_{At}(Q)$ and that there is $(X, Y) \in Mod^B_{At}(P)$ such that $T_P(Y)|_B \neq T_Q(Y)|_B$. Since $(X, Y) \in Mod^B_{At}(Q)$, too, Y is a model of both P and Q and $T_P(Y)|_B \neq T_Q(Y)|_B$. Clearly, $Y' = Y \cap (At(P \cup Q) \cup B)$ is a model of both P and Q, too, and $T_P(Y')|_B \neq T_Q(Y')|_B$. Picking any $a \in At$ yields a pair (a, Y'), with $Y' \subseteq At(P \cup Q) \cup B)$, for which (3) holds.

It follows that the algorithm is correct. Moreover, checking whether $Y \models P$ and $Y \models Q$ can clearly be done in polynomial time in the total size of P, Q, and B; the same holds for checking $T_P(Y)|_B \neq T_Q(Y)$. Finally, by Lemma 6.1, testing $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$ and $(Y \setminus \{a\}, Y) \in Mod^B_{At}(Q)$ are polynomial-time (with respect to the size of the input) tasks, too; the only problematic condition is (1c) from Lemma 6.1. However, we need to test that $Z \not\models P$ there for only one Z such that $Y \setminus \{a\} \subseteq Z \subset Y$, namely $Z = Y \setminus \{a\}$. Thus, the algorithm runs in polynomial time. It follows that the complement of our problem is in the class NP and so the assertion follows.

We continue with coNP-membership for SUPPMIN^{At}. By Theorem 4.3, P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, At)$ if and only if $Mod^{At}_A(P) = Mod^{At}_A(Q)$ and for every $(X, Y) \in Mod^{At}_A(Y)$, $T_P(Y) = T_Q(Y)$. It follows that to decide that P and Q are *not* suppmin-equivalent relative to $\mathcal{HB}^n(A, At)$, it suffices to guess a pair (X, Y), where $X \subseteq Y \subseteq At(P \cup Q) \cup A$, and verify that (a) (X, Y) belongs to exactly one of $Mod^{At}_A(P)$ and $Mod^{At}_A(P)$, or (b) that (X, Y) belongs to both $Mod^{At}_A(P)$ and $Mod^{At}_A(P)$, and $T_P(Y) \neq T_Q(Y)$. Indeed, if $(X, Y) \in Mod^{At}_A(P) \cup Mod^{At}_A(Q)$, then $X \subseteq Y \subseteq At(P \cup Q) \cup A$ (the latter inclusion following from the fact that $Y \in Mod_A(P) \cup Mod_A(Q)$). By Lemma 6.1, and since $X \subseteq Y \subseteq At(P \cup Q) \cup A$, all these tests can be executed in polynomial time in the total size of P, Q and A. Thus, the the complementary problem is in NP and so, the membership in coNP follows.

We now switch over to the hardness results and start this time with SUPPMIN$^{At}_A$. In the proof of Theorem 5.6, we have shown that for every program P, and any finite $A \subseteq At$, $Mod_A(P) = Mod_A(P')$, where $P' = P \cup \{g \leftarrow g \mid g \in A \cap At(P)\}$. Moreover, in the same proof, we have shown that P and P' are supp-equivalent relative to $\mathcal{HB}^n(A, At)$ if and only if for each $Y \in Mod_A(P)$ with $Y \subseteq At(P)$, Y is a supported model of P. Next, we show that P and P' are supp-equivalent relative to $\mathcal{HB}^n(A, At)$, if and only if P and P' are suppmin-equivalent relative to $\mathcal{HB}^n(A, At)$. The only-if direction follows from the fact that P and P' have the same models. Indeed, this property implies that for every $R \in \mathcal{HB}^n(A, At)$, $P \cup R$ and $Q \cup R$ have the same supported models and the same models. Consequently, they have the same supported minimal models.

For the if-direction, we recall that suppmin-equivalence relative to $\mathcal{HB}^n(A, At)$ between P and P' implies (by Theorem 4.3) $Mod_A^{At}(P) = Mod_A^{At}(P')$ and $T_P(Y) = T_{P'}(Y)$, for every $(X, Y) \in Mod_A^{At}(P)$. In view of Lemma 6.1, it is easy to see that $Mod_A(P) = Mod_A(P')$ follows. Moreover, by the same lemma, if $Y \in Mod_A(P)$ then $(Y, Y) \in Mod_A^{At}(P)$. Thus, $T_P(Y) = T_{P'}(Y)$, for every $Y \in Mod_A(P)$. By Theorem 3.3, P and P' are supp-equivalent relative to $\mathcal{HB}^n(A, At)$.

It follows that for every $Y \in Mod_A(P)$ with $Y \subseteq At(P)$, Y is a supported model for P if and only if P and P' are suppmin-equivalent relative to $\mathcal{HB}^n(A, At)$. Thus, the assertion follows from Lemma 5.4.

