Relativized Hyperequivalence of Logic Programs for Modular Programming

MIROSŁAW TRUSZCZYŃSKI

Department of Computer Science, University of Kentucky, Lexington, KY 40506-0046, USA
(*e-mail:* mirek@cs.uky.edu)

STEFAN WOLTRAN

Institute for Information Systems 184/2, Technische Universität Wien, Favoritenstrasse 9-11, 1040 Vienna, Austria
(*e-mail:* woltran@dbai.tuwien.ac.at)

submitted 21 January 2009; revised 24 June 2009; accepted 21 July 2009

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 with respect to the three other types of context programs.

KEYWORDS: answer-set programming, strong equivalence, uniform equivalence, relativized equivalence, stable models, supported models, minimal models, complexity

1 Introduction

We study variants of relativized hyperequivalence that are relevant for the development and analysis of disjunctive 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 constraint problems (Marek and Truszczyński 1999; Niemelä 1999; Gelfond and Leone 2002; Baral 2003). 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 (Gebser et al. 2007).

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* (Gelfond 2002; Janhunen 2006; Oikarinen and Janhunen 2006; Janhunen et al. 2007). This work builds on ideas for compositional semantics of logic programs proposed by Gaifman and Shapiro (1989) and encompasses earlier results on stratification and *program splitting* (Lifschitz and Turner 1994).

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* with respect to 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 with respect to $\mathcal{C}$. If $\mathcal{C}$ contains the empty program (which is typically the case and, in particular, is the case for the families of programs we consider in the paper), the equivalence for substitution with respect to $\mathcal{C}$ 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 with respect to the class of *all* programs, known more commonly as *strong equivalence*, was proposed and studied by Lifschitz et al. (2001). That work prompted extensive investigations of the concept that resulted in new characterizations (Lin 2002; Turner 2003) and connections to certain non-standard logics (de Jongh and Hendriks 2003). Hyperequivalence with respect to contexts consisting of facts was studied by Eiter and Fink (2003). This version of hyperequivalence, known as *uniform equivalence*, appeared first in the database area in the setting of DATALOG and query equivalence (Sagiv 1988). Hyperequivalence with respect to contexts restricted to a given alphabet, or *relativized* hyperequivalence, was proposed by Woltran (2004) and Inoue and Sakama (2004). Both uniform equivalence and relativized hyperequivalence were analyzed in depth by Eiter et al. (2007), and later generalized by Woltran (2008) to allow contexts that use (possibly) different alphabets for the heads and bodies of rules. That 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 has also received some attention (Eiter et al. 2005; Oetsch et al. 2007).

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

All those results concern the stable-model semantics of programs. There has been little work on other semantics, with the work by Cabalar et al. (2006) long being a notable single exception. Recently however, Truszczyński and Woltran (2008) introduced and investigated relativized hyperequivalence of programs under the semantics of supported models (Clark 1978) and supported minimal models, two other major semantics of logic programs. Truszczyński and Woltran (2008) 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 earlier works (Woltran 2008; Truszczyński and Woltran 2008), 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 atoms 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) with respect to $\mathcal{HB}(A, B)$. 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

We postpone technical preliminaries to the following section. For the sake of the present section it is enough to say that we focus our study on finite propositional programs over a fixed countable infinite set At of atoms. It is also necessary to introduce one piece of notation: $X^c = At \setminus X$.

To argue that contexts specified in terms of the complement of a finite set are of interest, let us consider the following scenario. 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 (such atoms facilitate importing information form other modules, hence the term "interface"), as well as atoms used locally to represent some needed auxiliary concepts. Let P be a particular A-defining program with L as the set of its local atoms. 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 appropriate in our setting is that of hyperequivalence

[2] A-defining programs were introduced by Erdogan and Lifschitz (2004). However, that work considered more restricted classes of programs with which A-defining programs could be combined.

with respect to $\mathcal{HB}((A \cup L)^c, L^c)$ 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 the conditions (1) - (4) is by means of the following $\{a, b\}$-defining program P (in this example we assume that $\{a, b\}$-defining programs do not use local atoms, that is, $L = \emptyset$):

$a \leftarrow c, not\ b;$
$b \leftarrow d, not\ a.$

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 $R = \{c \leftarrow not\ d;\ d \leftarrow a, not\ c\}$. Here, d can only be true if a is true and c is false, which is impossible given the way a is defined. 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: $\{a, c, d\}$. 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:

$a \leftarrow c, d, not\ b;$
$b \leftarrow c, d, not\ a;$
$a \leftarrow c, not\ d;$
$b \leftarrow d, not\ c.$

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 gives rise to a different class of contexts. We call a program P *A-completing* if it completes partial and non-recursive definitions of atoms in A given by other modules which, for instance, might specify the base conditions for a recursive definition of atoms in A. Any program with all atoms in the heads of rules in A can be regarded as an A-completing program. Assuming that P is an

A-completing program (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. However, atoms from A may occur in the heads of rules from R, which constitute a partial, non-recursive part of the definition of A, "completed" by P. Such programs R form precisely the class $\mathcal{HB}(L^c, (A \cup L)^c)$.

Finally, let us consider a situation where we are to express partial problem specifications as a logic program. In that program, we need to use concepts represented by atoms from some set A that are defined elsewhere in terms of concepts described by atoms from some set B. Here two programs P and Q expressing these partial specifications can serve as each other's substitute precisely when they are hyperequivalent with respect to the class of programs $\mathcal{HB}(A, B)$.

These examples demonstrate that hyperequivalence with respect to context classes $\mathcal{HB}(A, B)$, where A and B are either specified directly or in terms of their 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. We recall that we consider a fixed countable infinite set of propositional atoms At. *Disjunctive logic programs* (programs, for short) are finite sets of (program) *rules* — expressions of the form

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

where a_i, b_i and c_i are atoms in At, '$\vee$' 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 $H \leftarrow B^+, \mathit{not}\ B^-$, 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 $B^+, \mathit{not}\ B^-$, 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 $H(P) = \bigcup_{r \in P} H(r)$, $B^{\pm}(P) = \bigcup_{r \in P}(B^+(r) \cup B^-(r))$, and $At(P) = H(P) \cup B^{\pm}(P)$.

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. That is, $M \models B(r)$, if jointly $B^+(r) \subseteq M$ and $B^-(r) \cap M = \emptyset$; $M \models H(r)$, if $H(r) \cap M \neq \emptyset$; and $M \models r$, if $M \models B(r)$ implies $M \models H(r)$. 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 with respect to 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 (with respect to 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 called a *supported model* of P (Brass and Dix 1997; Inoue and Sakama 1998).[3] In addition, M is called a *supported minimal model* of P if it is a supported model of P and a minimal model of P. One can check that supported models of P are indeed models of P.

A stable model of a program is a supported model of the program and a minimal model of the program. Thus, a stable model of a program is a supported minimal model of the program. However, the converse does not hold in general. 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, $T_P(M) = \{H(r) \mid r \in P,\ M \models B(r)\}$. 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 $T_P(M) \subseteq M$ (which, according to our convention, is an abbreviation for: T_P is defined for M and $T_P(M) \subseteq M$). Similarly, M is a *supported* model of P if $T_P(M) = M$ (Apt 1990) (that is, if T_P is defined for M and $T_P(M) = M$).

For a rule $r = a_1 \vee \ldots \vee a_k \leftarrow B$, where $k \geq 2$, 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 $T_{sh(P)}(M) = M$.

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 $B^{\pm}(P) \subseteq B$. Clearly, $\emptyset \in \mathcal{HB}(A, B)$ holds, for arbitrary $A, B \subseteq At$. Thus, as we noted in the introduction, for each of the semantics and every sets A and B, the corresponding hyperequivalence implies the standard equivalence with respect to that semantics.

When studying supp- and suppmin-equivalence we will restrict ourselves to the case of normal programs. Indeed, disjunctive programs P and Q are supp-equivalent (suppmin-equivalent, respectively) with respect to $\mathcal{HB}(A, B)$ if and only if normal

[3] A set X is a *hitting* set for a family $\mathcal{F}$ of sets if for every $F \in \mathcal{F}$, $X \cap F \neq \emptyset$.

programs $sh(P)$ and $sh(Q)$ are supp-equivalent (suppmin-equivalent, respectively) with respect to $\mathcal{HB}(A, B)$ (Truszczyński and Woltran 2008). Thus, from now on whenever we consider supp- and suppmin-equivalence, we implicitly assume that programs under comparison are normal. In particular, we use that convention in the definition below and the subsequent theorem.

For supp-equivalence and suppmin-equivalence, we need the set $Mod_A(P)$, defined by Truszczyński and Woltran (2008). Given a program P, and a set $A \subseteq At$,

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

Truszczyński and Woltran (2008) explain that elements of $Mod_A(P)$ can be viewed as *candidates* for becoming supported models of an extension of P by some program $R \in \mathcal{HB}(A, B)$. Indeed, each such candidate interpretation Y has to be a classical model of P, as 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, as otherwise programs from $\mathcal{HB}(A, B)$ cannot close this gap. The set $Mod_A(P)$ is the key to the characterization of supp-equivalence.

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 $Mod_A^B(P)$ (Truszczyński and Woltran 2008), 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 by Woltran (2008). We define $SE_A^B(P)$ to consist of all pairs (X, Y), where $X, Y \subseteq At$, such that:[4]

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

[4] We use a slightly different presentation than the one given by Woltran (2008). It is equivalent to the original one.

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 $SE^B_A(P) = SE^B_A(Q)$.

Decision problems. We are interested in problems of deciding hyperequivalence relative to classes of programs of the form $\mathcal{HB}(A', B')$, 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 means of its finite *complement*. 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 context programs: either A or the complement A^c of A, depending on whether $\delta = d$ or c. The parameter A 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 with respect to $\mathcal{HB}(A, B^c)$. Similarly, $\text{STABLE}_{c,c}(\cdot, \cdot)$ denotes the following problem: given programs P and Q, and sets A and B, decide whether P and Q are stable-equivalent with respect to $\mathcal{HB}(A^c, B^c)$. 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 essential 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.[5] 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 earlier (Truszczyński and Woltran 2008), and their complexity was fully determined there. The complexity of problems $\text{STABLE}_{d,d}(\cdot, \cdot)$, was also established before (Woltran 2008). Problems similar to $\text{STABLE}_{c,c}(A, A)$ were already studied by Eiter et al. (2007). In this paper, we complete the results on

[5] As demonstrated by Woltran (2008), we can also restrict the programs used as contexts to normal ones, as that makes no difference.

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.

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 straightforward polynomial-time reductions from one to the other. We illustrate these relationships in Figure 1. Each arrow indicates that the "arrowtail" problem can be reduced in polynomial time to the "arrowhead" one. Consequently, if there is a path from a problem Π to the problem Π' in the diagram, Π' is at least as hard as Π and Π is at most as hard as Π'. We use this observation in proofs of all complexity results.

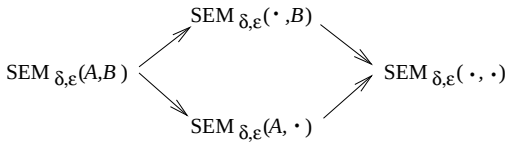

Fig. 1. A simple comparison of the hardness of problems

Finally, we note that throughout the paper, we write *Pol* instead of the more common P to denote the class of all problems that can be solved by deterministic polynomial-time algorithms. As decision problems we consider typically refer to a program P, we want to avoid the ambiguity of using the same symbol in two different meanings.

