Relativized Hyperequivalence of Logic Programs for Modular Programming

Mirosław Truszczyński[1] and Stefan Woltran[2]

[1] Department of Computer Science, University of Kentucky, Lexington, KY 40506, USA
[2] Institut für Informationssysteme 184/2, Technische Universität Wien, Favoritenstraße 9-11, A-1040 Vienna, Austria

Abstract. A recent framework of relativized hyperequivalence of programs offers a unifying generalization of strong and uniform equivalence. It seems to be especially well suited for applications in program optimization and modular programming due to its flexibility that allows us to restrict, independently of each other, the head and body alphabets in context programs. We study relativized hyperequivalence for the three semantics of logic programs given by stable, supported and supported minimal models. For each semantics, we identify four types of contexts, depending on whether the head and body alphabets are given directly or as the *complement* of a given set. Hyperequivalence relative to contexts where the head and body alphabets are specified directly has been studied before. In this paper, we establish the complexity of deciding relativized hyperequivalence wrt the three other types of context programs.

1 Introduction

We study variants of relativized hyperequivalence that are relevant for the development and analysis of logic programs with modular structure. Our main results concern the complexity of deciding relativized hyperequivalence for the three major semantics of logic programs given by stable, supported and supported minimal models.

Logic programming with the semantics of stable models, nowadays often referred to as *answer-set programming*, is a computational paradigm for knowledge representation, as well as modeling and solving of constraint problems [1,2,3,4]. In recent years, it has been steadily attracting more attention. One reason is that answer-set programming is truly declarative. Unlike in, say, Prolog, the order of rules in programs and the order of literals in rules have no effect on the meaning of the program. Secondly, the efficiency of the latest tools for processing programs, especially solvers, reached the level that makes it feasible to use them for problems of practical importance [5].

It is broadly recognized in software engineering that modular programs are easier to design, analyze and implement. Hence, essentially all programming languages and environments support the development of modular programs. Accordingly, there has been much work recently to establish foundations of *modular* answer-set programming. One line of investigations has focused on the notion of an answer-set program *module* [6,7,8,9]. This work builds on ideas for compositional semantics of logic programs proposed in [10] and encompasses earlier results on stratification and *program splitting* [11].

The other main line of research, to which our paper belongs, has centered on program equivalence and, especially, on the concept of equivalence for substitution. Programs P

and Q are *equivalent for substitution* wrt a class $\mathcal{C}$ of programs called *contexts*, if for every context $R \in \mathcal{C}$, $P \cup R$ and $Q \cup R$ have the same stable models. Thus, if a logic program is the union of programs P and R, where $R \in \mathcal{C}$, then P can be replaced with Q, with the guarantee that the semantics is preserved no matter what R is (as long as it is in $\mathcal{C}$) precisely when P and Q are equivalent for substitution wrt $\mathcal{C}$. If $\mathcal{C}$ contains the empty program (which is typically the case), the equivalence for substitution wrt $\mathcal{C}$ implies the standard equivalence under the stable-model semantics.[1] *The converse is not true*. We refer to these stronger forms of equivalence collectively as *hyperequivalence*.

Hyperequivalence wrt the class of *all* programs, known more commonly as *strong equivalence*, was proposed and studied in [12]. That work prompted extensive investigations of the concept that resulted in new characterizations [13,14] and connections to certain non-standard logics [15]. Hyperequivalence wrt contexts consisting of facts was studied in [16,17]. This version of hyperequivalence, known as *uniform equivalence*, appeared first in the database area in the setting of DATALOG and query equivalence [18]. Hyperequivalence wrt contexts restricted to a given alphabet, or *relativized* hyperequivalence, was proposed in [17,19]. It was generalized in [20] to allow contexts that use (possibly) different alphabets for the heads and bodies of rules. The approach offers a unifying framework for strong and uniform equivalence. Hyperequivalence, in which one compares projections of answer sets on some designated sets of atoms rather than entire answer sets was investigated in [21,22].

All those results concern the stable-model semantics of programs. There has been little work on other semantics, with [23] long being a notable single exception. Recently however, [24] introduced and investigated relativized hyperequivalence of programs under the semantics of supported models [25] and supported minimal models, two other major semantics of logic programs. [24] characterized these variants of hyperequivalence and established the complexity of some associated decision problems.

In this paper, we continue research of relativized hyperequivalence under all three major semantics of logic programs. As in [20,24], we focus on contexts of the form $\mathcal{HB}(A, B)$, where $\mathcal{HB}(A, B)$ stands for the set of all programs that use elements from A in the heads and atoms from B in the bodies of rules. Our main goal is to establish the complexity of deciding whether two programs are hyperequivalent (relative to a specified semantics) wrt $\mathcal{HB}(A, B)$. We consider the cases when A and B are either specified directly or in terms of their complement. As we point out in the following section, such contexts arise naturally when we design modular logic programs.

2 Motivation

In the paper we consider finite propositional programs only, all over a fixed countable infinite set of atoms At. For a set of atoms X, we define $X^c = At \setminus X$.

A logic program is *A-defining* if it specifies the definitions of atoms in A. The definitions may be recursive, they may involve *interface* atoms, that is, atoms defined in other modules, as well as atoms used locally to represent some needed auxiliary concepts. Let L be the set of local atoms, and let P be a particular logic program expressing the

[1] Two programs are equivalent under the stable-model semantics if they have the same stable models.

definitions. For P to behave properly when combined with other programs, these "context" programs must not have any occurrences of atoms from L and must have no atoms from A in the heads of their rules. In our terminology, these are precisely programs in $\mathcal{HB}((A \cup L)^c, L^c)$.[2]

The definitions of atoms in A can in general be captured by several different A-defining programs. A key question concerning such programs is whether they are equivalent. Clearly, two A-defining programs P and Q, both using atoms from L to represent local auxiliary concepts, should be regarded as equivalent if they behave in the same way in the context of any program from $\mathcal{HB}((A \cup L)^c, L^c)$. In other words, the notion of equivalence that is appropriate in our setting is hyperequivalence wrt $\mathcal{HB}((A \cup L)^c, L^c)$ under a selected semantics (stable, supported or supported-minimal).

Example 1. Let us assume that $A = \{a, b\}$ and that c and d are interface atoms (atoms defined elsewhere). We need a module that works as follows:

1. If c and d are both true, exactly one of a and b must be true
2. If c is true and d is false, only a must be true
3. If d is true and c is false, only b must be true
4. If c and d are both false, a and b must be false.

We point out that c and d may depend on a and b and so, in some cases the overall program may have no models of a particular type (to be concrete, for a time being we fix attention to stable models).