To prove the coNP-hardness of SUPPMIN$_{At}^{B}$, we proceed as follows. Let φ be a CNF formula, and let Y be the set of atoms in φ. We define P and Q as in the proof of Theorem 5.2. Since Q has no models, $Mod_{At}^B(Q) = \emptyset$.

Let Y be a model of P. Clearly, (Y, Y) satisfies the conditions (1a)-(1d) from Lemma 6.1. Thus, $Mod_{At}^B(P) \neq \emptyset$. Conversely, if $Mod_{At}^B(P) \neq \emptyset$, then there is $(X, Y) \in Mod_{At}^B(P)$. By Lemma 6.1(1a), Y is a model of P.

Thus, P has models if and only if $Mod_{At}^B(P) \neq \emptyset$. By Theorem 4.3, P has models if and only if P and Q are not suppmin-equivalent.

In the proof of Theorem 5.2, we already showed that P has models if and only if φ has models. Thus, φ has models if and only if P and Q are not suppmin-equivalent. Consequently, the claim follows. □

We will now establish the complexity of deciding relativized suppmin-equivalence when A and B are finite (fixed as part of the problem specification, or given as part of instance specification). We start with an auxiliary result needed to derive upper bounds for the complexity.

Lemma 6.4 *The following problem is in coNP: given a normal logic program P and sets $X, Y, A, B \subseteq At$, decide whether $(X, Y) \in Mod_A^B(P)$.*

Proof. We already established earlier that deciding whether $Y \notin Mod_A(P)$ (condition (1)) can be done in polynomial time in the size of P, Y and A. The same is evident for deciding $X \not\subseteq Y|_{A\cup B}$ (condition (2)) and $Y \setminus T_P(Y) \not\subseteq X$, in case, $X|_B = Y|_B$ (condition (5)).

The remaining two conditions defining $(X, Y) \in Mod_A^B(P)$, that is, (3) and (4), can be checked for violation as follows. We guess $Z \subset Y$ such that either $Z|_{A\cup B} = Y|_{A\cup B}$, or jointly $Z|_B = X|_B$ and $Z|_A \supseteq X|_A$. Then, we check whether $Z \models P$. Thus, deciding whether $(X, Y) \notin Mod_A^B(P)$, for given sets $X, Y, A, B \subseteq At$, is in the class NP. Consequently, deciding whether $(X, Y) \in Mod_A^B(P)$, for given sets $X, Y, A, B \subseteq At$, is in the class coNP. □

With this result in hand, Π_2^P-membership of SUPPMIN can be shown by suitably guessing pairs (X, Y) in $Mod_A^B(P)$ and $Mod_A^B(P)$, respectively.

Theorem 6.5 *The problem* SUPPMIN *is in* Π_2^P.

Proof. The complementary problem can be decided in non-deterministic polynomial time with an access to an NP-oracle. Indeed, we note that if $(X,Y) \in Mod_A^B(P)$, then $Y \subseteq At(P) \cup A$. Thus, if there exists (X,Y) that belongs to exactly one of $Mod_A^B(P)$ and $Mod_A^B(Q)$, then there is (X',Y') with that property and such that $Y' \subseteq At(P \cup Q) \cup A$. Moreover, if $Mod_A^B(P) = Mod_A^B(Q)$ and there is $(X,Y) \in Mod_A^B(P)$ such that $T_P(Y)|_B \neq T_Q(Y)|_B$, then there is $(X',Y') \in Mod_A^B(P)$ such that $Y' \subseteq At(P \cup Q) \cup A$ and $T_P(Y')|_B \neq T_Q(Y')|_B$. Thus, to decide the complementary problem, it suffices to guess sets $X, Y \subseteq At(P \cup Q) \cup A$ and check that (X,Y) is in exactly one of $Mod_A^B(P)$ and in $Mod_A^B(Q)$, or that (X,Y) is in both $Mod_A^B(P)$ and $Mod_A^B(Q)$, and $T_P(Y)|_B \neq T_Q(Y)|_B$. By Lemma 6.4, the tests for the membership in $Mod_A^B(P)$ and $Mod_A^B(Q)$ can be accomplished by an NP-oracle and all other tasks are evidently in the class P. $\Box$

We show the matching lower bound for the more specialized problem SUPPMIN$_B^A$.

Theorem 6.6 *The problem* SUPPMIN$_B^A$ *is* Π_2^P*-hard, for every finite* $A, B \subseteq At$.