4 Supp-equivalence

As the alphabet for the bodies of context programs plays no role in supp-equivalence (cf. Theorem 1), the problems $\text{SUPP}_{d,c}(A,\beta)$ and $\text{SUPP}_{d,c}(\cdot,\beta)$ coincide with the problems $\text{SUPP}_{d,d}(A,\beta)$ and $\text{SUPP}_{d,d}(\cdot,\beta)$, respectively, whose complexity was shown to be coNP-complete (Truszczyński and Woltran 2008). For the same reason, problems $\text{SUPP}_{c,d}(A,\beta)$ and $\text{SUPP}_{c,d}(\cdot,\beta)$ coincide with $\text{SUPP}_{c,c}(A,\beta)$ and $\text{SUPP}_{c,c}(\cdot,\beta)$. Thus, to complete the complexity picture for problems $\text{SUPP}_{\delta,\epsilon}(\alpha,\beta)$, it suffices to focus on $\text{SUPP}_{c,d}(A,\beta)$ and $\text{SUPP}_{c,d}(\cdot,\beta)$.

First, we prove an upper bound on the complexity of the problem $\text{SUPP}_{c,d}(\cdot,\cdot)$. The proof depends on two lemmas.

Lemma 1
Let P be a program and A and Y sets of atoms. Then, $Y \in Mod_{A^c}(P)$ if and only if $Y' \in Mod_{A^c}(P)$, where $Y' = Y \cap (At(P) \cup A)$.

Proof
First, we note that atoms that do not occur in P have no effect on whether an interpretation satisfies the body of a rule in P. Thus, $T_P(Y) = T_P(Y')$. 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 $Y' \models P$ (as before, atoms that do not occur in P have no effect on whether an interpretation is a model of P or not). 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, $Y \models P$ (by the comment made above). 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, as $y \in Y$ and $Y' = Y \cap (At(P) \cup A)$, $y \notin A$, 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, $Y \in Mod_{A^c}(P)$. □

Lemma 2
Let P and Q be programs and A a set of atoms. Then, $Mod_{A^c}(P) \neq Mod_{A^c}(Q)$ 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 $T_P(Y') \neq T_Q(Y')$.

Proof
Clearly, we only 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 Lemma 1, there is $Y' \subseteq At(P \cup Q) \cup A$ with that property. Thus, let us 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 again 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 $T_P(Y') = T_P(Y) \neq T_Q(Y) = T_Q(Y')$. □

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

Proof
It is sufficient to show that $\text{SUPP}_{c,d}(\cdot,\emptyset)$ is in coNP, since (P, Q, A) is a YES instance of $\text{SUPP}_{c,d}(\cdot,\emptyset)$ if and only if (P, Q, A, B) is a YES instance of $\text{SUPP}_{c,d}(\cdot,\cdot)$ (cf. Theorem 1).

Thus, we will now focus on proving that $\text{SUPP}_{c,d}(\cdot,\emptyset)$ is in coNP. Theorem 1 and Lemma 2 imply the correctness of the following algorithm to decide the complementary problem to $\text{SUPP}_{c,d}(\cdot,\emptyset)$ for an instance (P, Q, A):

1. nondeterministically guess $Y \subseteq At(P \cup Q) \cup A$, and
2. 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 (in the size of the input, which is given by $|At(P \cup Q) \cup A|$). 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 runs in polynomial time. Hence, the complementary problem to $\text{SUPP}_{c,d}(\cdot, \emptyset)$ is in NP. It follows that the problem $\text{SUPP}_{c,d}(\cdot, \emptyset)$ is in coNP and so, the assertion follows. □

For the lower bound we use the problem $\text{SUPP}_{c,d}(A, B)$. Let us comment that the reduction, from the satisfiability problem to $\text{SUPP}_{c,d}(A, B)$, used in the following hardness proof, is indeed computable in polynomial time. The same is true for the reductions used in all other places in the paper. In each case, the polynomial-time computability of the reductions is evident, and we do not state it explicitly in the proofs.

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

Proof
Let us consider a CNF formula φ,[6] 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

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

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. □

The observations made at the beginning of this section, Theorems 4 and 5, and the relations depicted in Figure 1 imply the following corollary.

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 is already known (Truszczyński and Woltran 2008).

[6] Here and throughout the paper, by *CNF formula* we mean a formula in the conjunctive normal form.

5.1 Upper bounds

The argument consists of a series of auxiliary results. The first two lemmas are concerned with the basic problem of deciding whether $(X, Y) \in Mod^{B'}_{A'}(P)$, where A' and B' stand for A or A^c and B or B^c, respectively.

Lemma 3
The following problems are in the class coNP: Given a program P, and sets X, Y, A, and B, decide whether

i. $(X, Y) \in Mod^{B}_{A^c}(P)$;
ii. $(X, Y) \in Mod^{B^c}_{A}(P)$;
iii. $(X, Y) \in Mod^{B^c}_{A^c}(P)$.

Proof
We first show that the complementary problem, this is, to decide whether $(X, Y) \notin Mod^{B}_{A^c}(P)$, is in NP. To this end, we observe that $(X, Y) \notin Mod^{B}_{A^c}(P)$ 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 4
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. $(X, Y) \in Mod^{B^c}_{A^c}(P)$;
ii. $(X, Y) \in Mod^{B^c}_{A}(P)$.

Proof
In each case, the argument follows the same lines as that for Lemma 3. 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, fixed, and not a part of the input, 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 5
Let P be a program and $A, B \subseteq At$. If $(X, Y) \in Mod_A^B(P)$ then $X \subseteq Y \subseteq At(P) \cup A$.

Proof
We have $Y \in Mod_A(P)$. Thus, $Y \setminus T_P(Y) \subseteq A$ and, consequently, $Y \subseteq T_P(Y) \cup A \subseteq At(P) \cup A$. We also have $X \subseteq Y|_{A \cup B} \subseteq Y$. □

Theorem 6
The problem $\text{SUPPMIN}_{d,c}(\cdot, \cdot)$ is in the class Π_2^P. The problem $\text{SUPPMIN}_{d,c}(\cdot, B)$ is in the class coNP.

Proof
We start with an argument for the problem $\text{SUPPMIN}_{d,c}(\cdot, \cdot)$. By Theorem 2, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(A, B^c)$ if and only if there is $(X, Y) \in Mod_A^{B^c}(P) \div Mod_A^{B^c}(Q)$, or there is $(X, Y) \in Mod_A^{B^c}(P)$ and $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$. Thus, by Lemma 5, to decide that P and Q are not suppmin-equivalent relative to $\mathcal{HB}(A, B^c)$, one can guess X and Y such that $X \subseteq Y \subseteq At(P \cup Q) \cup A$ and verify that $(X, Y) \in Mod_A^{B^c}(P) \div Mod_A^{B^c}(Q)$, or that $(X, Y) \in Mod_A^{B^c}(P)$ and $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$. By Lemma 3(ii), deciding the membership of (X, Y) in $Mod_A^{B^c}(P)$ and $Mod_A^{B^c}(Q)$ can be accomplished by means of two calls to a coNP oracle. Deciding $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$ can be accomplished in polynomial time (we note that $T_P(Y)|_{B^c} = T_P(Y) \setminus B$ and $T_Q(Y)|_{B^c} = T_Q(Y) \setminus B$). The argument for the second part of the assertion is essentially the same. The only difference is that we use Lemma 4(ii) instead of Lemma 3(ii) to obtain a stronger bound. □

Lemma 5 is too weak for the membership results for complement-direct and complement-complement problems. Indeed, 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 results that provide stronger limits on Y and can be used in proofs of the membership results. The proofs are quite technical. To preserve the overall flow of the argument, we present them in the appendix.

Lemma 6
Let P, Q be programs and $A, B \subseteq At$.

1. 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 $Y' \subseteq At(P \cup Q) \cup A$.
2. 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 7
The problems $\text{SUPPMIN}_{c,d}(\cdot,\cdot)$ and $\text{SUPPMIN}_{c,c}(\cdot,\cdot)$ are contained in the class Π_2^P. The problem $\text{SUPPMIN}_{c,c}(\cdot, B)$ is in the class coNP.

Proof
The argument is similar to that of Theorem 6. First, we will consider the problem $\text{SUPPMIN}_{c,d}(\cdot,\cdot)$. By Theorem 2, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(A^c, B)$ if and only if there is $(X, Y) \in Mod^B_{A^c}(P) \div Mod^B_{A^c}(Q)$, or $(X, Y) \in Mod^B_{A^c}(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$. By Lemma 6, 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 $(X, Y) \in Mod^B_{A^c}(P) \div Mod^B_{A^c}(Q)$, or $(X, Y) \in Mod^B_{A^c}(P)$ 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 $(X, Y) \in Mod^B_{A^c}(P) \div Mod^B_{A^c}(Q)$, or that $(X, Y) \in Mod^B_{A^c}(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$. The first task can be decided by NP oracles (Lemma 3(i)), and testing $T_P(Y)|_B \neq T_Q(Y)|_B$ can be accomplished in polynomial time.

The remaining arguments are similar. To avoid repetitions, we only list essential differences. In the case of $\text{SUPPMIN}_{c,c}(\cdot,\cdot)$, we use Lemma 3(iii). To obtain a stronger upper bound for $\text{SUPPMIN}_{c,c}(\cdot, B)$, we use Lemma 4(i) instead of Lemma 3(iii). □

When A is fixed to $\emptyset$, that is, we have $A^c = At$, which means there is no restriction on atoms in the heads of rules, a stronger bound on the complexity of the complement-complement and complement-direct problems can be derived. We first state a key lemma (the proof is in the appendix).

Lemma 7
Let P, Q be programs and $B \subseteq At$. If $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$, then there is $Y \subseteq At(P \cup Q)$ 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 $Mod^B_{At}(Q)$.

Theorem 8
The problems $\text{SUPPMIN}_{c,c}(\emptyset,\cdot)$ and $\text{SUPPMIN}_{c,d}(\emptyset,\cdot)$ are in the class coNP.

Proof
The case of $\text{SUPPMIN}_{c,d}(\emptyset,\cdot)$ was settled before by Truszczyński and Woltran (2008) (they denoted the problem by SUPPMIN_{At}). Thus, we consider only the problem $\text{SUPPMIN}_{c,c}(\emptyset,\cdot)$. We will show that the following nondeterministic algorithm verifies, given programs P, Q and a set $B \subseteq At$, that P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$. We guess a pair (a, Y), where $Y \subseteq At(P \cup Q)$, and $a \in At(P \cup Q)$ such that (a) Y is a model of exactly one of P and Q; or (b) $a \in Y$ and $(Y \setminus \{a\}, Y)$ belongs to exactly one of $Mod^{B^c}_{At}(P)$ and $Mod^{B^c}_{At}(Q)$; or (c) Y is model of P and $T_P(Y) \setminus B \neq T_Q(Y) \setminus B$.

Such a pair exists if and only if P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$. Indeed, let us assume that such a pair (a, Y) exists. If (a) holds for (a, Y), say Y is a model of P but not of Q, then $(Y, Y) \in Mod^{B^c}_{At}(P) \setminus Mod^{B^c}_{At}(Q)$

(easy to verify from the definition of $Mod^{B^c}_{At}(\cdot)$). Thus, $Mod^{B^c}_{At}(P) \neq Mod^{B^c}_{At}(Q)$ and, by Theorem 2, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$. If (b) holds for (a, Y), $Mod^{B}_{At}(P) \neq Mod^{B}_{At}(Q)$ again, and we are done, as above, by Theorem 2. Finally, if (c) holds, $(Y, Y) \in Mod^{B^c}_{At}(P)$ (as $Y \models P$) and $T_P(Y)|_{B^c} = T_P(Y) \setminus B \neq T_Q(Y) \setminus B = T_Q(Y)|_{B^c}$. Thus, one more time by Theorem 2, P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$.