One way to express above conditions is by means of the following $\{a, b\}$-defining program P (in this example we assume programs without local atoms, that is, $L = \emptyset$):

$$a \leftarrow c, \mathit{not}\, b; \quad b \leftarrow d, \mathit{not}\, a.$$

Combining P with programs that specify facts: $\{c, d\}$, $\{c\}$, $\{d\}$ and $\emptyset$, it is easy to see that P behaves as required. For instance, $P \cup \{c\}$ has exactly one stable model $\{a, c\}$.

However, P may also be combined with more complex programs. For instance, let us consider the program $R = \{c \leftarrow \mathit{not}\, d;\ d \leftarrow a, \mathit{not}\, c\}$. Here, d can only be true if a is true and c is false. But then b must be true and a must be *false*, a contradiction. Thus, d must be false and c must be true. According to the specifications, there should be exactly one stable model for $P \cup R$ in this case: $\{a, c\}$. It is easy to verify that it is indeed the case.

The specifications for a and b can also be expressed by other $\{a, b\}$-defining programs, in particular, by the following program Q:

$$a \leftarrow c, d, \mathit{not}\, b; \quad b \leftarrow c, d, \mathit{not}\, a; \quad a \leftarrow c, \mathit{not}\, d; \quad b \leftarrow d, \mathit{not}\, c.$$

The question arises whether Q behaves in the same way as P relative to programs from $\mathcal{HB}(\{a, b\}^c, \emptyset^c) = \mathcal{HB}(\{a, b\}^c, At)$. For all contexts considered earlier, it is the case. However, in general, it is not so. For instance, if $R = \{c \leftarrow\ ;\ d \leftarrow a\}$ then, $\{a, c, d\}$ is a stable model of $P \cup R$, while $Q \cup R$ has no stable models. Thus, P and Q cannot be viewed as equivalent $\{a, b\}$-defining programs. □

A similar scenario is as follows: We call a program *A-completing* if it completes partial and non-recursive definitions of atoms in A given elsewhere in the overall program

[2] A-defining programs were introduced in [26]. However, that work considered more restricted classes of programs with which A-defining programs could be combined.

(which, for instance, might specify the base conditions for a recursive definition of atoms in A). Assuming that P is an implementation of such a module (again with L as a set of local atoms), P can be combined with any program R that has no occurrences of atoms from L and no occurrences of atoms from A in the bodies of its rules. This is precisely the class $\mathcal{HB}(L^c, (A \cup L)^c)$.

One can also construct scenarios that give rise to hyperequivalence wrt context classes $\mathcal{HB}(A, B)$, where A or B is specified directly. Thus, hyperequivalence wrt context classes $\mathcal{HB}(A, B)$, where each of A and B is specified directly or in terms of its complement is of interest. Our goal is to study the complexity of deciding whether two programs are hyperequivalent relative to such classes of contexts.

3 Technical Preliminaries

Basic logic programming notation and definitions. *Disjunctive logic programs* (programs, for short) are sets of (program) *rules* — expressions of the form

$$a_1|\ldots|a_k \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n, \tag{1}$$

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*. Programs consisting of normal rules are called *normal*.

We often write the rule (1) as $H \leftarrow B^+, not\, B^-$, where $H = \{a_1, \ldots, a_k\}$, $B^+ = \{b_1, \ldots, b_m\}$ and $B^- = \{c_1, \ldots, c_n\}$. We call H the *head* of the rule, and the conjunction $B^+, not\, B^-$, the *body* of the rule. The sets B^+ and B^- form the positive and negative body of the rule. Given a rule r, we write $H(r)$, $B(r)$, $B^+(r)$ and $B^-(r)$ to denote the head, the body, the positive body and the negative body of r, respectively. For a program P, we set $H(P) = \bigcup_{r \in P} H(r)$, and $B^\pm(P) = \bigcup_{r \in P} B^+(r) \cup B^-(r)$.

For an interpretation $M \subseteq At$ and a rule r, we define entailments $M \models B(r)$, $M \models H(r)$ and $M \models r$ in the standard way. An interpretation $M \subseteq At$ is a *model* of a program P ($M \models P$), if $M \models r$ for every $r \in P$.

The *reduct* of a disjunctive logic program P wrt a set M of atoms, denoted by P^M, is the program $\{H(r) \leftarrow B^+(r) \mid r \in P,\ M \cap B^-(r) = \emptyset\}$. A set M of atoms is a *stable model* of P if M is a minimal model (wrt inclusion) of P^M.

If a set M of atoms is a minimal hitting set of $\{H(r) \mid r \in P,\ M \models B(r)\}$, then M is a *supported model* of P. M is a *supported minimal model* of P if it is a supported model of P and a minimal model of P. We recall that each stable model is also supported, but (generally) not vice versa. Supported models of a *normal* logic program P have a useful characterization in terms of the (partial) one-step provability operator T_P, defined as follows. For $M \subseteq At$, if there is a constraint $r \in P$ such that $M \models B(r)$ (that is, $M \not\models r$), then $T_P(M)$ is undefined. Otherwise, $T_P(M) = \bigcup\{H(r) \mid r \in P,\ M \models B(r)\}$. 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 $T_P(M) = M$) [27].

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

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

where $i = 1, \ldots, k$. If r is normal, 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 M of atoms is a (minimal) model of P if and only if M is a (minimal) model of $sh(P)$. It is easy to check that M is a supported (minimal) model of P if and only if it is a supported (minimal) model of $sh(P)$. Moreover, M is a supported model of P if and only if $T_{sh(P)}(M) = M$. Thus, in all results concerning supported models, we will use implicitly the shift of programs involved (see also [24] for further details).

Characterizations of hyperequivalence of programs. Let $\mathcal{C}$ be a class of (disjunctive) logic programs. Programs P and Q are *supp-equivalent* (*suppmin-equivalent*, *stable-equivalent*, respectively) relative to $\mathcal{C}$ if for every program $R \in \mathcal{C}$, $P \cup R$ and $Q \cup R$ have the same supported (supported minimal, stable, respectively) models.

In this paper, we are interested in equivalence of all three types relative to classes of programs defined by the *head* and *body alphabets*. Let $A, B \subseteq At$. By $\mathcal{HB}(A, B)$ we denote the class of all programs P such that $H(P) \subseteq A$ and $B^{\pm}(P) \subseteq B$. Hence, the empty program is contained in any such $\mathcal{HB}(A, B)$.