Proof. Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$. We can assume that $(A \cup B) \cap X = \emptyset$ (if not, variables in X can be renamed). Next, we can assume that $A, B \subseteq Y$. Indeed, φ^+ obtained by expanding φ with clauses $z \vee \neg z$, for each $z \in A \cup B$, has the property that $\forall Y \exists X \varphi$ is true if and only if $\forall Y^+ \exists X \varphi^+$ is true, where $Y^+ = Y \cup A \cup B$.

We will construct programs $P(\varphi)$ and $Q(\varphi)$ so that $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are suppmin-equivalent relative to $\mathcal{HB}^n(A,B)$. Since the problem to decide whether a given QBF $\forall Y \exists X \varphi$ is true is Π_2^P-complete, the assertion will follow.

We use priming and $\hat{c}$ as discussed above and define the following programs:

$$\begin{array}{rcl}
P(\varphi) & = & \{z \leftarrow not\, z';\ z' \leftarrow not\, z \mid z \in X \cup Y\} \cup \\
& & \{\leftarrow y, y' \mid y \in Y\} \cup \\
& & \{x \leftarrow u, u';\ x' \leftarrow u, u' \mid x, u \in X\} \cup \\
& & \{x \leftarrow \hat{c};\ x' \leftarrow \hat{c} \mid x \in X, c \text{ is a clause in } \varphi\}; \\
Q(\varphi) & = & \{z \leftarrow not\, z';\ z' \leftarrow not\, z \mid z \in X \cup Y\} \cup \\
& & \{\leftarrow z, z' \mid z \in X \cup Y\} \cup \\
& & \{\leftarrow \hat{c} \mid c \text{ is a clause in } \varphi\}.
\end{array}$$

To simplify notation, from now on we write P for $P(\varphi)$ and Q for $Q(\varphi)$. We observe that $At(P) = At(Q) = W$, where $W = X \cup X' \cup Y \cup Y'$.

One can check that the models of Q contained in W are sets

$$I \cup (Y \setminus I)' \cup J \cup (X \setminus J)', \tag{1}$$

where $J \subseteq X$, $I \subseteq Y$ and $I \cup J \models \varphi$. Each model of Q is also a model of P but P has additional models contained in W. They are of the form:

$$I \cup (Y \setminus I)' \cup X \cup X', \tag{2}$$

for *each* $I \subseteq Y$. Clearly, for each model M of Q such that $M \subseteq W$, $T_Q(M) = M$. Similarly, for each model M of P such that $M \subseteq W$, $T_P(M) = M$.

From these comments, it follows that for every model M of Q (resp. P), $T_Q(M) = M \cap W$ (resp. $T_P(M) = M \cap W$). Since $B \subseteq W$, for every model M of both P and Q, $T_Q(M)|_B = M \cap W \cap B = T_P(M)|_B$. Thus, P and Q are suppmin-equivalent if and only if $Mod_A^B(P) = Mod_A^B(Q)$ (indeed, we recall that if $(N, M) \in Mod_A^B(R)$ then M is a model of R).

Let us assume that $\forall Y \exists X \varphi$ is false. Hence, there exists an assignment $I \subseteq Y$ to atoms Y such that for every $J \subseteq X$, $I \cup J \not\models \varphi$. Let $N = I \cup (Y \setminus I)' \cup X \cup X'$. We will show that $(N|_{A \cup B}, N) \in Mod_A^B(P)$.

Since N is a supported model of P, $N \in Mod_A(P)$. The requirement (2) for $(N|_{A\cup B}, N) \in Mod_A^B(P)$ is evident. The requirement (5) holds, since $N \setminus T_P(N) = \emptyset$. By the property of I, N is a minimal model of P. Thus, the requirements (3) and (4) hold, too. It follows that $(N|_{A\cup B}, N) \in Mod_A^B(P)$, as claimed. Since N is not a model of Q, $(N|_{A\cup B}, N) \notin Mod_A^B(Q)$.

Let us assume that $\forall Y \exists X \varphi$ is true. First, we observe that $Mod_A^B(Q) \subseteq Mod_A^B(P)$. Indeed, let $(M, N) \in Mod_A^B(Q)$. It follows that N is a model of Q and, consequently, of P. From our earlier comments, it follows that $T_Q(N) = T_P(N)$. Since $N \setminus T_Q(N) \subseteq A$, $N \setminus T_P(N) \subseteq A$. Thus, $N \in Mod_A(P)$. Moreover, if $M|_B = N|_B$ then $N \setminus T_Q(N) \subseteq M$ and, consequently, $N \setminus T_P(N) \subseteq M$. Thus, the requirement (5) for $(M, N) \in Mod_A^B(P)$ holds. The condition $M \subseteq N|_{A\cup B}$ is evident (it holds as $(M, N) \in Mod_A^B(Q)$). Since N is a model of Q, $N = N' \cup V$, where N' is of the form (1) and $V \subseteq At \setminus W$. Thus, every model $Z \subset N$ of P is also a model of Q. It implies that the requirements (3) and (4) for $(M, N) \in Mod_A^B(P)$ hold. Hence, $(M, N) \in Mod_A^B(P)$ and, consequently, $Mod_A^B(Q) \subseteq Mod_A^B(P)$.