Conversely, if P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$, then $Mod^{B^c}_{At}(P) \neq Mod^{B^c}_{At}(Q)$, or there is $(X, Y) \in Mod^{B^c}_{At}(P)$ such that $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$. By Lemma 7, if $Mod^{B^c}_{At}(P) \neq Mod^{B^c}_{At}(Q)$ then there is (a, Y) such that $Y \subseteq At(P \cup Q)$ and (a, Y) satisfies (a) or (b). Thus, let us assume that there is $(X, Y) \in Mod^{B^c}_{At}(P)$ such that $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$. Then, $Y \models P$ and $T_P(Y)|_{B^c} \neq T_Q(Y)|_{B^c}$ or, equivalently, $T_P(Y) \setminus B \neq T_Q(Y) \setminus B$. Let $Y' = Y \cap At(P \cup Q)$. Clearly, $Y' \models P$, $T_P(Y) = T_P(Y')$, and $T_Q(Y) = T_Q(Y')$. Thus, $T_P(Y') \setminus B \neq T_Q(Y') \setminus B$. Picking any $a \in At(P \cup Q)$ (since P and Q are not suppmin-equivalent relative to $\mathcal{HB}(At, B^c)$, $At(P \cup Q) \neq \emptyset$) yields a pair (a, Y'), with $Y' \subseteq At(P \cup Q)$, for which (c) 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 $T_P(Y) \setminus B \neq T_Q(Y) \setminus B$. Finally, testing $(Y \setminus \{a\}, Y) \in Mod^{B^c}_{At}(P)$ and $(Y \setminus \{a\}, Y) \in Mod^{B^c}_{At}(Q)$ are polynomial-time tasks (with respect to the size of the input), too. The conditions (1) - (3) and (5) are evident. To verify the condition (4), we need to verify that $Z \not\models P$ for just one set Z, 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. □

5.2 Lower bounds and exact complexity results

We start with direct-complement problems.

Theorem 9
The problem $\text{SUPPMIN}_{d,c}(A, \cdot)$ is Π^P_2-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, one can add to φ "dummy" clauses $y \vee \neg y$, for $y \in Y$). 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^P_2-complete, the assertion will follow.

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

$$P(\varphi) \quad = \quad \{z \leftarrow not\ z';\ z' \leftarrow not\ z \mid z \in X \cup Y\} \cup \{\leftarrow y, y' \mid y \in Y\} \cup$$

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

To simplify notation, from now on we write P for $P(\varphi)$ and Q for $Q(\varphi)$. We 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 $M \subseteq B$, $T_P(M) = M$. Hence, each such model M is also supported for both P and Q.

From these comments, it follows that for every model M of Q (P, respectively), $T_Q(M) = M \cap B$ ($T_P(M) = M \cap B$, respectively). 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 $Mod_A^{B^c}(P) = Mod_A^{B^c}(Q)$ (indeed, we recall that if $(N, M) \in Mod_A^{B^c}(R)$ 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, i.e., $Mod_A^{B^c}(P) \subseteq Mod_A^{B^c}(Q)$. 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 $N' =$

$I \cup (Y \setminus I)' \cup X \cup X'$, for some $I \subseteq Y$. By our assumption, there is $J \subseteq X$ such that $I \cup J \models \varphi$. It follows that $Z = I \cup (Y \setminus I)' \cup J \cup (X \setminus J)' \cup V$ is a model of P. Clearly, $Z \subset N$. Moreover, since $B^c \cap (X \cup X' \cup Y \cup Y') = A \cap (X \cup X' \cup Y \cup Y') = \emptyset$, 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 10
The problem $\text{SUPPMIN}_{d,c}(A, B)$ is coNP-hard.

Proof
Let us consider a CNF formula φ over a set of atoms Y. Without loss of generality we can assume that $Y \cap B = \emptyset$. For each atom $y \in Y$, we introduce a fresh atom y'. Thus, in particular, $B \cap (Y \cup Y') = \emptyset$. Finally, we consider programs $P(\varphi)$ and $Q = \{f \leftarrow; \leftarrow f\}$ from the proof of Theorem 5. In the remainder of the proof, we write P for $P(\varphi)$.

From the proof of Theorem 5, we know that P has a model if and only if φ has a model (is satisfiable). We will now show that $Mod_A^{B^c}(P) \neq \emptyset$ if and only if φ is satisfiable. It is easy to check that $Mod_A^{B^c}(Q) = \emptyset$. Thus, the assertion will follow 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 the 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 9 and 10 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.

Before we move on to complement-direct and complement-complement problems, we present a construction that will be of use in both cases. Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$. Without loss of generality we can assume that X and Y are non-empty.

We define X', Y' and $\hat{c}$, for each clause c of φ, as before. Next, let $A, B \subseteq At$ be such that: $A \neq \emptyset$, $A \cap (X \cup X' \cup Y \cup Y') = B \cap (X \cup X' \cup Y \cup Y') = \emptyset$, and let $g \in A$.

We define $W = X \cup X' \cup Y \cup Y' \cup \{g\}$ and observe that $X \cup X' \cup Y \cup Y' \subseteq A^c$ and $g \notin A^c$. Finally, we select an arbitrary element x_0 from X and define the programs $P(\varphi)$ and $Q(\varphi)$ as follows:

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

Lemma 8
Under the notation introduced above, $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are suppmin-equivalent relative to $\mathcal{HB}(A^c, B)$.

Proof
As usual, to simplify notation we write P for $P(\varphi)$ and Q for $Q(\varphi)$. We observe that $At(P) = At(Q) = W$. We observe that both P and Q have the following models that are contained in W:

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

Moreover, P has also additional models contained in W:

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

For each model M of the type 1, $T_P(M) = T_Q(M) = M$, thanks to the rules $u \leftarrow x_0, x_0', u$, where $u \in W$. Thus, for each model M of type 1, we have $M \in Mod_{A^c}(P)$ and $M \in Mod_{A^c}(Q)$.

Let M be a model of P of one of the other two types. Then, we have $T_P(M) = \emptyset$. Moreover, since $g \in M$ and $g \notin A^c$, $M \setminus T_P(M) \not\subseteq A^c$. Thus $M \notin Mod_{A^c}(P)$. Similarly, if M is a model of Q of type 2, $T_Q(M) = \emptyset$. For the same reasons as above, $M \notin Mod_{A^c}(Q)$. Hence, $Mod_{A^c}(P) = Mod_{A^c}(Q)$, and both $Mod_{A^c}(P)$ and $Mod_{A^c}(Q)$ consist of interpretations N of the form $N' \cup V$, where N' is a set of the type 1 and $V \subseteq At \setminus W$. Clearly, for each such set N, $T_P(N) = N' = T_Q(N)$. Thus $T_P(N)|_B = T_Q(N)|_B$ holds for each $(M, N) \in Mod^B_{A^c}(P)$ (as $(M, N) \in Mod^B_{A^c}(P)$ implies $N \in Mod_{A^c}(P)$). By Theorem 2, it follows that P and Q are suppmin-equivalent relative to $\mathcal{HB}(A^c, B)$ if and only if $Mod^B_{A^c}(P) = Mod^B_{A^c}(Q)$.

Thus, to complete the proof, it suffices to show that $\forall Y \exists X \varphi$ is true if and only if $Mod^B_{A^c}(P) = Mod^B_{A^c}(Q)$.

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 $N = \{g\} \cup I \cup (Y \setminus I)' \cup X \cup X'$. We will show that $(\{g\}|_B, N) \in Mod^B_{A^c}(Q)$. Since N is of the type 1, $N \in Mod_{A^c}(Q)$. The requirement (2) for $(\{g\}|_B, N) \in Mod^B_{A^c}(Q)$ is evident, as $g \in N$. The requirement (5) holds, since $N \setminus T_Q(N) = \emptyset \subseteq \{g\}|_B$. By the property of I, N is a minimal model of Q. Thus, the requirements (3) and (4)

hold, too. It follows that $(\{g\}|_B, N) \in Mod^B_{A^c}(Q)$, as claimed. On the other hand $(\{g\}|_B, N) \notin Mod^B_{A^c}(P)$. Indeed, let $M = \{g\} \cup I \cup (Y \setminus I)'$. Then $M \models P$ (it is of the type 3). We now observe that $M \subset N$, $\{g\}|_B = M|_B$ (as $B \cap (Y \cup Y') = \emptyset$), and $M|_{A^c} \supseteq (\{g\}|_B)|_{A^c}$ (as $(\{g\}|_B)|_{A^c} = \emptyset$, due to the fact that $g \notin A^c$). It follows that $(\{g\}|_B, N)$ violates the condition (4) for $(\{g\}|_B, N) \in Mod^B_{A^c}(P)$.

Conversely, let us assume that $\forall Y \exists X \varphi$ is true. We first observe that $Mod^B_{A^c}(P) \subseteq Mod^B_{A^c}(Q)$. Indeed, let $(M, N) \in Mod^B_{A^c}(P)$. Then, $N \in Mod_{A^c}(P)$ and, consequently, $N \in Mod_{A^c}(Q)$. Moreover, if $M|_B = N|_B$, then $N \setminus T_P(N) \subseteq M$ and, as $T_P(N) = T_Q(N)$, $N \setminus T_Q(N) \subseteq M$. Next, as $(M, N) \in Mod^B_{A^c}(P)$, $M \subseteq N|_{A^c \cup B}$. Thus, the requirements (1), (5) and (2) for $(M, N) \in Mod^B_{A^c}(Q)$ hold. Since every model of Q is a model of P, it follows that the conditions (3) and (4) hold, too.

We will now use the assumption that $\forall Y \exists X \varphi$ is true to prove the converse inclusion $Mod^B_{A^c}(Q) \subseteq Mod^B_{A^c}(P)$. To this end, let us consider $(M, N) \in Mod^B_{A^c}(Q)$. Reasoning as above, we can show that the conditions (1), (5) and (2) for $(M, N) \in Mod^B_{A^c}(P)$ hold.

By our earlier comments, $N = N' \cup V$, where N' is of the form 1 and $V \subseteq At \setminus W$. More specifically, $N' = \{g\} \cup I \cup (Y \setminus I)' \cup X \cup X'$, for some $I \subseteq Y$.

Let us consider $Z \subset N$ such that $Z|_{A^c \cup B} = N|_{A^c \cup B}$. Since $W \setminus \{g\} \subseteq A^c$, $Z \supseteq N|_{A^c \cup B} \supseteq I \cup (Y \setminus I)' \cup X \cup X'$. It follows that $Z \cap W$ is not of the type 3. Thus, since $Z \not\models Q$, $Z \not\models P$. Consequently, the condition (3) for $(M, N) \in Mod^B_{A^c}(P)$ holds.

So, let us consider $Z \subset N$ such that $Z|_B = M|_B$ and $Z|_{A^c} \supseteq M|_{A^c}$. Let us assume that $Z \models P$. Since $Z \not\models Q$, $Z = Z' \cup U$, where Z' is a set of the type 3 and $U \subseteq At \setminus W$. Since $Z \subseteq N$, $Z' \subseteq N'$, and so, $Z' = \{g\} \cup I \cup (Y \setminus I)'$.