For supp-equivalence and suppmin-equivalence, we need the following concept introduced in [24]. Given a program P, and a set $A \subseteq At$, we define

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

Theorem 1. *Let P and Q be programs, $A \subseteq At$, and $\mathcal{C}$ a class of programs such that $\mathcal{HB}(A, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}(A, At)$. Then, P and Q are supp-equivalent relative to $\mathcal{C}$ if and only if $Mod_A(P) = Mod_A(Q)$ and for every $Y \in Mod_A(P)$, $T_P(Y) = T_Q(Y)$.*

To characterize suppmin-equivalence, we use the set $Mod_A^B(P)$ (following [24]), which consists 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 $Y \setminus T_P(Y) \subseteq X$.

Theorem 2. *Let $A, B \subseteq At$ and let P, Q be programs. Then, P and Q are suppmin-equivalent relative to $\mathcal{HB}(A, B)$ if and only if $Mod_A^B(P) = Mod_A^B(Q)$ and for every $(X, Y) \in Mod_A^B(P)$, $T_P(Y)|_B = T_Q(Y)|_B$.*

Relativized stable-equivalence of programs was characterized in [20]. We define $SE_A^B(P)$ to consist of all pairs (X, Y) such that:[3]

1. $Y \models P$
2. $X = Y$ or jointly $X \subseteq Y|_{A \cup B}$ and $X|_A \subset Y|_A$
3. for each $Z \subset Y$ such that $Z|_A = Y|_A$, $Z \not\models P^Y$
4. for each $Z \subset Y$ such that $Z|_B \subseteq X|_B$, $Z|_A \supseteq X|_A$, and either $Z|_B \subset X|_B$ or $Z|_A \supset X|_A$, $Z \not\models P^Y$
5. there is $Z \subseteq Y$ such that $X|_{A \cup B} = Z|_{A \cup B}$ and $Z \models P^Y$.

[3] We use a slightly different presentation than [20].

Theorem 3. *Let $A, B \subseteq At$ and let P, Q be programs. Then, P and Q are stable-equivalent relative to $\mathcal{HB}(A, B)$ if and only if $SE_A^B(P) = SE_A^B(Q)$.*

Decision problems. We study problems of deciding hyperequivalence relative to program classes $\mathcal{HB}(A', B')$, where A' and B' stand either for finite sets or for complements of finite sets. In the former case, the set is given *directly*. In the latter, it is specified by its finite *complement* (the set itself is infinite). Thus, we obtain the classes of *direct-direct*, *direct-complement*, *complement-direct* and *complement-complement* decision problems. We denote them using strings of the form $\text{SEM}_{\delta,\varepsilon}(\alpha, \beta)$, where (1) SEM stands for SUPP, SUPPMIN or STABLE and identifies the semantics relative to which we define hyperequivalence; (2) δ and ε stand for d or c (direct and complement, respectively), and specify one of the four classes of problems mentioned above; (3) α is either $\cdot$ or A, where $A \subseteq At$ is finite. If $\alpha = A$, then α specifies a *fixed* alphabet for the heads of rules in contexts: either A or the complement A^c of A, depending on whether $\delta = d$ or c. The parameter does not belong to and does not vary with input. If $\alpha = \cdot$, then the specification A of the head alphabet is part of the input and defines it as A or A^c, again according to δ; (4) β is either $\cdot$ or B, where $B \subseteq At$ is finite. It obeys the same conventions as α but defines the body alphabet according to the value of ε.

For instance, $\text{SUPPMIN}_{d,c}(A, \cdot)$, where $A \subseteq At$ is finite, stands for the following problem: given programs P and Q, and a set B, decide whether P and Q are suppmin-equivalent wrt $\mathcal{HB}(A, B^c)$. With some abuse of notation, we often talk about "the problem $\text{SEM}_{\delta,\varepsilon}(A, B)$" as a shorthand for "an arbitrary problem of the form $\text{SEM}_{\delta,\varepsilon}(A, B)$ with fixed finite sets A and B"; likewise we do so for $\text{SEM}_{\delta,\varepsilon}(\cdot, B)$ and $\text{SEM}_{\delta,\varepsilon}(A, \cdot)$.

As we noted, for supp- and suppmin-equivalence, there is no difference between normal and disjunctive programs. For stable-equivalence, allowing disjunctions in the heads of rules affects the complexity. Thus, in the case of stable-equivalence, we distinguish versions of the problems $\text{STABLE}_{\delta,\varepsilon}(\alpha, \beta)$, where the input programs are normal.[4] We denote these problems by $\text{STABLE}^n_{\delta,\varepsilon}(\alpha, \beta)$.

Direct-direct problems for the semantics of supported and supported minimal models were considered in [24] and their complexity was fully determined there. The complexity of problems $\text{STABLE}_{d,d}(\cdot, \cdot)$, was established in [20], and problems similar to $\text{STABLE}_{c,c}(A, A)$ were already studied in [17]. In this paper, we complete the results on the complexity of problems $\text{SEM}_{\delta,\varepsilon}(\alpha, \beta)$ for all three semantics. In particular, we establish the complexity of the problems with at least one of δ and ε being equal to c.

The complexity of problems involving the complement of A or B is not a straightforward consequence of the results on direct-direct problems. In the direct-direct problems, the class of context programs is essentially finite, as the head and body alphabets for rules are finite. It is no longer the case for the three remaining problems, where at least one of the alphabets is infinite and so, the class of contexts is infinite, as well.

Finally, we note that when we change A or B to $\cdot$ in the problem specification, the resulting problem is at least as hard as the original one. Indeed for each such pair of problems, there are some straightforward reductions from one to the other. We illustrate these relationships in Figure 1.

[4] We can also restrict the programs used as contexts to normal ones, since this makes no difference, cf. [20].

$$\begin{tikzcd}
 & \mathrm{SEM}_{\delta,\varepsilon}(\cdot,B) \arrow[dr] & \\
\mathrm{SEM}_{\delta,\varepsilon}(A,B) \arrow[ur] \arrow[dr] & & \mathrm{SEM}_{\delta,\varepsilon}(\cdot,\cdot) \\
 & \mathrm{SEM}_{\delta,\varepsilon}(A,\cdot) \arrow[ur] &
\end{tikzcd}$$

Fig. 1. A simple comparison of the hardness of problems

4 Supp-Equivalence

As the alphabet for the bodies of context programs plays no role in supp-equivalence (cf. Theorem 1), the complexity of $\mathrm{SUPP}_{d,c}(A,\beta)$ and $\mathrm{SUPP}_{d,c}(\cdot,\beta)$ is already solved (β is $\cdot$ or a set B of atoms) by the complexity of the corresponding direct-direct problems which have been shown coNP-complete in [24]. It remains to consider $\mathrm{SUPP}_{c,d}(A,\beta)$ and $\mathrm{SUPP}_{c,d}(\cdot,\beta)$ (which coincide with $\mathrm{SUPP}_{c,c}(A,\beta)$, and respectively, $\mathrm{SUPP}_{c,c}(\cdot,\beta)$).