We will now use the assumption that $\forall Y \exists X \varphi$ is true to prove the converse inclusion. To this end, let us consider $(M, N) \in Mod_A^B(P)$. If $N = N' \cup V$, where N' is of the form (1) and $V \subseteq At \setminus W$, then arguing as above, one can show that $(M, N) \in Mod_A^B(Q)$. Therefore, let us assume that $N = N' \cup V$, where N' is of the form (2) and $V \subseteq At \setminus W$. More specifically, let $N' = I \cup (Y \setminus I)' \cup X \cup X'$. By our assumption, there is $J \subseteq X$ such that $I \cup J \models \varphi$. That is, $Z = I \cup (Y \setminus I)' \cup J \cup (X \setminus J)'$ is a model of P. Clearly, $Z \subset N$. Moreover, since $A, B \subseteq Y$, it follows that $Z|_{A\cup B} = N|_{A\cup B}$. Since $(M, N) \in Mod_A^B(P)$, the requirement (3) implies that Z is not a model of P, a contradiction. Hence, the latter case is impossible and $Mod_A^B(P) \subseteq Mod_A^B(Q)$ follows.

We proved that $\forall Y \exists X \varphi$ is true if and only if $Mod_A^B(P) = Mod_A^B(Q)$. This completes the proof of the assertion. □

Putting together Theorems 6.5 and 6.6 yields the following corollary.

Theorem 6.7 *The following problems are Π_2^P-complete:*

1. SUPPMIN,

2. SUPPMIN^B, SUPPMIN$_A$, SUPPMIN$_A^B$, *for every finite $A, B \subseteq At$.*

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

Theorem 6.8 *Let $A, B \subseteq At$ be finite and such that $A \cap B \neq \emptyset$. The following problem is Π_2^P-complete: given normal programs P, Q such that $Mod_A^B(P) = Mod_A^B(Q)$, decide whether P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, B)$.*

Proof. The problem reduces to the one considered in Theorem 6.5. Thus, it belongs to Π_2^P. To prove Π_2^P-hardness we proceed as follows. Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF over $X \cup Y$. We can assume that $(A \cup B) \cap (X \cup Y) = \emptyset$ (as we can always rename variables in φ). Let us also choose and fix an element $g \in A \cap B$.

We use priming and $\hat{c}$ as before, and select an atom $x_0 \in X$. We define the following programs $P(\varphi)$ and $Q(\varphi)$:

$$\begin{aligned}
P(\varphi) &= \{z \leftarrow not\, z';\ z' \leftarrow not\, z \mid z \in X \cup Y\} \cup \\
&\quad\ \{\leftarrow y, y' \mid y \in Y\} \cup \\
&\quad\ \{x \leftarrow u, u';\ x' \leftarrow u, u' \mid x, u \in X\} \cup \\
&\quad\ \{x \leftarrow \hat{c};\ x' \leftarrow \hat{c} \mid x \in X, c \text{ is a clause in } \varphi\} \cup \\
&\quad\ \{\leftarrow not\, g;\ g \leftarrow x_0, not\, x_0';\ g \leftarrow x_0', not\, x_0\}; \\
Q(\varphi) &= P(\varphi) \cup \{g \leftarrow x_0, x_0'\};
\end{aligned}$$

To simplify notation, from now on we write P for $P(\varphi)$ and Q for $Q(\varphi)$. We set $W = X \cup X' \cup Y \cup Y' \cup \{g\}$.

Clearly, every model of P contains g. It follows that P and Q have the same models. To describe them, we first observe that every model of P (and Q) contained in W is of one of the following two types:

1. $\{g\} \cup I \cup (Y \setminus I)' \cup J \cup (X \setminus J)'$, for each $I \subseteq Y$ and $J \subseteq X$ such that $I \cup J \models \varphi$;

2. $\{g\} \cup I \cup (Y \setminus I)' \cup X \cup X'$, for each $I \subseteq Y$.

Thus, every model of P (and of Q) is of the form $N \cup S$, where $N \subseteq W$ is of type 1 or type 2, above, and $S \subseteq At \setminus W$. We refer to N as the W-*core* of the model $N \cup S$. We refer to a model of P (and Q) as type 1 or type 2, according to the form of its W-core.