Since $\forall Y \exists X \varphi$ is true, there is $J \subseteq X$ such that $I \cup J \models \varphi$. It follows that

$$N'' = \{g\} \cup I \cup (Y \setminus I)' \cup J \cup (X \setminus J)' \cup U$$

is a model of both P and Q (of the type 2). Since $B \cap W \subseteq \{g\}$, it follows that $N''|_B = Z|_B = M|_B$. Since $N'' \supseteq Z$, $N''|_{A^c} \supseteq Z|_{A^c} \supseteq M|_{A^c}$. Moreover, $N'' \subset N$. Since $(M, N) \in Mod^B_{A^c}(Q)$, $N'' \not\models Q$, a contradiction. Thus, $Z \not\models P$ and, consequently, the condition (4) for $(M, N) \in Mod^B_{A^c}(P)$ holds. This completes the proof of $Mod^B_{A^c}(Q) \subseteq Mod^B_{A^c}(P)$ and of the lemma. □

We now apply this lemma to complement-direct problems. We have the following result.

Theorem 11
The problem $\text{SUPPMIN}_{c,d}(A, B)$, where $A \neq \emptyset$, is Π^P_2-hard.

Proof
Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$ such that X and Y are nonempty. We can assume that $A \cap (X \cup Y) = B \cap (X \cup Y) = \emptyset$ (if not, variables in the QBF can be renamed). We define X' and Y' as in other places. Thus, $(A \cup B) \cap (X' \cup Y') = \emptyset$. Finally, we pick $g \in A$, and define $P(\varphi)$ and $Q(\varphi)$ as above. By Lemma 8, $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are suppmin-equivalent with respect to $\mathcal{HB}(A^c, B)$. Thus, the assertion follows. (We note that

since B is fixed, we cannot assume $g \in B$ or $g \notin B$ here; however, Lemma 8 takes care of both cases). □

We are now in a position to establish exactly the complexity of complement-direct problems.

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

Proof
For problems $\text{SUPPMIN}_{c,d}(A, B)$ (where $A \neq \emptyset$), $\text{SUPPMIN}_{c,d}(\cdot, B)$, $\text{SUPPMIN}_{c,d}(A,\cdot)$ (where $A \neq \emptyset$), and $\text{SUPPMIN}_{c,d}(\cdot,\cdot)$, the upper bound follows from Theorem 7, and the lower bound from Theorem 11. The problems $\text{SUPPMIN}_{c,d}(\emptyset, B)$ and $\text{SUPPMIN}_{c,d}(\emptyset,\cdot)$ were proved to be coNP-complete by Truszczyński and Woltran (2008) (in fact, they denoted these problems by SUPPMIN_{At}^{B} and SUPPMIN_{At}, respectively). □

We will now apply Lemma 8 to complement-complement problems.

Theorem 12
The problem $\text{SUPPMIN}_{c,c}(A,\cdot)$, where $A \neq \emptyset$, 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 select $g \in A$, and define X' and Y' as usual. Without loss of generality we can assume that $A \cap (X \cup X' \cup Y \cup Y') = \emptyset$. In particular, $g \notin X \cup X' \cup Y \cup Y'$. We set $B = X \cup X' \cup Y \cup Y'$ and so, $B^c \cap (X \cup X' \cup Y \cup Y') = \emptyset$. Finally, we set $W = X \cup X' \cup Y \cup Y' \cup \{g\}$ and define programs P and Q as we did in preparation for Lemma 8. By Lemma 8, $\forall Y \exists X \varphi$ is true if and only if P and Q are suppmin-equivalent with respect to $\mathcal{HB}(A^c, B^c)$. Thus, the assertion follows. □

Next, we determine the lower bound for the problem $\text{SUPPMIN}_{c,c}(A, B)$.

Theorem 13
The problem $\text{SUPPMIN}_{c,c}(A, B)$ is coNP-hard.

Proof
The problem $\text{SUPPMIN}_{c,c}(\emptyset, \emptyset)$ is coNP-complete (Truszczyński and Woltran 2008) (in the paper proving that fact, the problem was denoted by SUPPMIN_{At}^{At}). We will show that it can be reduced to $\text{SUPPMIN}_{c,c}(A, B)$ (for any finite $A, B \subseteq At$).

Thus, let us fix A and B as two finite subsets of At, and let P and Q be normal logic programs. We define P' and Q' to be programs obtained by replacing consistently atoms in P and Q that belong to $A \cup B$ with atoms that do not belong to $At(P \cup Q) \cup A \cup B$. Clearly, P and Q are suppmin-equivalent relative to $\mathcal{HB}(At, At)$ if and only if P' and Q' are suppmin-equivalent relative to $\mathcal{HB}(At, At)$.

Moreover, it is clear that suppmin-equivalence relative to $\mathcal{HB}(At, At)$ between P' and Q' implies suppmin-equivalence relative to $\mathcal{HB}(A^c, B^c)$ between P' and Q'. We will now show the converse implication. To this end, let R be an arbitrary program from $\mathcal{HB}(At, At)$. By R' we denote the program obtained by replacing consistently atoms in R that belong to $A \cup B$ with atoms that do not belong to $At(P' \cup Q') \cup A \cup B$. Since P' and Q' are suppmin-equivalent relative to $\mathcal{HB}(A^c, B^c)$, $P' \cup R'$ and $Q' \cup R'$ have the same suppmin models. Now, we note that because $(A \cup B) \cap At(P' \cup Q') = \emptyset$, $P' \cup R'$ and $Q' \cup R'$ have the same suppmin models if and only if $P' \cup R$ and $Q' \cup R$ have the same suppmin models. Thus, $P' \cup R$ and $Q' \cup R$ have the same suppmin models and, consequently, P' and Q' are suppmin-equivalent relative to $\mathcal{HB}(At, At)$. It follows that P and Q are suppmin-equivalent relative to $\mathcal{HB}(At, At)$.

By this discussion P and Q are suppmin-equivalent relative to $\mathcal{HB}(At, At)$ if and only if P' and Q' are suppmin-equivalent relative to $\mathcal{HB}(A^c, B^c)$. coNP-hardness of $\text{SUPPMIN}_{c,c}(A, B)$ thus follows from the coNP-hardness of $\text{SUPPMIN}_{c,c}(\emptyset, \emptyset)$. □

Taking into account Theorems 7 and 8, Theorems 12 and 13 yield the following result.

Corollary 4
The problems $\text{SUPPMIN}_{c,c}(A, \cdot)$, with $A \neq \emptyset$, and $\text{SUPPMIN}_{c,c}(\cdot, \cdot)$ are Π_2^P-complete. The problems $\text{SUPPMIN}_{c,c}(A, B)$, $\text{SUPPMIN}_{c,c}(\cdot, B)$, and $\text{SUPPMIN}_{c,c}(\emptyset, \cdot)$ are coNP-complete.

6 Stable-equivalence

In this section, we establish the complexity for direct-complement, complement-direct and complement-complement problems of deciding stable-equivalence. We will again make use of the relations depicted in Figure 1 to obtain our results. Thus, for instance, when we derive an upper bound for a problem $\text{STABLE}_{\delta,\varepsilon}(\cdot, \cdot)$ and a matching lower bound for $\text{STABLE}_{\delta,\varepsilon}(A, B)$, we obtain the exact complexity result for all problems between $\text{STABLE}_{\delta,\varepsilon}(A, B)$ and $\text{STABLE}_{\delta,\varepsilon}(\cdot, \cdot)$ (inclusively). As we will show, for stable equivalence those bounds match in all cases other than $\delta = \varepsilon = c$.

We also mention that for the upper bounds for relativized hyperequivalence with respect to the stable-model semantics, some relevant results were established before. Specifically, the direct-direct problem $\text{STABLE}_{d,d}(\cdot, \cdot)$ is known to be in the class Π_2^P and, under the restriction to normal logic programs, in coNP (Woltran 2008). However, for the sake of completeness we treat the direct-direct problems here in full detail as, in the case of fixed alphabets, they were not considered before.

6.1 Upper Bounds

The following lemmas mirror the corresponding results from the previous section but show some interesting differences. For instance, as the following result shows, the problem of model checking is slightly harder now compared to Lemma 3.

Namely, it is located in the class D^P. (We recall that the class D^P consists of all problems expressible as the conjunction of a problem in NP and a problem in coNP.) However, this increase in complexity compared to Lemma 3 does not influence the subsequent Π_2^P-membership results, since a call to a D^P-oracle amounts to two NP-oracle calls.

Lemma 9
The following problems are in the class D^P: given a program P, and sets X, Y, A, and B, decide whether $(X, Y) \in SE^{B'}_{A'}(P)$, where A' stands for one of A and A^c, and B' stands for one of B and B^c,

Proof
We use similar arguments as in the proof of Lemma 3, but we need now both an NP and a coNP test.

We recall 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 $V \cap A = \emptyset$. The same holds for B^c.

Let $A' \in \{A, A^c\}$ and $B' \in \{B, B^c\}$. We will use the observation above to establish upper bounds on the complexity of deciding each of the conditions (1) - (5) for $(X, Y) \in SE^{B'}_{A'}(P)$.

The condition (1) can clearly be decided in polynomial time. The same holds for the condition (2). It is evident once we note that $X \subseteq Y|_{A' \cup B'}$ is equivalent to $X \subseteq Y \cap (A \cup B)$, $X \subseteq (Y \cap B) \cup (Y \setminus A)$, $X \subseteq (Y \cap A) \cup (Y \setminus B)$, and $X \subseteq Y \setminus (A \cap B)$, depending on the form of A' and B'.

It is also easy to show that each of the conditions (3) and (4) can be decided by means of a single coNP test, and that the condition (5) can be decided by means of one NP test. For all instantiations of A' and B', the arguments are similar. We present the details for one case only. For example, if A' stands for A and B' stands for B^c, to decide whether (X, Y) violates the condition (4), we guess a set $Z \subset Y$ and verify that (a) $Z|_{B^c} \subseteq X|_{B^c}$ (by checking that $Z \setminus B \subseteq X \setminus B$); (b) $X|_A \subseteq Z|_A$; (c) one of the two inclusions is proper; and (d) $Z \models P^Y$. All these tasks can be accomplished in polynomial time, and so deciding that the condition (4) does not hold amounts to an NP test. Consequently, deciding that the condition (4) holds can be accomplished by a coNP test. □

When we fix A and B (they are no longer components of the input), the complexity of testing whether $(X, Y) \in SE^{B^c}_{A^c}(P)$ is lower — the problem is in the class *Pol*. Comparing with Lemma 4, the lower complexity holds only for $A' = A^c$ and $B' = B^c$. Moreover, *both* A and B must be fixed.

Lemma 10
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 $(X, Y) \in SE^{B^c}_{A^c}(P)$.

Proof
As we noted, testing the conditions (1) and (2) for $(X, Y) \in SE^{B^c}_{A^c}(P)$ can be done in polynomial time.

For the condition (3) we check all candidate sets Z. Since $Z|_{A^c} = Y|_{A^c}$ all elements of Z are determined by Y except possibly for those that are also in A. Thus, there are at most $2^{|A|}$ possible sets Z to consider. Since A is fixed (not a part of the input), checking for all these sets Z whether $Z \models P^Y$ and $Z \subset Y$ can be done in polynomial time.

For the condition (4), the argument is similar. We note that Z is, in particular, restricted by $Z|_{B^c} \subseteq X|_{B^c}$ and $X|_{A^c} \subseteq Z|_{A^c}$. The two conditions imply that $X|_{A^c \cap B^c} = Z|_{A^c \cap B^c}$. Thus, all elements of Z are determined except possibly for those that are also in $A \cup B$. It follows that there are at most $2^{|A \cup B|}$ possibilities for Z to consider. Clearly, for each of them, we can check whether it satisfies or fails the premises and the consequent of (4) in polynomial time. Thus, checking the condition (4) is a polynomial-time task.