First, we prove an upper bound on the complexity of the problem $\mathrm{SUPP}_{c,d}(\cdot,\cdot)$.

Theorem 4. *The problem* $\mathrm{SUPP}_{c,d}(\cdot,\cdot)$ *is in the class coNP.*

Proof: It is sufficient to show that $\mathrm{SUPP}_{c,d}(\cdot,\emptyset)$ is in coNP, since (P,Q,A) is a YES instance of $\mathrm{SUPP}_{c,d}(\cdot,\emptyset)$ if and only if (P,Q,A,B) is a YES instance of $\mathrm{SUPP}_{c,d}(\cdot,\cdot)$.

Let $Y' = Y \cap (At(P) \cup A)$. We will show that $Y \in Mod_{A^c}(P)$ if and only if $Y' \in Mod_{A^c}(P)$. First, we note that $T_P(Y) = T_P(Y')$. If $Y \in Mod_{A^c}(P)$, then $Y \models P$ and $Y \setminus T_P(Y) \subseteq A^c$. The former property implies that $Y' \models P$. Since $Y' \setminus T_P(Y') = Y' \setminus T_P(Y) \subseteq Y \setminus T_P(Y)$, the latter one implies that $Y' \setminus T_P(Y') \subseteq A^c$. Thus, $Y' \in Mod_{A^c}(P)$.

Conversely, let $Y' \in Mod_{A^c}(P)$. Then $Y' \models P$ and, consequently, $Y \models P$. Moreover, we also have $Y' \setminus T_P(Y') \subseteq A^c$. Let $y \in Y \setminus T_P(Y)$. If $y \notin Y'$, then $y \notin A$, that is, $y \in A^c$. If $y \in Y'$, then $y \in Y' \setminus T_P(Y')$ (we recall that $T_P(Y) = T_P(Y')$). Hence, $y \in A^c$ in this case, too. It follows that $Y \setminus T_P(Y) \subseteq A^c$ and so, $Y \in Mod_{A^c}(P)$.

Next, we prove that $Mod_{A^c}(P) \neq Mod_{A^c}(Q)$ or, for some $Y \in Mod_{A^c}(P)$, $T_P(Y) \neq T_Q(Y)$ if and only if there is $Y' \subseteq At(P \cup Q) \cup A$ such that Y' belongs to exactly one of $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$, or Y' belongs to both $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$ and $T_P(Y') \neq T_Q(Y')$. Clearly, we need to prove the "only-if" implication. To this end, we note that if $Mod_{A^c}(P) \neq Mod_{A^c}(Q)$, then by the observation proved above, there is $Y' \subseteq At(P \cup Q) \cup A$ with that property. Thus, assume that $Mod_{A^c}(P) = Mod_{A^c}(Q)$. If for some $Y \in Mod_{A^c}(P)$, $T_P(Y) \neq T_Q(Y)$ then, Y belongs to both $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$. By the argument given above, $Y' = Y \cap (At(P \cup Q) \cup A)$ belongs to both $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$, and $T_P(Y') \neq T_Q(Y')$.

Thus, to decide the complementary problem, we nondeterministically guess $Y \subseteq At(P \cup Q) \cup A$, and verify that Y belongs to exactly one of $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$, or that Y belongs to $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$, and that $T_P(Y) \neq T_Q(Y)$.

Checking $Y \models P$ and $Y \models Q$ can be done in polynomial time. Similarly, for $R = P$ or Q, $Y \setminus T_R(Y) \subseteq A^c$ if and only if $(Y \setminus T_R(Y)) \cap A = \emptyset$. Thus, checking $Y \setminus T_R(Y) \subseteq A^c$ can be done in polynomial time, too, and so the algorithm is polynomial. Hence, the complementary problem is in NP, which implies the assertion. □

For the lower bound we use the problem $\mathrm{SUPP}_{c,d}(A,B)$.

Theorem 5. *The problem* $\text{SUPP}_{c,d}(A, B)$ *is coNP-hard.*

Proof: Let us consider a CNF φ, let Y be the set of atoms in φ, and let $Y' = \{y' \mid y \in Y\}$ be a set of new atoms. We define

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

where, for each clause $c \in \varphi$, say $c = y_1 \vee \cdots \vee y_k \vee \neg y_{k+1} \vee \cdots \vee \neg y_m$, $\hat{c}$ denotes the the sequence $y'_1, \ldots, y'_k, y_{k+1}, \ldots, y_m$. 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 *supported* model of P and, consequently, satisfies $M = T_P(M)$.

Next, let Q consist of f and $\leftarrow f$. As Q has no models, Theorem 1 implies that Q is supp-equivalent to P relative to $\mathcal{HB}(A^c, B)$ if and only if $Mod_{A^c}(P) = \emptyset$. If $M \in Mod_{A^c}(P)$, then there is $M' \subseteq At(P)$ such that $M' \in Mod_{A^c}(P)$. Since every model M' of P such that $M' \subseteq At(P)$ satisfies $M' = T_P(M')$, it follows that $Mod_{A^c}(P) = \emptyset$ if and only if P has no models. Thus, φ is unsatisfiable if and only if Q is supp-equivalent to P relative to $\mathcal{HB}(A^c, B)$, and the assertion follows. □

We combine Theorems 4 and 5 via the relations depicted in Figure 1 and obtain:

Corollary 1. *The problem* $\text{SUPP}_{\delta,\varepsilon}(\alpha, \beta)$ *is coNP-complete, for any combination of* $\delta, \varepsilon \in \{c, d\}$, $\alpha \in \{A, \cdot\}$, $\beta \in \{B, \cdot\}$.

5 Suppmin-Equivalence

In this section, we establish the complexity for direct-complement, complement-direct and complement-complement problems of deciding suppmin-equivalence. The complexity of direct-direct problems was determined in [24].

Upper bounds. The argument consists of a series of auxiliary results. Due to space restrictions, we omit some of the proofs. The first two lemmas are concerned with the basic problem of deciding whether $(X, Y) \in Mod_{A'}^{B'}(P)$, where A' and B' stand for A or A^c and B or B^c, respectively.