Next, we observe that for every $N \subseteq At$, $T_P(N) \subseteq T_Q(N)$ and $T_Q(N) \setminus T_P(N) \subseteq \{g\}$. Let $N \in Mod_A(P)$. It follows that N is a model of P and so, of Q, too. We also have $N \setminus T_Q(N) \subseteq N \setminus T_P(N) \subseteq A$. It follows that $N \in Mod_A(Q)$. Conversely, let $N \in Mod_A(Q)$. Then, $N \models Q$ and so, $N \models P$, Moreover, $N \setminus T_P(N) \subseteq (N \setminus T_Q(N)) \cup \{g\} \subseteq A \cup \{g\} = A$. Thus, $N \in Mod_A(P)$. It follows that $Mod_A(P) = Mod_A(Q)$.

Let $(M, N) \in Mod_A^B(P)$. Then $N \in Mod_A(P)$ and so, $N \in Mod_A(Q)$. We also have $M \subseteq N|_{A \cup B}$. Thus, the conditions (1) and (2) required for $(M, N) \in Mod_A^B(Q)$ hold. The conditions (3) and (4) for $(M, N) \in Mod_A^B(Q)$ hold as they hold for P, and P and Q have the same models. Finally, the condition (5) for $(M, N) \in Mod_A^B(Q)$ holds, too, as $N \setminus T_Q(N) \subseteq N \setminus T_P(N)$. Thus, $Mod_A^B(P) \subseteq Mod_A^B(Q)$.

Conversely, let $(M, N) \in Mod_A^B(Q)$. Reasoning as above, we show that the conditions (1)-(4) for $(M, N) \in Mod_A^B(P)$ hold. To prove the condition (5), let us assume that $N|_B = M|_B$. Since

$N \in Mod_A(Q)$, N is a model of Q, and thus $g \in N$. Moreover, since $g \in B$, $g \in M$ as well. We have $N \setminus T_Q(N) \subseteq M$. Thus, $N \setminus T_P(N) \subseteq M$ follows from our previous observations. Consequently, the condition (5) for $(M, N) \in Mod_A^B(P)$ holds, and the inclusion $Mod_A^B(Q) \subseteq Mod_A^B(P)$ follows.

Since $Mod_A^B(P) = Mod_A^B(Q)$, P and Q form a valid instance to the problem we are considering. We will show that $\forall Y \exists X \varphi$ is true if and only if P and Q are suppmin-equivalent relative to $\mathcal{HB}^n(A, B)$.

Let us assume that $\forall Y \exists X \varphi$ is false. Then, there is $I \subseteq Y$ such that for every $J \subseteq X$, $I \cup J \not\models \varphi$. Let $N = \{g\} \cup I \cup (Y \setminus I)' \cup X \cup X'$. We have that $N \models Q$, and $T_Q(N) = N$. Thus, $N \setminus T_Q(N) = \emptyset \subseteq A$ and, consequently, $N \in Mod_A(Q)$. Let $M = N|_{A \cup B}$. By the definition, $M \subseteq N|_{A \cup B}$. Thus, the conditions (1) and (2) for $(M, N) \in Mod_A^B(Q)$ hold. Next, by the property of I, N is a minimal model of Q. It follows that (M, N) satisfies the conditions (3) and (4) for $(M, N) \in Mod_A^B(Q)$. Finally, we have $N \setminus T_Q(N) = \emptyset \subseteq M$. Thus, the condition (5) for $(M, N) \in Mod_A^B(Q)$ holds and so, $(M, N) \in Mod_A^B(Q)$. We observe that $T_P(N) = N \setminus \{g\}$ and $T_Q(N) = N$. Since $g \in B$ and $g \in N$, , $T_P(N)|_B \neq T_Q(N)|_B$ follows. Hence, by Theorem 4.3 (we recall that $(M, N) \in Mod_A^B(Q)$ and so, $(M, N) \in Mod_A^B(p)$), P and Q are not suppmin-equivalent relative $\mathcal{HB}^n(A, B)$.

Next, let us assume that $\forall Y \exists X \varphi$ is true. Let $(M, N) \in Mod_A^B(P)$. Let us assume that N is of the type 2. Let $\{g\} \cup I \cup (Y \setminus I)' \cup X \cup X'$, where $I \subseteq Y$, be the W-core of N. Since $\forall Y \exists X \varphi$ is true, there is $J \subseteq X$ such that $I \cup J \models \varphi$. We define $K = \{g\} \cup I \cup (Y \setminus I)' \cup J \cup (Y \setminus J)'$. Clearly, $K \models P$. We have $K \subset N$ and $K|_{A \cup B} = \{g\} = N|_{A \cup B}$ (we recall that $(A \cup B) \cap (W \setminus \{g\} = \emptyset)$. Thus, by the condition (3) for $(M, N) \in Mod_A^B(P)$, $K \not\models P$, a contradiction.