The same (essentially) argument works also for the condition (5). Since $Z|_{A^c \cup B^c} = X|_{A^c \cup B^c}$, all elements of Z are determined except possibly for those that are also in $A \cap B$. Thus, there are at most $2^{|A \cap B|}$ possible sets Z to consider. Given that A and B are fixed, checking all those sets Z for $Z \models P^Y$ and $Z \subset Y$ can be done in polynomial time. □

The reduct of a *normal* program is a Horn program. That property allows us to obtain stronger upper bounds for the case of normal logic programs.

Lemma 11
The following problems are in the class Pol. Given a normal program P, and sets X, Y, A, and B, decide whether $(X, Y) \in SE^{B'}_{A'}(P)$, where A' stands for A or A^c, and B' stands for B or B^c.

Proof
As we noted, deciding the conditions (1) and (2) can be accomplished in polynomial time (even without the assumption of normality).

To show that the condition (3) can be decided in polynomial time, we show that the complement of (3) can be decided in polynomial time. The complement of (3) has the form: there is $Z \subset Y$ such that $Z|_{A'} = Y|_{A'}$ and $Z \models P^Y$. Let us consider the Horn program $P' = P^Y \cup Y|_{A'}$. Since P, Y and A are given, P' can be constructed in polynomial time (for instance, if $A = A^c$, $P' = P^Y \cup (Y \setminus A)$). We will show that the complement of the condition (3) holds if and only if P' is consistent and its least model, say L, satisfies $L \subset Y$ and $L|_{A'} = Y|_{A'}$. First, we observe that if the complement of (3) holds, then P' has a model Z such that $Z \subset Y$ and $Z|_{A'} = Y|_{A'}$. It follows that P' is consistent and its least model, say L, satisfies $L \subseteq Z$. Thus, $L \subset Y$ and $L|_{A'} \subseteq Y|_{A'}$. Moreover, since $L \models P'$, $Y|_{A'} \subseteq L$. Thus, $Y|_{A'} \subseteq L|_{A'}$. Therefore, we have $L \subset Y$ and $L|_{A'} = Y|_{A'}$ as needed. The converse implication is trivial. Since P' can be constructed in polynomial time and L can be computed in polynomial time (P' is Horn), deciding the complement of the condition (3) can be accomplished in polynomial time, too.

To settle the condition (4), we again demonstrate that the complement of the condition (4) can be decided in polynomial time. To this end, we observe that the complement of (4) holds if and only if one of the following two conditions holds:

$4'$. there is $Z \subset Y$ such that, $X|_{A'} \subseteq Z|_{A'}$, $Z|_{B'} \subset X|_{B'}$ and $Z \models P^Y$
$4''$. there is $Z \subset Y$ such that, $X|_{A'} \subset Z|_{A'}$, $Z|_{B'} \subseteq X|_{B'}$ and $Z \models P^Y$.

One can check that $(4')$ holds if and only if $P^Y \cup X|_{A'}$ is consistent and its least model, say L, satisfies $L \subset Y$ and $L|_{B'} \subset X|_{B'}$. Similarly, $(4'')$ holds if and only if there is $y \in (Y \setminus X)|_{A'}$ such that $P^Y \cup (X \cup \{y\})|_{A'}$ is consistent and its least model, say L, satisfies $L \subset Y$ and $L|_{B'} \subseteq X|_{B'}$. Thus, the conditions $(4')$ and $(4'')$ can be checked in polynomial time.

The argument for the condition (5) is similar to that for the complement of the condition (3). The difference is that instead of P' we use the Horn program $P^Y \cup X|_{A' \cup B'}$. Reusing the argument for (3) with the arbitrary containment of Z in Y (rather than a proper one) shows that the complement of (5) can be decided in polynomial time. □

The next lemma plays a key role in establishing an upper bound on the complexity of the problems $\text{STABLE}_{\delta,\varepsilon}(\cdot,\cdot)$. Its proof is technical and we present it in the appendix.

Lemma 12
Let P, Q be programs and $A, B \subseteq At$. If $(X, Y) \in SE^B_A(P) \setminus SE^B_A(Q)$, then there are sets $X', Y' \subseteq At(P \cup Q)$, such that at least one of the following conditions holds:

i. $(X', Y') \in SE^B_A(P) \setminus SE^B_A(Q)$
ii. $A \setminus At(P \cup Q) \neq \emptyset$ and for every $y, z \in A \setminus At(P \cup Q)$, $(X', Y' \cup \{y, z\}) \in SE^B_A(P) \setminus SE^B_A(Q)$

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

Theorem 14
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 $\delta, \varepsilon \in \{c, d\}$.

Proof
Given finite programs P and Q, and finite subsets A, B of At the following algorithm decides the complementary problem to $\text{STABLE}_{\delta,\varepsilon}(\cdot,\cdot)$. If $\delta = d$ and $A \setminus At(P \cup Q) = \emptyset$, the algorithm guesses two sets $X, Y \subseteq At(P \cup Q)$. It verifies whether $(X, Y) \in SE^B_A(P) \div SE^B_A(Q)$ and if so, returns YES. Otherwise, the algorithm guesses two sets $X, Y \subseteq At(P \cup Q)$. If $\delta = d$, it selects two elements $y, z \in A \setminus At(P \cup Q)$ or, if $\delta = c$, it selects two elements $y, z \in A^c \setminus At(P \cup Q)$. The algorithm verifies whether $(X, Y) \in SE^B_{A'}(P) \div SE^B_{A'}(Q)$ (where $A' = A$ if $\delta = d$, and $A' = A^c$ if $\delta = c$) and if so, returns YES. Otherwise, the algorithm verifies whether $(X, Y \cup \{y, z\}) \in$

$SE^B_{A'}(P) \div SE^B_{A'}(Q)$ (where $A' = A$ if $\delta = d$, and $A' = A^c$ if $\delta = c$) and if so, returns YES.

The correctness of the algorithm follows by Lemma 12. Since the sizes of X and Y are polynomial in the size of $P \cup Q$, the membership of the complementary problem in the class Σ^P_2 follows by Lemma 9.

The remaining claims of the assertion follow in the same way by Lemmas 10 and 11, respectively. □

6.2 Lower bounds and exact complexity results

We start with the case of normal programs.

Theorem 15
The problem $\text{STABLE}^n_{\delta,\varepsilon}(A, B)$ is coNP-hard for any $\delta, \varepsilon \in \{c, d\}$.

Proof
Let us fix δ and ε, and let A' and B' be sets of atoms defined by the combinations A and δ, and B and ε. We will show that UNSAT can be reduced to $\text{STABLE}^n_{\delta,\varepsilon}(A, B)$.

Let φ be a CNF over of set of atoms Y. We define $P(\varphi)$ and Q as in the proof of Theorem 5. We note that both programs are normal. As before, we write P instead of $P(\varphi)$ in order to simplify the notation.

To prove the assertion it suffices to show that φ is unsatisfiable if and only if P and Q are stable-equivalent with respect to $\mathcal{HB}(A', B')$. To this end, we will show that φ is unsatisfiable if and only if $SE^{B'}_{A'}(P) = SE^{B'}_{A'}(Q)$ (cf. Theorem 3).

Since Q has no models, $SE^{B'}_{A'}(Q) = \emptyset$. Moreover, $SE^{B'}_{A'}(P) = \emptyset$ if and only if P has no models (indeed, if $(X, Y) \in SE^{B'}_{A'}(P)$, then Y is a model of P; if Y is a model of P, then $(Y, Y) \in SE^{B'}_{A'}(P)$). It follows that $SE^{B'}_{A'}(P) = SE^{B'}_{A'}(Q)$ if and only if P has no models.

In the proof of Theorem 5, we noted that P has models if and only if φ has models. Thus, $SE^{B'}_{A'}(P) = SE^{B'}_{A'}(Q)$ if and only if φ is unsatisfiable. □

Together with the matching coNP-membership results for $\text{STABLE}^n_{\delta,\varepsilon}(\cdot, \cdot)$ from Theorem 14 we obtain the following result.

Corollary 5
The following problems are coNP-complete for any $\delta, \varepsilon \in \{c, d\}$: $\text{STABLE}^n_{\delta,\varepsilon}(\cdot, \cdot)$, $\text{STABLE}^n_{\delta,\varepsilon}(A, \cdot)$, $\text{STABLE}^n_{\delta,\varepsilon}(\cdot, B)$ and $\text{STABLE}^n_{\delta,\varepsilon}(A, B)$.

We now turn to the case of disjunctive programs. It turns out that the problems $\text{STABLE}_{c,d}(A, B)$, $\text{STABLE}_{d,d}(A, B)$ and $\text{STABLE}_{d,c}(A, B)$ are Π^P_2-hard. The situation is different for $\text{STABLE}_{c,c}(A, B)$. By Theorems 14 and Corollary 5, the problem is coNP-complete. However, the two immediate successors of that problem, $\text{STABLE}_{c,c}(A, \cdot)$ and $\text{STABLE}_{c,c}(\cdot, B)$ (cf. Figure 1) are Π^P_2-hard. We will now show these results.

To start with we provide some technical results concerning the structure of the set $SE^B_A(P)$ when $At(P) \subseteq A$ and $At(P) \cap B = \emptyset$. It will be applicable to programs we construct below.

Lemma 13
Let P be a program and $A, B \subseteq At$. If $At(P) \subseteq A$ and $At(P) \cap B = \emptyset$, then $(X, Y) \in SE^B_A(P)$ if and only if there are $X', Y' \subseteq At(P)$ and $W \subseteq A \setminus At(P)$ such that one of the following conditions holds:

a. $X = X' \cup W$, $Y = Y' \cup W$, and $(X', Y') \in SE^B_A(P)$
b. $X = X' \cup W$, $(X', X') \in SE^B_A(P)$ and $Y = X' \cup W \cup \{y\}$, for some $y \in A \setminus At(P)$
c. $X = X' \cup W$, $(X', X') \in SE^B_A(P)$ and $Y = X' \cup W \cup D$, for some $D \subseteq B \cap (A \setminus At(P))$ such that $W \cap D = \emptyset$ and $|D| \geq 2$.

The proof of this result is technical and we give it in the appendix. This lemma points to the crucial role played by those pairs $(X, Y) \in SE^B_A(P)$ that satisfy $Y \subseteq At(P)$. In particular, as noted in the next result, it allows to narrow down the class of pairs (X, Y) that need to be tested for the membership in $SE^B_A(P)$ and $SE^B_A(Q)$ when considering stable-equivalence of P and Q with respect to $\mathcal{HB}(A, B)$.

Lemma 14
Let P and Q be programs, and A, B subsets of At such that $At(P \cup Q) \subseteq A$ and $At(P \cup Q) \cap B = \emptyset$. Then, P and Q are stable-equivalent with respect to $\mathcal{HB}(A, B)$ if and only if for every X, Y such that $Y \subseteq At(P \cup Q)$, $(X, Y) \in SE^B_A(P)$ if and only if $(X, Y) \in SE^B_A(Q)$.

Proof
Without loss of generality, we can assume that $At(P) = At(Q)$. Indeed, let $P' = P \cup \{a \leftarrow a \mid a \in At(Q) \setminus At(P)\}$ and $Q' = Q \cup \{a \leftarrow a \mid a \in At(P) \setminus At(Q)\}$. It is easy to see that P and P' (Q and Q', respectively) are stable-equivalent with respect to $\mathcal{HB}(A, B)$. Thus, in particular, $SE^B_A(P) = SE^B_A(P')$ and $SE^B_A(Q) = SE^B_A(Q')$. Moreover, $At(P') = At(Q') = At(P \cup Q)$. Therefore, $At(P' \cup Q') \subseteq A$ if and only if $At(P \cup Q) \subseteq A$, and $At(P' \cup Q') \cap B = \emptyset$ if and only if $At(P \cup Q) \cap B = \emptyset$.