Lemma 1. *The following problems are in the class coNP: Given a program P, and sets X, Y, A, and B, decide whether (i)* $(X, Y) \in Mod_{A^c}^{B}(P)$*; (ii)* $(X, Y) \in Mod_{A}^{B^c}(P)$*; (iii)* $(X, Y) \in Mod_{A^c}^{B^c}(P)$.

Proof: We first show that the complementary problem to decide whether $(X, Y) \notin Mod_{A^c}^{B}(P)$ is in NP. To this end, we observe that $(X, Y) \notin Mod_{A^c}^{B}(P)$ if and only if at least one of the following conditions holds: (1) $Y \notin Mod_{A^c}(P)$, (2) $X \not\subseteq Y|_{A^c \cup B}$ (3) there is $Z \subset Y$ such that $Z|_{A^c \cup B} = Y|_{A^c \cup B}$ and $Z \models P$, (4) there is $Z \subset Y$ such that $Z|_B = X|_B$, $Z|_{A^c} \supseteq X|_{A^c}$ and $Z \models P$, (5) $X|_B = Y|_B$ and $Y \setminus T_P(Y) \not\subseteq X$. We note that verifying any condition involving A^c can be reformulated in terms of A. For instance, for every set V, we have $V|_{A^c} = V \setminus A$, and $V \subseteq A^c$ if and only if $V \cap A = \emptyset$. Thus, the conditions (1), (2) and (5) can be decided in polynomial time. Conditions (3) and (4) can be decided by a nondeterministic polynomial time algorithm. Indeed, once we nondeterministically guess Z, all other tests can be decided in polynomial time. The proofs for the remaining two claims use the same ideas and differ only in technical details depending on which of A and B is subject to the complement operation. □

Lemma 2. *For every finite set $B \subseteq At$, the following problems are in the class Pol: given a program P, and sets X, Y and A, decide whether (i) $(X, Y) \in Mod_{A^c}^{B^c}(P)$; (ii) $(X, Y) \in Mod_{A}^{B^c}(P)$.*

Proof: In each case, the argument follows the same lines as that for Lemma 1. The difference is in the case of the conditions (3) and (4). Under the assumptions of this lemma, they can be decided in *deterministic* polynomial time. Indeed, let us note that there are no more than $2^{|B|}$ sets Z such that $Z|_{A^c \cup B^c} = Y|_{A^c \cup B^c}$ (or, for the second problem, such that $Z|_{A \cup B^c} = Y|_{A \cup B^c}$). Since B is finite and fixed, the condition (3) can be checked in polynomial time by a simple enumeration of all possible sets Z such that $Z \subset Y$ and $Z|_{A^c \cup B^c} = Y|_{A^c \cup B^c}$ and checking for each of them whether $Z \models P$. For the condition (4), the argument is similar. Since Z is constrained by $Z|_{B^c} = X|_{B^c}$, there are no more than $2^{|B|}$ possible candidate sets Z to consider in this case, too. □

The role of the next lemma is to show that $(X, Y) \in Mod_A^B(P)$ implies constraints on X and Y.

Lemma 3. *Let P be a program and $A, B \subseteq At$. If $(X, Y) \in Mod_A^B(P)$ then $X \subseteq Y \subseteq At(P) \cup A$.*

Lemma 3 is too weak for the membership results for complement-direct and complement-complement problems, as for these two types of problems, it only limits Y to subsets of $At(P) \cup A^c$, which is infinite. To handle these two classes of problems we use the following lemma that can be derived from Theorem 2.

Lemma 4. *Let P, Q be programs and $A, B \subseteq At$. If $(X, Y) \in Mod_{A^c}^B(P) \setminus Mod_{A^c}^B(Q)$ then there is $(X', Y') \in Mod_{A^c}^B(P) \setminus Mod_{A^c}^B(Q)$ such that $Y' \subseteq At(P \cup Q) \cup A$. If $(X, Y) \in Mod_{A^c}^B(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$, then there is $(X', Y') \in Mod_{A^c}^B(P)$ such that $T_P(Y')|_B \neq T_Q(Y')|_B$ and $Y' \subseteq At(P \cup Q) \cup A$.*

Theorem 6. *The following problems are contained in the class Π_2^P: $\text{SUPPMIN}_{c,d}(\cdot, \cdot)$, $\text{SUPPMIN}_{c,c}(\cdot, \cdot)$ and $\text{SUPPMIN}_{d,c}(\cdot, \cdot)$. The following problems are in the class coNP: $\text{SUPPMIN}_{d,c}(\cdot, B)$, $\text{SUPPMIN}_{c,c}(\cdot, B)$, $\text{SUPPMIN}_{c,c}(\emptyset, \cdot)$ and $\text{SUPPMIN}_{c,d}(\emptyset, \cdot)$.*

Proof: We provide a detailed argument for the problem $\text{SUPPMIN}_{c,d}(\cdot, \cdot)$. Clearly, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(A^c, B)$ if and only if there is $(X, Y) \in Mod_{A^c}^B(P) \div Mod_{A^c}^B(Q)$, or $(X, Y) \in Mod_{A^c}^B(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$. By Lemma 4, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(A^c, B)$ if and only if there is (X, Y) such that $X \subseteq Y \subseteq At(P \cup Q) \cup A$ and $(X, Y) \in Mod_{A^c}^B(P) \div Mod_{A^c}^B(Q)$, or $(X, Y) \in Mod_{A^c}^B(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$.

Thus, to decide the complementary problem, it suffices to guess $X, Y \subseteq At(P \cup Q) \cup A$ and check that $(X, Y) \in Mod_{A^c}^B(P) \div Mod_{A^c}^B(Q)$, or that (X, Y) is in both sets and $T_P(Y)|_B \neq T_Q(Y)|_B$. The first task can be decided by NP oracles (Lemma 1(i)) and testing $T_P(Y)|_B \neq T_Q(Y)|_B$ can be accomplished in polynomial time.

The remaining arguments are similar. The only differences are: For $\textsc{suppmin}_{d,c}(\cdot,\cdot)$ and $\textsc{suppmin}_{d,c}(\cdot,B)$ we use Lemma 3 to ensure that the decision algorithm can restrict in the guessing phase to pairs (X,Y) with $Y \subseteq At(P \cup Q) \cup A$; for $\textsc{suppmin}_{d,c}(\cdot,\cdot)$ and $\textsc{suppmin}_{c,c}(\cdot,\cdot)$, we use Lemma 1(ii)-(iii); to obtain a stronger upper bound for $\textsc{suppmin}_{d,c}(\cdot,B)$ and $\textsc{suppmin}_{c,c}(\cdot,B)$, we make use of Lemma 2. The result for $\textsc{suppmin}_{c,d}(\emptyset,\cdot)$ was settled in [24] (although not directly, the case of $\textsc{suppmin}_{c,c}(\emptyset,\cdot)$ follows also from [24]; we provide details in the full version). For problems involving B^c, we test $T_P(Y)|_{B^c} = T_Q(Y)|_{B^c}$ by comparing $T_P(Y) \setminus B$ and $T_Q(Y) \setminus B$. □

Suppmin-equivalence — lower bounds and exact complexity results. To illustrate methods we use to obtain our results, we will provide full details for the case of direct-complement problems. For the other two types of problems, we only state the results.