It follows that N is of type 1. Consequently, $T_P(N) = T_Q(N)$ and so, $T_P(N)|_B = T_Q(N)|_B$. By Theorem 4.3, 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 $Mod_A^B(P) = Mod_A^B(Q)$. Let $(X, Y) \in Mod_A^B(P)$ and $b \in T_P(Y)|_B$. Then $b \in Y$ (as $T_P(Y) \subseteq Y$) and $b \notin A$ (as $b \in B$ and $A \cap B = \emptyset$). 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.

7 Discussion

In this section, we discuss relations between the semantics of supported models and stable models in the context of hyperequivalence. We start with a comparison of the characterizations for the most important cases, strong and uniform equivalence. We then move on to highlight some interesting differences in the complexity.

First, let us consider characterizations of the notion of strong equivalence, i.e., hyperequivalence relative to the class of all programs, $\mathcal{HB}^d(At, At)$. To avoid references to $sh(P)$ and $sh(Q)$, we limit our discussion to the case when P and Q are normal. According to Corollary 3.4, normal programs P and Q are supp-equivalent in this sense if and only if

(1) P and Q have the same models, and for every model Y of P, $T_P(Y) = T_Q(Y)$.

We recall that in this case suppmin-equivalence has the same characterization (cf. Corollary 4.5). Let us thus turn to stable equivalence relative to $\mathcal{HB}(At, At)$. As shown in [29], the notion can be characterized in terms of SE-models, which are defined as follows. A pair of interpretations (X, Y) with $X \subseteq Y$ is an SE-model of a program P if $Y \models P$ and $X \models P^Y$. Two programs are equivalent relative to $\mathcal{HB}^n(At, At)$ under the stable-model semantics if and only if they have the same SE-models. A simple reformulation yields that P and Q are strongly equivalent in the stable models setting if and only if

(2) P and Q have the same models, and for every model Y of P, $Mod(P[Y]) = Mod(Q[Y])$,

where $P[Y] = P^Y \cup \{\leftarrow z \mid z \in At \setminus Y\}$, and $P^Y = \{hd(r) \leftarrow bd^+(r) \mid r \in P, Y \models bd^-(r)\}$ is the reduct of P with respect to Y. Despite differences, the basic intuition behind strong equivalence under supported- and stable-model semantics is quite similar for both settings. First, one checks whether the candidates Y (i.e., interpretations that might become a supported/stable model given an extension of the respective program) are the same for the two programs under consideration. Then, one checks whether any such extension has the same effect on both programs. In the case of the supported-model semantics, this is exactly the case if $T_P(Y) = T_Q(Y)$, while in the case of the stable-model semantics, the models of the reduct have to coincide.

Next, we will compare characterizations of uniform equivalence under supported minimal and stable models (we recall that, by Theorem 3.3, in case of supported models, strong and uniform equivalence coincide). Our characterization of suppmin-equivalence uses the definition of $Mod_A^B(P)$ as given in Section 4. This definition simplifies for uniform equivalence (i.e., for $A = At$ and $B = \emptyset$) as follows: $(X, Y) \in Mod_{At}^{\emptyset}(P)$ if and only if

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

By Corollary 4.4, uniform suppmin-equivalence between programs P and Q holds if and only if $Mod_{At}^{\emptyset}(P) = Mod_{At}^{\emptyset}(Q)$. To characterize uniform equivalence for the case of stable models, [7] introduced UE-models as special SE-models. A pair (X, Y) is an UE-model of P, if

1. $Y \models P$
2. $X \subseteq Y$
3. for each Z with $X \subset Z \subset Y$, $Z \not\models P^Y$
4. $X \models P^Y$.

Hence, in other words, UE-models of P are all SE-models of P of the form (Y, Y) plus SE-models (X, Y) of P, where X is maximal among the proper subsets of Y, which can appear with Y in an SE-model. Finite programs P and Q are uniform equivalent with respect to stable models if and only if the UE-models of P and Q coincide [7] (we note that the case of uniform equivalence of infinite programs has a slightly more elaborate characterization).

We will now compare the two characterizations for finite programs. Again, we observe that in the suppmin model case, $T_P(Y)$ plays a major role, while in the stable case, this role is taken over by the reduct P^Y. However, the remaining parts of the characterization show interesting similarities. On the one hand, as already discussed above, Y serves as a candidate to become a supported/stable model after some program extension. On the other hand, we observe that both characterizations depend on a very similar set of countermodels (either of the the program itself, or of the reduct P^Y) which are subsets of Y. For infinite programs, direct comparison of uniform equivalence under the two semantics gets harder since, as we noted, the UE-model characterization of uniform equivalence for stable-model semantics does not hold any more (see [7] for details on this issue).