Thus, let us assume that $At(P) = At(Q)$. Only the "if" part of the claim requires a proof, the other implication being evident. Let us assume that $(X, Y) \in SE^B_A(P)$. By Lemma 13, there are $X', Y' \subseteq At(P)$ and $W \subseteq A \setminus At(P)$ such that one of the conditions (a) - (c) holds. If (a) holds, $(X', Y') \in SE^B_A(Q)$ and so, $(X, Y) \in SE^B_A(Q)$. If (b) or (c) holds, $(X', X') \in SE^B_A(Q)$ and so, $(X, Y) \in SE^B_A(Q)$, as well. □

Finally, we note that under the assumptions of Lemma 13, if $Y \subseteq At(P)$, then the conditions for $(X, Y) \in SE^B_A(P)$ simplify.

Lemma 15
Let P be a program and $A, B \subseteq At$. If $At(P) \subseteq A$, $At(P) \cap B = \emptyset$ and $Y \subseteq At(P)$, then $(X, Y) \in SE^B_A(P)$ if and only if $Y \models P$, $X \subseteq Y$, $X \models P^Y$, and for every $Z \subset Y$ such that $X \subset Z$, $Z \not\models P^Y$.

Proof
Under the assumptions of the lemma, the four conditions are equivalent to the conditions (1), (2), (5) and (4) for $(X, Y) \in SE^B_A(P)$, respectively, and the condition (3) is vacuously true. □

Our first Π^P_2-hardness result for stable equivalence results concerns the problem $\text{STABLE}_{c,d}(A, B)$.

Theorem 16
The problem $\text{STABLE}_{c,d}(A, B)$ is hard for the class Π^P_2.

Proof
According to our notational convention, we have to show that $\text{STABLE}_{c,d}(A, B)$ is Π^P_2-hard, for every finite $A, B \subseteq At$.

Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$. Without loss of generality we can assume that every clause in φ contains at least one literal x or $\neg x$, for some $x \in X$. Furthermore, we can also assume that $A \cap (X \cup Y) = \emptyset$ and $B \cap (X \cup Y) = \emptyset$ (if not, variables in φ can be renamed). We select the primed (fresh) variables so that $A \cap (X' \cup Y') = \emptyset$ and $B \cap (X' \cup Y') = \emptyset$, as well.

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 stable-equivalent relative to $\mathcal{HB}(A^c, B)$. Since the problem to decide whether a given QBF $\forall Y \exists X \varphi$ is true is Π^P_2-complete, the assertion will follow.

To construct $P(\varphi)$ and $Q(\varphi)$ we select an additional atom $a \notin X \cup X' \cup Y \cup Y' \cup A \cup B$, and use $\hat{c}$, as defined in some of the arguments earlier in the paper. We set

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

and define

$$\begin{aligned} P(\varphi) &= \{x \vee x' \mid x \in X\} \cup R(\varphi) \\ Q(\varphi) &= \{x \vee x' \leftarrow u \mid x \in X, u \in \{a\} \cup X \cup X'\} \cup R(\varphi) \end{aligned}$$

To simplify notation, from now on we write P for $P(\varphi)$ and Q or $Q(\varphi)$.

We note that $At(P) = At(Q)$, $At(P) \subseteq A^c$, $At(Q) \subseteq A^c$, $At(P) \cap B = \emptyset$, and $At(Q) \cap B = \emptyset$. Thus, to determine whether P and Q are stable-equivalent with respect to $\mathcal{HB}(A^c, B)$, we will focus only on pairs $(N, M) \in SE^B_{A^c}(P)$ and $(N, M) \in SE^B_{A^c}(Q)$ that satisfy $N \subseteq M \subseteq At(P)$ (cf. Lemma 14). By Lemma 15, to identify such pairs, we need to consider models (contained in $At(P) = At(Q)$) of the two programs, and models (again contained in $At(P) = At(Q)$) of the reducts of the two programs with respect to their models. From now on in the proof, whenever we use the term "model" (of a program or the reduct of a program) we assume that it is a subset of $At(P) = At(Q)$.

First, one can check that the models of P and Q coincide and are of the form:

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

Next, we look at models of the reducts of P and Q with respect to their models, that is, sets of the form (1). Let M be such a set. Since $a \in M$, then every model of P is a model of P^M, and the same holds for Q.

However, P^M and Q^M have additional models. First, each reduct has as its models sets of the form

2. $I \cup (Y \setminus I)' \cup J \cup (X \setminus J)'$, where $J \subseteq X$, $I \subseteq Y$ and $I \cup J \models \varphi$.

Furthermore, Q^M has additional models, namely, sets of the form

3. $I \cup (Y \setminus I)'$, for each $I \subseteq Y$.

Indeed, it is easy to check that $I \cup (Y \setminus I)'$ satisfies all rules of Q^M (in the case of the rules $a \leftarrow \hat{c}$, we use the fact that every sequence $\hat{c}$ contains an atom x or x' for some $x \in X$).

We will now show that $\forall Y \exists X \varphi$ is true if and only if P and Q are stable-equivalent relative to $\mathcal{HB}(A^c, B)$. To this end, we will show that $\forall Y \exists X \varphi$ is true if and only if $SE^B_{A^c}(P) = SE^B_{A^c}(Q)$.

We recall that since $At(P) = At(Q) \subseteq A^c$ and $At(P) \cap B = At(Q) \cap B = \emptyset$, we can use Lemmas 14 and 15. Thus, if $M \subseteq At(P)$, $(N, M) \in SE^B_{A^c}(P)$ if and only if M is a set of type (1), that is, $M = I \cup (Y \setminus I)' \cup X \cup X' \cup \{a\}$, for some $I \subseteq Y$, and either $N = M$ or N is a set of type (2), that is, $N = I \cup (Y \setminus I)' \cup J \cup (X \setminus J)'$, for some $J \subseteq X$ such that $I \cup J \models \varphi$.

The same pairs (N, M) belong to $SE^B_{A^c}(Q)$ (still under the assumption that $M \subseteq At(P) = At(Q)$). However, $SE^B_{A^c}(Q)$ contains also pairs (N, M) where M is a set of type (1), $N = I \cup (Y \setminus I)'$ *and* for every $J \subseteq X$, $I \cup J \not\models \varphi$ (given that the only models of Q^M that are proper *supersets* of N and proper *subsets* of M are models of type (2), that is precisely what is needed to ensure that for every Z, $N \subset Z \subset M$ implies $Z \not\models Q^M$).

Let us assume that $\forall Y \exists X \varphi$ is false. Then, there exists $I \subseteq Y$ such that for every $J \subseteq X$, $I \cup J \not\models \varphi$. Let $N = I \cup (Y \setminus I)'$ and $M = I \cup (Y \setminus I)' \cup X \cup X' \cup \{a\}$. From our discussion, it is clear that $(N, M) \in SE^B_{A^c}(Q)$ but $(N, M) \notin SE^B_{A^c}(P)$. Thus, $SE^B_{A^c}(P) \neq SE^B_{A^c}(Q)$.

Conversely, if $\forall Y \exists X \varphi$ is true, then for every $I \subseteq Y$ there is $J \subseteq X$ such that $I \cup J \models \varphi$. This implies that there are no pairs $(N, M) \in SE^B_{A^c}(Q)$ of the last kind. Thus, in that case, if $M \subseteq At(P)$=$At(Q)$, then $(N, M) \in SE^B_{A^c}(P)$ if and only if $(N, M) \in SE^B_{A^c}(Q)$. By Lemma 14, $SE^B_{A^c}(P) = SE^B_{A^c}(Q)$. □

Combining Theorem 16 with Theorem 14 yields the following result.

Corollary 6
The problems $\text{STABLE}_{c,d}(A, B)$, $\text{STABLE}_{c,d}(\cdot, B)$, $\text{STABLE}_{c,d}(A, \cdot)$ and $\text{STABLE}_{c,d}(\cdot, \cdot)$, are Π^P_2-complete.

Next, we consider the problems $\text{STABLE}_{d,c}(A, B)$, and $\text{STABLE}_{d,d}(A, B)$. We have the following simple result.

Lemma 16
Let P and Q be programs and A, B subsets of At such that $At(P \cup Q) \cap A = \emptyset$. Then, P and Q are stable-equivalent with respect to $\mathcal{HB}(A, B)$ if and only if P and Q have the same stable models.

Proof
Let $R \in \mathcal{HB}(A, B)$. Since $At(P \cup Q) \cap A = \emptyset$, we can apply the splitting theorem (Lifschitz and Turner 1994) to $P \cup R$. It follows that M is a stable model of $P \cup R$ if and only if $M = M' \cup M''$, where M' is a stable model of P and M'' is a stable model of $M'' \cup R$. Similarly, M is a stable model of $Q \cup R$ if and only if $M = M' \cup M''$, where M' is a stable model of Q and M'' is a stable model of $M'' \cup R$. Thus, the assertion follows. □

We now use this result to determine the lower bounds on the complexity of problems $\text{STABLE}_{d,c}(A, B)$ and $\text{STABLE}_{d,d}(A, B)$.

Theorem 17
The problems $\text{STABLE}_{d,c}(A, B)$ and $\text{STABLE}_{d,d}(A, B)$ are hard for the class Π_2^P.

Proof
To be precise, we have to show that $\text{STABLE}_{d,c}(A, B)$ and $\text{STABLE}_{d,d}(A, B)$ are Π_2^P-hard, for every finite $A, B \subseteq At$.

It is well known that the problem to decide whether a logic program P has a stable model is Σ_2^P-complete (Eiter and Gottlob 1995). We will reduce this problem to the complement of $\text{STABLE}_{d,c}(A, B)$ ($\text{STABLE}_{d,d}(A, B)$, respectively). That will complete the proof.

Thus, let P be a logic program. Without loss of generality, we can assume that $At(P) \cap A = \emptyset$ (if not, we can rename atoms in P, without affecting the existence of stable models). Let f be an atom not in A. and define $Q = \{f, \leftarrow f\}$. Clearly, $At(P \cup Q) \cap A = \emptyset$. Moreover, P and Q do not have the same stable models if and only if P has stable models. By Lemma 16, P has stable models if and only if P and Q are not stable-equivalent relative to $\mathcal{HB}(A, B^c)$. Similarly (as B is immaterial for the stable-equivalence in that case), P has stable models if and only if P and Q are not stable-equivalent relative to $\mathcal{HB}(A, B)$. □

We now explicitly list all cases, where we are able to give completeness results (membership results are from Theorem 14).

Corollary 7
The problems $\text{STABLE}_{d,d}(A, B)$, $\text{STABLE}_{d,d}(\cdot, B)$, $\text{STABLE}_{d,d}(A, \cdot)$ and $\text{STABLE}_{d,d}(\cdot, \cdot)$, are Π_2^P-complete.

Corollary 8
The problems $\text{STABLE}_{d,c}(A, B)$, $\text{STABLE}_{d,c}(\cdot, B)$, $\text{STABLE}_{d,c}(A, \cdot)$ and $\text{STABLE}_{d,c}(\cdot, \cdot)$, are Π_2^P-complete.

Finally, we show Π_2^P-hardness of problems $\text{STABLE}_{c,c}(A, \cdot)$ and $\text{STABLE}_{c,c}(\cdot, B)$.

Theorem 18
The problems $\text{STABLE}_{c,c}(A, \cdot)$ and $\text{STABLE}_{c,c}(\cdot, B)$. are Π_2^P-hard.