Theorem 7. *The problem* $\textsc{suppmin}_{d,c}(A,\cdot)$ *is* Π_2^P*-hard.*

Proof: Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$. We can assume that $A \cap X = \emptyset$ (if not, variables in X can be renamed). Next, we can assume that $A \subseteq Y$ (if not, add "dummy" tautology clauses to φ). We will construct programs $P(\varphi)$ and $Q(\varphi)$, and a set B, so that $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are suppmin-equivalent relative to $\mathcal{HB}(A, B^c)$. 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.

For every atom $z \in X \cup Y$, we introduce a fresh atom z'. Given a set of "non-primed" atoms Z, we define $Z' = \{z' \mid z \in Z\}$. In particular, $A \cap (Y' \cup X') = \emptyset$. We use $\hat{c}$ as in the proof of Theorem 5 and define the following programs:

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

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

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

1. $I \cup (Y \setminus I)' \cup J \cup (X \setminus J)'$, 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 B, viz.

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

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

From these comments, it follows that for every model M of Q (resp. P), $T_Q(M) = M \cap B$ (resp. $T_P(M) = M \cap B$). Thus, for every model M of both P and Q, $T_Q(M)|_{B^c} = T_P(M)|_{B^c}$. It follows that P and Q are suppmin-equivalent with respect to $\mathcal{HB}(A, B^c)$ if and only if $\mathit{Mod}_A^{B^c}(P) = \mathit{Mod}_A^{B^c}(Q)$ (indeed, we recall that if $(N, M) \in \mathit{Mod}_A^{B^c}(R)$ 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 $(N|_{A \cup B^c}, N) \in Mod_A^{B^c}(P)$.

Since N is a supported model of P, $N \in Mod_A(P)$. The requirement (2) for $(N|_{A \cup B^c}, N) \in Mod_A^{B^c}(P)$ 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 $(N|_{A \cup B^c}, N) \in Mod_A^{B^c}(P)$, as claimed. Since N is not a model of Q, $(N|_{A \cup B^c}, N) \notin Mod_A^{B^c}(Q)$.

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

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 $(M, N) \in Mod_A^{B^c}(P)$. If $N = N' \cup V$, where N' is of type 1 and $V \subseteq At \setminus B$, then arguing as above, one can show that $(M, N) \in Mod_A^{B^c}(Q)$. Therefore, let us assume that $N = N' \cup V$, where N' is of type 2 and $V \subseteq At \setminus B$. 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 $Z, N \subseteq B$, we have $Z|_{A \cup B^c} = N|_{A \cup B^c}$. Since $(M, N) \in Mod_A^{B^c}(P)$, the requirement (3) implies that Z is not a model of P, a contradiction. Hence, the latter case is impossible and $Mod_A^{B^c}(P) \subseteq Mod_A^{B^c}(Q)$ follows.

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

Theorem 8. *The problem* SUPPMIN$_{d,c}(A, B)$ *is coNP-hard.*

Proof: Consider a CNF φ over atoms Y, and the programs $P(\varphi)$ and $Q = \{f \leftarrow;\ \leftarrow f\}$ from the proof of Theorem 5. We use P for $P(\varphi)$ in the following. We already know that P has a model if and only if φ is true. We now show that $Mod_A^{B^c}(P) \neq \emptyset$ if and only if φ is true. Since $Mod_A^{B^c}(Q) = \emptyset$ holds (as is easily seen), the assertion follows by Theorem 2.

Let us assume that P has a model. Then P has a model, say M, such that $M \subseteq Y \cup Y'$. We show that $(M, M) \in Mod_A^{B^c}(P)$. Indeed, since $T_P(M) = M$, $M \in Mod_A(P)$. Also, since $Y \cup Y' \subseteq B^c$, $M|_{A \cup B^c} = M$ and so, $M \subseteq M|_{A \cup B^c}$. Lastly, $M \setminus T_P(M) = \emptyset \subseteq M$. Thus, the conditions (1), (2) and (5) for $(M, M) \in Mod_A^{B^c}(P)$ hold. Since $M|_{A \cup B^c} = M$ and $M|_{B^c} = M$, there is no $Z \subset M$ such that $Z|_{A \cup B^c} = M|_{A \cup B^c}$ or $Z|_{B^c} = M|_{B^c}$. Thus, also conditions (3) and (4) hold, and $Mod_A^{B^c}(P) \neq \emptyset$ follows. Conversely, let $Mod_A^{B^c}(P) \neq \emptyset$ and let $(N, M) \in Mod_A^{B^c}(P)$. Then $M \in Mod_A(P)$ and, in particular, M is a model of P. □

Combining Theorems 7 and 8 with Theorem 6 yields the following result that fully determines the complexity of direct-complement problems.

Corollary 2. *The problems* $\text{SUPPMIN}_{d,c}(A, \cdot)$ *and* $\text{SUPPMIN}_{d,c}(\cdot, \cdot)$ *are* Π_2^P*-complete. The problems* $\text{SUPPMIN}_{d,c}(A, B)$ *and* $\text{SUPPMIN}_{d,c}(\cdot, B)$ *are coNP-complete.*

This concludes the more detailed discussion on the direct-complement problems. Next, we just give the corresponding results for the remaining settings we have to study for suppmin-equivalence, complement-complement and complement-direct problems.

Theorem 9. *With* $A \neq \emptyset$*,* $\text{SUPPMIN}_{c,c}(A, \cdot)$ *and* $\text{SUPPMIN}_{c,d}(A, B)$ *are* Π_2^P*-hard. The problems* $\text{SUPPMIN}_{c,c}(\emptyset, \cdot)$ *and* $\text{SUPPMIN}_{c,c}(A, B)$ *are coNP-hard.*

Combining Theorem 9 with Theorem 6 yields the following corollary completing the picture of the complexity for suppmin-equivalence. The coNP-completeness results for the complement-direct problems were already proved in [24].