We now turn to the complexity results, where some interesting differences can be observed (the complexity results for the stable model semantics we discuss below are from [7, 31]): First, deciding hyperequivalence with respect to supported models is coNP-complete, no matter how the context $\mathcal{HB}(A, B)$ is specified, as shown in Section 5. The same complexity class captures deciding hyperequivalence under stable models, *as long as we restrict to normal programs*. However, for disjunctive logic programs, deciding hyperequivalence in the stable-semantics setting is more complex for most instantiations of $\mathcal{HB}(A, B)$ (one exception is the case of strong equivalence, i.e., the case $A = B = At$, which remains coNP-complete). On the other hand, for supported models, disjunctions do not play a major role, and thus deciding hyperequivalence with respect to supported models remains in coNP even for disjunctive programs.

Changing the semantics to suppmin models has a more substantial effect, as we have shown in Section 6. Indeed, the complexity of deciding hyperequivalence with respect to suppmin models goes up to Π_2^P-completeness (already for normal programs). A notable exception is the case when at least one of A and B consists of all atoms, for which the corresponding problems of deciding hyperequivalence remain in coNP. Interestingly, this is not necessarily so in the stable-semantics world. As mentioned above, this holds for strong equivalence ($A = B = At$), but uniform equivalence ($A = At$, $B = \emptyset$) with respect to stable models remains Π_2^P-complete for disjunctive programs, while uniform suppmin-equivalence, as we noted, drops back to coNP.

Table 1 highlights these results in terms of completeness results, comparing the case of normal and disjunctive programs with respect to the different semantics and different instantiation of the context class, including strong-, uniform-, and the general case of hyperequivalence.

8 Conclusions

In this paper we extended the concept of hyperequivalence to two other major semantics of logic programs: the supported-model semantics and the supported minimal model semantics. We char-

normal / disjunctive	$\mathcal{HB}(At, At)$	$\mathcal{HB}(At, \emptyset)$	$\mathcal{HB}(A, B)$
supp	coNP/coNP	coNP/coNP	coNP/coNP
suppmin	coNP/coNP	coNP/coNP	Π_2^P/Π_2^P
stable	coNP/coNP	coNP/Π_2^P	coNP/Π_2^P

Table 1: Complexity of hyperequivalence for different semantics.

acterized these concepts of hyperequivalence and derived several complexity results.

Our characterizations were mainly based on the (partial) one-step provability operator T_P [30] and thus, unlike in the case of stable-model semantics, did not require any references to the reduct. However, some similarities to the case of the stable-model semantics appeared for more complex versions of hyperequivalence we studied, namely relativized supp- and suppmin-equivalence, which required additional concepts such as sets $Mod_A(P)$ and $Mod_A^B(P)$.

As concerns the complexity, the picture is uniform in the case of hyperequivalence with respect to supported models — problems that arise naturally turn out to be coNP-complete. The situation is different for hyperequivalence with respect to suppmin models. When at least one of the sets 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 and the decision problems become Π_2^P-complete. The results we presented demonstrate that with problems in which the departure from $A = At$ and $B = At$ is major: A and B are required to be finite (either as a parameter of the problem, or a part of the input). However, in some cases a much less drastic change has the same effect on the complexity. For instance, 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. A detailed analysis of this behavior is the subject of an ongoing work.

While to the best of our knowledge, this is the first paper concerning hyperequivalence for supported (minimal) semantics, hyperequivalence between programs with respect to other semantics have been studied extensively. The concept of uniform equivalence appeared first in the area of databases in the setting of DATALOG. In that setting queries are (non-ground) programs. Uniform equivalence of programs was introduced by [26], as a decidable approximation to query equivalence, and thus as a tool for query optimization. Several other equivalence notions in that context were studied in [23].

In the area of logic programming with the stable-model semantics, the need for stronger (than ordinary) equivalence was already recognized in [2, 13, 20], before [19] coined the name of strong equivalence for "equivalence for substitution." In particular, [2, 20] defined local rule transformations which retained the semantics of entire program and thus provided first explicit results in this area. Many successor papers of [19] then dealt with characterizations for strong equivalence [21, 29, 5], studied other forms of equivalence [14, 9, 25, 24, 31] or were concerned with programs transformations [7, 6, 22, 32].

We already addressed different realizations for hyperequivalence in this work. Future work thus is twofold. On the one hand, since our characterizations and techniques behind proofs are algebraic, generalizations to the language of partial operators on boolean algebras (cf. [28] for algebraic generalizations of hyperequivalence with respect to stable models) are of interest. 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 with respect to 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. The other direction concerns program simplification, for which our characterizations serve as a natural starting point. Moreover, in combination with the aforementioned extensions to autoepistemic logic, such techniques might also help to study new normal-form translations within that logic. As well, new investigations on the frontier between logic programs under supported models and nonmonotonic modal theories might be of interest.

References

[1] K. Apt. Logic Programming. In J. van Leeuven, editor, *Handbook of theoretical computer science*, pages 493–574. Elsevier, 1990.