Proof
We first show that the problem $\text{STABLE}_{c,c}(A, \cdot)$ is Π_2^P-hard, for every finite $A \subseteq At$. Let $\forall Y \exists X \varphi$ be a QBF, where φ is a CNF formula over $X \cup Y$. As in the proof of Theorem 16, without loss of generality we can assume that every clause in φ contains a literal x or $\neg x$, for some $x \in X$, and that $A \cap (X \cup Y) = \emptyset$ (if not, variables in φ can be renamed).

Let $P(\varphi)$ and $Q(\varphi)$ be the programs used in the proof of Theorem 16, where we choose primed variables so that $A \cap (X' \cup Y') = \emptyset$. We define $B = At(P)$. We have that $At(P) \subseteq A^c$ and $At(P) \cap B^c = \emptyset$.

We recall that the argument used in the proof of Theorem 16 to show that $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are stable-equivalent with respect to $\mathcal{HB}(A^c, B)$ does not depend on the finiteness of B but only on the fact that $B \cap At(P) = \emptyset$. Thus, the same argument shows that $\forall Y \exists X \varphi$ is true if and only if $P(\varphi)$ and $Q(\varphi)$ are stable-equivalent with respect to $\mathcal{HB}(A^c, B^c)$. It follows that $\text{STABLE}_{c,c}(A, \cdot)$ is Π_2^P-hard.

Next, we show that the problem $\text{STABLE}_{c,c}(\cdot, B)$ is Π_2^P-hard, for every finite $B \subseteq At$. We reason as in the proof of Theorem 17. That is, we construct a reduction from the problem to decide whether a logic program has no stable models. Specifically, let P be a logic program. We define $A = At(P)$. Clearly, we have $At(P) \cap A^c = \emptyset$. We recall the argument used in Theorem 17 to show that P has stable models if and only if P and $Q = \{f, \leftarrow f\}$ are not stable-equivalent with respect to $\mathcal{HB}(A, B)$ does not depend on the finiteness of A nor on B. Thus, it follows that P has stable models if and only if P and $Q = \{f, \leftarrow f\}$ are not stable-equivalent with respect to $\mathcal{HB}(A^c, B^c)$ and the Π_2^P-hardness of $\text{STABLE}_{c,c}(\cdot, B)$ follows. □

We put the things together using Theorem 15 for the coNP-hardness and Theorem 18 for the Π_2^P-hardness. The matching upper bounds are from Theorem 14.

Corollary 9
The problem $\text{STABLE}_{c,c}(A, B)$ is coNP-complete. The problems $\text{STABLE}_{c,c}(\cdot, B)$, $\text{STABLE}_{c,c}(A, \cdot)$ and $\text{STABLE}_{c,c}(\cdot, \cdot)$, are Π_2^P-complete.

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 (for the sake of completeness we also include

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

δ	ε	α	β	SUPP	SUPPMIN	STABLE	STABLE^n
d	d			coNP	Π_2^P	Π_2^P	coNP
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

the complexity of direct-direct problems). It shows that the problems concerning supp-equivalence (no normality restriction), and stable-equivalence for normal programs are all coNP-complete (cf. Corollaries 1 and 5, respectively). 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 $\emptyset$. The results for direct-direct problems were known earlier (Truszczyński and Woltran 2008), the results for the direct-complement problems are by Corollary 2, for the complement-complement problems results are by Corollary 4, and for the complement-direct problems results are by Corollary 3. 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. The results for direct-direct (direct-complement, complement-complement, complement-direct, respectively) problems are by Corollary 7 (8, 9, 6, respectively) in this paper. We also note that the complexity of problems for stable-equivalence is always at least that for suppmin-equivalence.

Our research opens questions worthy of further investigations. First, we believe that results presented here may turn out important for building "intelligent" programming environments supporting development of logic programs. For instance, a programmer might want to know the effect of changes she just made to a program (perhaps already developed earlier) that represents a module of a larger project. One way to formalize that effect is to define it as the maximal class of contexts of the form $\mathcal{HB}(A', B')$ with respect to which the original and the revised versions of the program are equivalent (say under the stable-model semantics). The sets A' and B' appearing in the specification of such a class of contexts will be of the form A^c and B^c, for some finite sets A and B. Finding the appropriate sets A and B would provide useful information to the programmer. Our results on the complexity of the

complement-complement version of the hyperequivalence problem and their proofs may yield insights into the complexity of finding such sets A and B, and suggest algorithms.

Second, there are other versions of hyperequivalence that need to be investigated. For instance, while stable-equivalence when only parts of models are compared (projections on a prespecified set of atoms) was studied (Eiter et al. 2005; Oetsch et al. 2007), 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

This work was partially supported by the NSF grant IIS-0325063, the KSEF grant KSEF-1036-RDE-008, and by the Austrian Science Fund (FWF) under grants P18019 and P20704.

Appendix

We present here proofs of some technical results we needed in the paper. We first prove Lemma 6. We start with two auxiliary results.

Lemma 17
Let P be a program and $A, B \subseteq At$. Let $y \in X$ be such that $y \notin At(P) \cup A$. Then $(X, Y) \in Mod^B_{A^c}(P)$ if and only if $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$.

Proof
($\Rightarrow$) Since $Y \in Mod_{A^c}(P)$, $Y \models P$ and $Y \setminus T_P(Y) \subseteq A^c$. We have $y \notin At(P)$. Thus, $Y \setminus \{y\} \models P$ and $T_P(Y) = T_P(Y \setminus \{y\})$. Since $Y \setminus \{y\} \subseteq Y$, $(Y \setminus \{y\}) \setminus T_P(Y \setminus \{y\}) \subseteq A^c$. It follows that $Y \setminus \{y\} \in Mod_{A^c}(P)$. Thus, the condition (1) for $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ holds. The condition (2) for $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ is evident.

Let $Z \subset Y \setminus \{y\}$ be such that $Z|_{A^c \cup B} = (Y \setminus \{y\})|_{A^c \cup B}$. Let $Z' = Z \cup \{y\}$. We have $y \in X$ and so, $y \in Y$. Hence, $Z' \subset Y$. Since $y \notin A$, $y \in A^c$. Thus, $Z'|_{A^c \cup B} = Y|_{A^c \cup B}$. It follows that $Z' \not\models P$ and, consequently, $Z \not\models P$ (as $y \notin At(P)$). Thus, the condition (3) for $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ holds.

Next, let $Z \subset Y \setminus \{y\}$ be such that $Z|_B = (X \setminus \{y\})|_B$ and $Z|_{A^c} \supseteq (X \setminus \{y\})|_{A^c}$. As before, let $Z' = Z \cup \{y\}$. Since $y \in X$ and $y \in Y$ (see above), $Z' \subset Y$, $Z'|_B = X|_B$ and $Z'|_{A^c} \supseteq X|_{A^c}$. Thus, $Z' \not\models P$. Since $y \notin At(P)$, $Z \not\models P$ and the condition (4) for $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ holds.

Finally, let $(X \setminus \{y\})|_B = (Y \setminus \{y\})|_B$. Clearly, it follows that $X|_B = Y|_B$. Thus, $Y \setminus T_P(Y) \subseteq X$. Since $y \notin At(P)$, $T_P(Y) = T_P(Y \setminus \{y\})$. It follows that $(Y \setminus \{y\}) \setminus T_P(Y \setminus \{y\}) \subseteq X \setminus \{y\}$. Consequently, the condition (5) for $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ is satisfied, as well.

($\Leftarrow$) By the assumption, we have $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$. Thus, $Y \setminus \{y\} \in$

$Mod_{A^c}(P)$ and, consequently, $Y \setminus \{y\}$ is a model of P. Since $y \notin At(P)$, Y is a model of P. We also have $(Y \setminus \{y\}) \setminus T_P(Y \setminus \{y\}) \subseteq A^c$. Since $y \notin At(P)$, $T_P(Y \setminus \{y\}) = T_P(Y)$. Thus, as $y \in A^c$, $Y \setminus T_P(Y) \subseteq A^c$. That is, the condition (1) for $(X, Y) \in Mod^B_{A^c}(P)$ holds. The condition (2) follows from $y \in A^c$ and $X \setminus \{y\} \subseteq (Y \setminus \{y\})|_{A^c \cup B}$.

Let $Z \subset Y$ be such that $Z|_{A^c \cup B} = Y|_{A^c \cup B}$. It follows that $y \in Z$ (we recall that $y \in X \subseteq Y$ and $y \in A^c$). Let $Z' = Z \setminus \{y\}$. We have $Z' \subset Y \setminus \{y\}$ and $Z'|_{A^c \cup B} = (Y \setminus \{y\})|_{A^c \cup B}$. Thus, $Z' \not\models P$ and, consequently, $Z \not\models P$. It follows that the condition (3) for $(X, Y) \in Mod^B_{A^c}(P)$ holds.

Let $Z \subset Y$ be such that $Z|_B = X|_B$ and $Z|_{A^c} \supseteq X|_{A^c}$. Since $y \in X$ and $y \in A^c$, $y \in Z$. Let $Z' = Z \setminus \{y\}$. It follows that $Z' \subset Y \setminus \{y\}$, $Z'|_B = (X \setminus \{y\})|_B$, and $Z'|_{A^c} \supseteq (X \setminus \{y\})|_{A^c}$. Hence, $Z' \not\models P$ and so, $Z \not\models P$. In other words, the condition (4) for $(X, Y) \in Mod^B_{A^c}(P)$, holds.

Finally, let $X|_B = Y|_B$. Clearly, $(X \setminus \{y\})|_B = (Y \setminus \{y\})|_B$ and so, $(Y \setminus \{y\}) \setminus T_P(Y \setminus \{y\}) \subseteq X \setminus \{y\}$. Since $T_P(Y \setminus \{y\}) = T_P(Y)$, we obtain $Y \setminus T_P(Y) \subseteq X$. Thus, (5) for $(X, Y) \in Mod^B_{A^c}(P)$, holds. □

Lemma 18
Let P be a program, $A, B \subseteq At$. If $X|_B \subset Y|_B$, $y \in (Y \setminus X) \setminus (At(P) \cup A)$, and $(Y \setminus \{y\})|_B \neq X|_B$, then $(X, Y) \in Mod^B_{A^c}(P)$ if and only if $(X, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$.

Proof
($\Rightarrow$) The arguments for the conditions (1), (2) and (3) for $(X, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ are essentially the same as in Lemma 17 (although the argument for the condition (2) requires also the assumption that $y \notin X$).

Next, let $Z \subset Y \setminus \{y\}$ be such that $Z|_B = X|_B$ and $Z|_{A^c} \supseteq X|_{A^c}$. Then $Z \subset Y$ and so, $Z \not\models P$. Thus, the condition (4) for $(X, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ holds.

Finally, $(Y \setminus \{y\})|_B \neq X|_B$; the condition (5) for $(X, Y \setminus \{y\}) \in Mod^B_{A^c}(P)$ is thus trivially true.

($\Leftarrow$) As above, the arguments for the conditions (1), (2) and (3) for $(X, Y) \in Mod^B_{A^c}(P)$ are the same as in Lemma 17.

Let $Z \subset Y$ be such that $Z|_B = X|_B$ and $Z|_{A^c} \supseteq X|_{A^c}$. Since $(Y \setminus \{y\})|_B \neq X|_B$, $Z \neq Y \setminus \{y\}$. Thus, $Z \subset Y \setminus \{y\}$ and so, $Z \not\models P$. That is, the condition (4) for $(X, Y) \in Mod^B_{A^c}(P)$, holds. Finally, since $X|_B \subset Y|_B$, the condition (5) for $(X, Y) \in Mod^B_{A^c}(P)$, holds, as well. □

We are now ready to prove Lemma 6.

Lemma 6
Let P, Q be programs and $A, B \subseteq At$.