Corollary 3. *The problems* $\text{SUPPMIN}_{c,c}(\cdot, \cdot)$*,* $\text{SUPPMIN}_{c,d}(\cdot, B)$ *and* $\text{SUPPMIN}_{c,d}(\cdot, \cdot)$ *are* Π_2^P*-complete. For* $A \neq \emptyset$*, also the problems* $\text{SUPPMIN}_{c,c}(A, \cdot)$*,* $\text{SUPPMIN}_{c,d}(A, B)$ *and* $\text{SUPPMIN}_{c,d}(A, \cdot)$*, are* Π_2^P*-complete. Moreover, the following problems are coNP-complete:* $\text{SUPPMIN}_{c,c}(\emptyset, \cdot)$*,* $\text{SUPPMIN}_{c,c}(A, B)$*,* $\text{SUPPMIN}_{c,c}(\cdot, B)$*,* $\text{SUPPMIN}_{c,d}(\emptyset, \cdot)$ *and* $\text{SUPPMIN}_{c,d}(\emptyset, B)$*.*

6 Stable-Equivalence

We turn now to stable-equivalence. Here we also consider direct-direct problems as, in the case of fixed alphabets, they were not considered in [20].

Upper bounds. The following lemmas mirror the respective results from the previous section but show some interesting differences.

Lemma 5. *The following problems are in the class* D^P *in general*[5] *and in the class Pol for normal programs: Given a program* P*, and sets* X*,* Y*,* A*, and* B*, decide whether (i)* $(X, Y) \in SE_A^B(P)$*; (ii)* $(X, Y) \in SE_{A^c}^B(P)$*; (iii)* $(X, Y) \in SE_A^{B^c}(P)$*; (iv)* $(X, Y) \in SE_{A^c}^{B^c}(P)$*.*

Lemma 6. *For every finite sets* $A, B \subseteq At$*, the following problem is in the class Pol: given a program* P*, and sets* X*,* Y *decide whether* $(X, Y) \in SE_{A^c}^{B^c}(P)$*.*

Hence, polynomial-time model-checking for *disjunctive* programs is only possible for the set $SE_{A^c}^{B^c}(P)$. Compared to Lemma 2, this is due to the more involved condition (4) for $SE_{A^c}^{B^c}(P)$. For *normal* programs the reduct P^Y is a Horn program, which is essential for the tractability results in Lemma 5.

The following lemmas hold for both disjunctive and normal programs.

Lemma 7. *Let* P *be a program and* $A, B \subseteq At$*. If* $(X, Y) \in SE_A^B(P)$ *then* $X \subseteq Y \subseteq At(P) \cup A$*.*

[5] The class D^P consists of all problems expressible as the conjunction of a problem in NP and a problem in coNP. However, this slight increase of complexity compared to Lemma 1 does not influence the subsequent Π_2^P-membership results, since a D^P-oracle amounts to an NP-oracle.

Lemma 8. *Let P, Q be programs and $A, B \subseteq At$. If $(X, Y) \in SE^B_{A^c}(P) \setminus SE^B_{A^c}(Q)$ then there is $(X', Y') \in SE^B_{A^c}(P) \setminus SE^B_{A^c}(Q)$ such that $Y' \subseteq At(P \cup Q) \cup A$.*

We can now use the similar arguments in the previous section to obtain the following collection of membership results:

Theorem 10. *The problem* $\text{STABLE}_{\delta,\varepsilon}(\cdot,\cdot)$*, is contained in the class Π^P_2, for any $\delta, \varepsilon \in \{c, d\}$;* $\text{STABLE}_{c,c}(A, B)$ *is contained in the class coNP. The problem* $\text{STABLE}^n_{\delta,\varepsilon}(\cdot,\cdot)$*, is contained in the class coNP for any $\delta, \varepsilon \in \{c, d\}$.*

Stable-equivalence — lower bounds and exact complexity results. We start with hardness for normal programs.

Theorem 11. *The problem* $\text{STABLE}^n_{\delta,\varepsilon}(A, B)$ *is coNP-hard for any $\delta, \varepsilon \in \{c, d\}$.*

Proof sketch: We use the standard reduction of UNSAT, thus let $P(\varphi)$ and Q be as in the proof of Theorem 5. It can be shown that $P(\varphi)$ has a stable model iff φ is satisfiable. Moreover, $P(\varphi) \cup R$ has no stable model (for arbitrary R) iff φ is not satisfiable. On the other hand, $Q \cup R$ has no stable model, for any R. Thus P is stable equivalent to Q relative to $\mathcal{C}$ iff φ is unsatisfiable, where $\mathcal{HB}(\emptyset, \emptyset) \subseteq \mathcal{C} \subseteq \mathcal{HB}(At, At)$, and thus where $\mathcal{C}$ is an arbitrary class. Hence, the result holds in particular for the desired classes. □

We now turn to the case of disjunctive programs. We note that coNP-hardness for $\text{STABLE}_{c,c}(A, B)$ follows immediately from the previous result. The remaining hardness results can be shown by suitable adaptations of the reductions used in [17].

Theorem 12. *The following problems are hard for the class Π^P_2:* $\text{STABLE}_{d,d}(A, B)$, $\text{STABLE}_{c,d}(A, B)$, $\text{STABLE}_{d,c}(A, B)$, $\text{STABLE}_{c,c}(A, \cdot)$, *and* $\text{STABLE}_{c,c}(\cdot, B)$.

Combining Theorems 11 and 12 with Theorem 10 yields the following corollary for the complete picture of the complexity for stable-equivalence.

Corollary 4. *The following problems are Π^P_2-complete for any combination of $\delta, \varepsilon \in \{c, d\}$:* $\text{STABLE}_{\delta,\varepsilon}(\cdot,\cdot)$, $\text{STABLE}_{\delta,\varepsilon}(A, \cdot)$, $\text{STABLE}_{\delta,\varepsilon}(\cdot, B)$. *As well,* $\text{STABLE}_{d,d}(A, B)$, $\text{STABLE}_{c,d}(A, B)$ *and* $\text{STABLE}_{d,c}(A, B)$ *are Π^P_2-complete, while* $\text{STABLE}_{c,c}(A, B)$ *is coNP-complete. The problem* $\text{STABLE}^n_{\delta,\varepsilon}(\alpha, \beta)$ *is coNP-complete, for any combination of $\delta, \varepsilon \in \{c, d\}$, $\alpha \in \{A, \cdot\}$, $\beta \in \{B, \cdot\}$.*

7 Discussion

We studied the complexity of deciding relativized hyperequivalence of programs under the semantics of stable, supported and supported minimal models. We focused on problems $\text{SEM}_{\delta,\epsilon}(\alpha, \beta)$, where at least one of δ and ϵ equals c, that is, at least one of the alphabets for the context problems is determined as the complement of the corresponding set A or B. As we noted, such problems arise naturally in the context of modular design of logic programs, yet they have received essentially no attention so far.