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

[3] P. Cabalar, S. Odintsov, D. Pearce, and A. Valverde. Analysing and Extending Well-Founded and Partial Stable Semantics Using Partial Equilibrium Logic. In S. Etalle and M. iroslaw Truszczynski, editors, *Proceedings of the 22nd International Conference on Logic Programming (ICLP 2006)*, volume 4079 of *LNCS*, pages 346–360. Springer, 2006.

[4] K.L. Clark. Negation as Failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[5] Dick de Jongh and Lex Hendriks. Characterizations of Strongly Equivalent Logic Programs in Intermediate Logics. *Theory and Practice of Logic Programming*, 3(3):259–270, 2003.

[6] T. Eiter, M. Fink, H. Tompits, P. Traxler, and S. Woltran. Replacements in Non-Ground Answer-Set Programming. In P. Doherty, J. Mylopoulos, and C. Welty, editors, *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 340–351. AAAI Press, 2006.

[7] T. Eiter, M. Fink, and S. Woltran. Semantical Characterizations and Complexity of Equivalences in Answer Set Programming. *ACM Transactions on Computational Logic*, 8(3), 2007. 53 pages.

[8] T. Eiter and G. Gottlob. Reasoning with Parsimonious and Moderately Grounded Expansions. *Fundamenta Informaticae*, 17(1-2):31–53, 1992.

[9] T. Eiter, H. Tompits, and S. Woltran. On solution correspondences in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, pages 97–102. Morgan Kaufmann, 2005.

[10] F. Fages. Consistency of Clark's Completion and Existence of Stable Models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

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

[12] M. Gebser, T. Schaub, H. Tompits, and S. Woltran. Alternative Characterizations for Program Equivalence under Answer-Set Semantics Based on Unfounded Sets. In S. Hartmann and G. Kern-Isberner, editors, *Foundations of Information and Knowledge Systems, 5th International Symposium, FoIKS 2008, Proceedings*, volume 4932 of *LNCS*, pages 24–41. Springer, 2008.

[13] K. Inoue and C. Sakama. Negation as Failure in the Head. *Journal of Logic Programming*, 35:39–78, 1998.

[14] K. Inoue and C. Sakama. Equivalence of Logic Programs Under Updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, volume 3229 of *LNCS*, pages 174–186. Springer, 2004.

[15] M. Kaminski. Embedding a default system into nonmonotonic logic. *Fundamenta Informaticae*, 14(3):345–353, 1991.

[16] K. Konolige. On the Relation Between Default and Autoepistemic Logic. *Artificial Intelligence*, 35(3):343–382, 1988.

[17] K. Konolige. Errata: On the Relation between Default and Autoepistemic Logic. *Artificial Intelligence*, 41(1):115, 1989.

[18] J. Lee and V. Lifschitz. Loop Formulas for Disjunctive Logic Programs. In C. Palamidessi, editor, *Proceedings of the 19th International Conference on Logic Programming (ICLP 2003)*, volume 2916 of *LNCS*, pages 451–465. Springer, 2003.

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

[20] V. Lifschitz, L. Tang, and H. Turner. Nested Expressions in Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):369–389, 1999.

[21] F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In D. Fensel, D. McGuinness, and M.A. Williams, editors, *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 170–176. Morgan Kaufmann, 2002.

[22] Fangzhen Lin and Yin Chen. Discovering Classes of Strongly Equivalent Logic Programs. *Journal of Artificial Intelligence Research*, 28:431–451, 2007.

[23] M.J. Maher. Equivalences of Logic Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Morgan Kaufmann, 1988.

[24] J. Oetsch, H. Tompits, and S. Woltran. Facts do not Cease to Exist Because They are Ignored: Relativised Uniform Equivalence with Answer-Set Projection. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, pages 458–464. AAAI Press, 2007.

[25] E. Oikarinen and T. Janhunen. Modular Equivalence for Normal Logic Programs. In *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI 2006*), pages 412–416. IOS Press, 2006.

[26] Y. Sagiv. Optimising DATALOG Programs. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, 1988.

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

[28] M. Truszczynski. Strong and Uniform Equivalence of Nonmonotonic Theories – An Algebraic Approach. *Annals of Mathematics and Artificial Intelligence*, 48(3-4):245–265, 2006.

[29] H. Turner. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *Theory and Practice of Logic Programming*, 3(4-5):609–622, 2003.

[30] M.H. van Emden and R.A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742, 1976.

[31] S. Woltran. A Common View on Strong, Uniform, and Other Notions of Equivalence in Answer-Set Programming. *Theory and Practice of Logic Programming*, 8(2):217–234, 2008.

[32] K. Wong. Sound and Complete Inference Rules for SE-Consequence. *Journal of Artificial Intelligence Research*, 31:205–216, 2008.