Table 1 summarizes the results. It shows that the problems concerning supp-equivalence (no normality restriction), and stable-equivalence for normal programs are all

Table 1. Complexity of $\textsc{sem}_{\delta,\varepsilon}(\alpha,\beta)$; all entries are completeness results

δ	ε	α	β	SUPP	SUPPMIN	STABLE	STABLE^n
d	c		$\cdot$	coNP	Π_2^P	Π_2^P	coNP
d	c		B	coNP	coNP	Π_2^P	coNP
c	c	$\cdot$ or $A \neq \emptyset$	$\cdot$	coNP	Π_2^P	Π_2^P	coNP
c	c	$\emptyset$	$\cdot$	coNP	coNP	Π_2^P	coNP
c	c	$\cdot$	B	coNP	coNP	Π_2^P	coNP
c	c	A	B	coNP	coNP	coNP	coNP
c	d	$\cdot$ or $A \neq \emptyset$		coNP	Π_2^P	Π_2^P	coNP
c	d	$\emptyset$		coNP	coNP	Π_2^P	coNP

coNP-complete (as are the corresponding direct-direct problems, studied in [24] and here). The situation is more diversified for suppmin-equivalence and stable-equivalence (no normality restriction) with some problems being coNP- and others Π_2^P-complete. For suppmin-equivalence lower complexity requires that B be a part of problem specification, or that A be a part of problem specification and be set to $\emptyset$. For stable-equivalence, the lower complexity only holds for the complement-complement problem with both A and B fixed as part of the problem specification. We also note that the complexity of problems for stable-equivalence is always at least that for suppmin-equivalence. Furthermore, our complexity results suggest possible algorithms for testing the equivalence notions under consideration. One such approach is to reduce the given characterizations to quantified Boolean formulas (QBFs) along the lines of previous work, e.g. [22], and then use extant solvers for QBFs to decide equivalence.

There are several questions worthy of further investigations. For instance, while stable-equivalence when only parts of models are compared was studied [21,22], no similar results are available for supp- and suppmin-equivalence. Also the complexity of the corresponding complement-direct, direct-complement and complement-complement problems for the three semantics in that setting has yet to be established.

Acknowledgments

We acknowledge support from the NSF (grant IIS-0325063), the KSEF (grant KSEF-1036-RDE-008), and the Austrian Science Fund (grants P18019-N04, P20704-N18).

References

1. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K., Marek, W., Truszczyński, M., Warren, D. (eds.) The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398. Springer, Berlin (1999)
2. Niemelä, I.: Logic programming with stable model semantics as a constraint programming paradigm. Annals of Mathematics and Artificial Intelligence 25, 241–273 (1999)
3. Gelfond, M., Leone, N.: Logic programming and knowledge representation – the A-prolog perspective. Artificial Intelligence 138, 3–38 (2002)
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)

5. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
6. Gelfond, M.: Representing knowledge in A-Prolog. In: Kakas, A., Sadri, F. (eds.) Computational Logic: Logic Programming and Beyond. LNCS, vol. 2408, pp. 413–451. Springer, Heidelberg (2002)
7. Janhunen, T.: Some (in)translatability results for normal logic programs and propositional theories. Journal of Applied Non-Classical Logics 16, 35–86 (2006)
8. Oikarinen, E., Janhunen, T.: Modular equivalence for normal logic programs. In: Proceedings of ECAI 2006, pp. 412–416. IOS Press, Amsterdam (2006)
9. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity aspects of disjunctive stable models. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 175–187. Springer, Heidelberg (2007)
10. Gaifman, H., Shapiro, E.: Fully abstract compositional semantics for logic programs. In: Proceedings of POPL 1989, pp. 134–142 (1989)
11. Lifschitz, V., Turner, H.: Splitting a logic program. In: Proceedings of ICLP 1994, pp. 23–37 (1994)
12. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic 2(4), 526–541 (2001)
13. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Proceedings of KR 2002, pp. 170–176. Morgan Kaufmann, San Francisco (2002)
14. Turner, H.: Strong equivalence made easy: nested expressions and weight constraints. Theory and Practice of Logic Programming 3, 609–622 (2003)
15. de Jongh, D., Hendriks, L.: Characterizations of strongly equivalent logic programs in intermediate logics. Theory and Practice of Logic Programming 3, 259–270 (2003)
16. Eiter, T., Fink, M.: Uniform equivalence of logic programs under the stable model semantics. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 224–238. Springer, Heidelberg (2003)
17. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Transactions on Computational Logic 8, 53 (2007)
18. Sagiv, Y.: Optimizing datalog programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 659–698. Morgan Kaufmann, San Francisco (1988)
19. Inoue, K., Sakama, C.: Equivalence of logic programs under updates. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 174–186. Springer, Heidelberg (2004)
20. Woltran, S.: A common view on strong, uniform, and other notions of equivalence in answer-set programming. Theory and Practice of Logic Programming 8, 217–234 (2008)
21. Eiter, T., Tompits, H., Woltran, S.: On solution correspondences in answer-set programming. In: Proceedings of IJCAI 2005, pp. 97–102. Morgan Kaufmann, San Francisco (2005)
22. Oetsch, J., Tompits, H., Woltran, S.: Facts do not cease to exist because they are ignored: Relativised uniform equivalence with answer-set projection. In: Proceedings of AAAI 2007, pp. 458–464. AAAI Press, Menlo Park (2007)
23. Cabalar, P., Odintsov, S., Pearce, D., Valverde, A.: Analysing and extending well-founded and partial stable semantics using partial equilibrium logic. In: Etalle, S., Truszczyński, M. (eds.) ICLP 2006. LNCS, vol. 4079, pp. 346–360. Springer, Heidelberg (2006)
24. Truszczyński, M., Woltran, S.: Hyperequivalence of logic programs with respect to supported models. In: Proceedings of AAAI 2008, pp. 560–565. AAAI Press, Menlo Park (2008)
25. Clark, K.: Negation as failure. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 293–322. Plenum Press, New York (1978)
26. Erdogan, S., Lifschitz, V.: Definitions in answer set programming. In: LPNMR 2004. LNCS (LNAI), vol. 2923, pp. 114–126. Springer, Heidelberg (2004)
27. Apt, K.: Logic programming. In: van Leeuven, J. (ed.) Handbook of Theoretical Computer Science, pp. 493–574. Elsevier, Amsterdam (1990)