1. *If $(X, Y) \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$ then there is $(X', Y') \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$ such that $Y' \subseteq At(P \cup Q) \cup A$.*
2. *If $(X, Y) \in Mod^B_{A^c}(P)$ and $T_P(Y)|_B \neq T_Q(Y)|_B$, then there is $(X', Y') \in Mod^B_{A^c}(P)$ such that $T_P(Y')|_B \neq T_Q(Y')|_B$ and $Y' \subseteq At(P \cup Q) \cup A$.*

Proof
(1) Let $(X, Y) \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$ and let $y \in X$ be such that $y \notin At(P \cup Q) \cup A$. Then, by Lemma 17, $(X \setminus \{y\}, Y \setminus \{y\}) \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$. By repeating this process, we arrive at a pair $(X'', Y'') \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$ such that $X'' \subseteq At(P \cup Q) \cup A$.

If $X''|_B = Y''|_B$, then $Y'' \setminus T_P(Y'') \subseteq X''$. Thus, $Y'' \subseteq T_P(Y'') \cup X'' \subseteq At(P \cup Q) \cup A$. Thus, let us consider the other possibility that $X''|_B \subset Y''|_B$ (indeed, as $X'' \subseteq Y''|_{A^c \cup B} \subseteq Y''$, there are no other possibilities). Let $y \in (Y'' \setminus X'') \setminus (At(P \cup Q) \cup A)$ be such that $(Y'' \setminus \{y\})|_B \neq X''|_B$. By Lemma 18, $(X'', Y'' \setminus \{y\}) \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$. By repeating this process, we arrive at a pair $(X', Y') \in Mod^B_{A^c}(P) \setminus Mod^B_{A^c}(Q)$ such that for every $y \in (Y' \setminus X') \setminus (At(P \cup Q) \cup A)$, $(Y' \setminus \{y\})|_B = X'|_B$. Since $X' = X''$, $X' \subseteq At(P \cup Q) \cup A$.

We also note that for every $y \notin X'$, $(Y' \setminus \{y\}) \supseteq X'$ (as $Y' \supseteq X'$) and so, $(Y' \setminus \{y\})|_{A^c} \supseteq X'|_{A^c}$. We will now show that $Y' \subseteq At(P \cup Q) \cup A$. To this end, let us assume that there is $y \in Y'$ such that $y \notin At(P \cup Q) \cup A$. Since $X' \subseteq At(P \cup Q) \cup A$, $y \notin X'$. Thus, $y \in (Y' \setminus X') \setminus (At(P \cup Q) \cup A)$. It follows that $(Y' \setminus \{y\})|_B = X'|_B$ and $(Y' \setminus \{y\})|_{A^c} \supseteq X'|_{A^c}$. Since $Y' \setminus \{y\} \subset Y'$ and $(X', Y') \in Mod^B_{A^c}(P)$, $Y' \setminus \{y\} \not\models P$. On the other hand, $Y' \models P$ and, since $y \notin At(P)$, $Y' \setminus \{y\} \models P$, a contradiction.

(2) It is easy to see that if we apply the construction described in (1) to (X, Y) we obtain (X', Y') such that $Y' \subseteq At(P \cup Q) \cup A$ and $T_P(Y')|_B \neq T_Q(Y')|_B$. Indeed, in every step of the construction, we eliminate an element y such that $y \notin At(P \cup Q)$, which has no effect on the values of T_P and T_Q. □

Lemma 7
Let P, Q be programs and $B \subseteq At$. If $Mod^B_{At}(P) \neq Mod^B_{At}(Q)$, then there is $Y \subseteq At(P \cup Q)$ 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). Without loss of generality we can assume that there is $(X, Y) \in Mod^B_{At}(P) \setminus Mod^B_{At}(Q)$. Moreover, by Lemma 6, we can assume that $Y \subseteq At(P \cup Q)$ (recall $At^c = \emptyset$). It follows that (X, Y) satisfies the conditions (1)-(5) for $(X, Y) \in Mod^B_{At}(P)$. Since P and Q have the same models, (X, Y) satisfies the conditions (1)-(4) for $(X, Y) \in Mod^B_{At}(Q)$. Hence, (X, Y) violates the condition (5) for $(X, Y) \in Mod^B_{At}(Q)$, that is, $X|_B = Y|_B$ and $Y \setminus T_Q(Y) \not\subseteq X$ hold. In particular, 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 and so, $Y \in Mod_{At}(P)$. Next, obviously, $Y \setminus \{a\} \subseteq Y$. Thus, the conditions (1) and (2) for $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$ hold. The condition (3) is trivially true.

Further, let $Z \subset Y$ be such that $Z|_B = (Y \setminus \{a\})|_B$ and $Z \supseteq Y \setminus \{a\}$. 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 $Y \setminus \{a\} \subset Y$ and $(X, Y) \in Mod^B_{At}(P)$,

$Y \setminus \{a\} \not\models P$, that is, $Z \not\models P$. Thus, the condition (4) for $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$ holds.

Since $a \notin B$, $(Y \setminus \{a\})|_B = Y|_B$. Thus, we also have to verify the condition (5). We have $Y \setminus T_P(Y) \subseteq X$ (we recall that $Y|_B = X|_B$) and so, $a \notin Y \setminus T_P(Y)$. Consequently, $Y \setminus T_P(Y) \subseteq Y \setminus \{a\}$. Hence, the condition (5) holds and $(Y \setminus \{a\}, Y) \in Mod^B_{At}(P)$. On the other hand, $a \in Y \setminus T_Q(Y)$ and $a \notin Y \setminus \{a\}$. Thus, the condition (5) for $(Y \setminus \{a\}, Y) \in Mod^B_{At}(Q)$ does not hold and so, $(Y \setminus \{a\}, Y) \notin Mod^B_{At}(Q)$. □

Next, we present proofs of the technical results needed in Section 6: Lemmas 12 and 13. First, we establish some auxiliary results. We start with conditions providing conditions restricting X and Y given that $(X, Y) \in SE^B_A(P)$.

Lemma 19
Let P be a program and $A, B \subseteq At$. If $(X, Y) \in SE^B_A(P)$ then $X \subseteq Y \subseteq At(P) \cup A$.

Proof
Let $(X, Y) \in SE^B_A(P)$. The inclusion $X \subseteq Y$ follows from the condition (2). To prove $Y \subseteq At(P) \cup A$, let us assume to the contrary that $Y \setminus (At(P) \cup A) \neq \emptyset$. Let $y \in Y \setminus (At(P) \cup A)$. We have $Y \models P$ and thus $Y \models P^Y$. Since $y \notin At(P)$, $y \notin At(P^Y)$. Thus, $Y \setminus \{y\} \models P^Y$. Since $y \notin A$, taking $Z = Y \setminus \{y\}$ shows that (X, Y) violates the condition (3) for $(X, Y) \in SE^B_A(P)$, a contradiction. □

The next two lemmas show that some atoms are immaterial for the membership of a pair (X, Y) in $SE^B_A(P)$.

Lemma 20
Let P be a program, $A, B, X, Y \subseteq At$, $y \in (X \cap Y) \setminus At(P)$, and $y \in A$. Then $(X, Y) \in SE^B_A(P)$ if and only if $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$.

Proof
We will show that each of the conditions (1) - (5) for $(X, Y) \in SE^B_A(P)$ is equivalent to its counterpart for $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$.

The case of the condition (1) is clear. Since $y \notin At(P)$, $Y \models P$ if and only if $Y \setminus \{y\} \models P$. It is also evident that $X = Y$ if and only if $X \setminus \{y\} = Y \setminus \{y\}$, $X \subseteq Y|_{A \cup B}$ if and only if $X \setminus \{y\} \subseteq (Y \setminus \{y\})|_{A \cup B}$, and $X|_A \subset Y|_A$ if and only if $(X \setminus \{y\})|_A \subset (Y \setminus \{y\})|_A$. Thus, the corresponding conditions (2) are also equivalent.

Let us assume the condition (3) for $(X, Y) \in SE^B_A(P)$. Let $Z \subset Y \setminus \{y\}$ be such that $Z|_A = (Y \setminus \{y\})|_A$. Let $Z' = Z \cup \{y\}$. Then $Z' \subset Y$ and $Z'|_A = Y|_A$ (as $y \in Y$). By the condition (3) for $(X, Y) \in SE^B_A(P)$, $Z' \not\models P^Y$. Since $y \notin At(P)$, $Z \not\models P^{Y \setminus \{y\}}$, and so, the condition (3) for $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$ follows. Conversely, let us assume the condition (3) for $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$ and let $Z \subset Y$ be such that $Z|_A = Y|_A$. It follows that $y \in Z$. We set $Z' = Z \setminus \{y\}$. Clearly, $Z' \subset Y \setminus \{y\}$ and $Z'|_A = (Y \setminus \{y\})|_A$. Thus, $Z' \not\models P^{Y \setminus \{y\}}$. As $y \notin At(P)$, $Z \not\models P^Y$ and, so, the condition (3) for $(X, Y) \in SE^B_A(P)$ follows.

Next, let us assume the condition (4) for $(X, Y) \in SE^B_A(P)$. Let $Z \subset Y \setminus \{y\}$ be such that $Z|_B \subset (X \setminus \{y\})|_B$ and $Z|_A \supseteq (X \setminus \{y\})|_A$, or $Z|_B \subseteq (X \setminus \{y\})|_B$ and $Z|_A \supset (X \setminus \{y\})|_A$. Let $Z' = Z \cup \{y\}$. We have $Z' \subset Y$. Moreover, it is evident that $Z'|_B \subset X|_B$ and $Z'|_A \supseteq X|_A$, or $Z'|_B \subseteq X|_B$ and $Z'|_A \supset X|_A$. Thus, $Z' \not\models P^Y$ and so, $Z \not\models P^{Y\setminus\{y\}}$. Similarly, let the condition (4) for $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$ hold. Let $Z \subset Y$ be such that $Z|_B \subset X|_B$ and $Z|_A \supseteq X|_A$, or $Z|_B \subseteq X|_B$ and $Z|_A \supset X|_A$. Since $y \in X$ and $y \in A$, $y \in Z$. We define $Z' = Z \setminus \{y\}$ and note that $Z' \subset Y \setminus \{y\}$. Moreover, as $y \in X$ and $y \in Y$, $Z'|_B \subset (X \setminus \{y\})|_B$ and $Z'|_A \supseteq (X \setminus \{y\})|_A$, or $Z'|_B \subseteq (X \setminus \{y\})|_B$ and $Z'|_A \supset (X \setminus \{y\})|_A$. Thus, $Z' \not\models P^{Y\setminus\{y\}}$ and so, $Z \not\models P^Y$.

Finally, a similar argument works also for the condition (5). Let the condition (5) for $(X, Y) \in SE^B_A(P)$ hold. Thus, there is $Z \subseteq Y$ such that $X|_{A\cup B} = Z|_{A\cup B}$ and $Z \models P^Y$. Let $Z' = Z \setminus \{y\}$. Since $y \in X$ and $y \in A$, $y \in Z$. Thus, $Z' \subseteq Y \setminus \{y\}$ and $(X \setminus \{y\})|_{A\cup B} = Z'|_{A\cup B}$. Moreover, since $Z \models P^Y$, $Z' \models P^{Y\setminus\{y\}}$. Conversely, let the condition (5) for $(X \setminus \{y\}, Y \setminus \{y\}) \in SE^B_A(P)$ hold. Then, there is $Z \subseteq Y \setminus \{y\}$ such that $Z|_{A\cup B} = (X \setminus \{y\})|_{A\cup B}$ and $Z \models P^{Y\setminus\{y\}}$. Let $Z' = Z \cup \{y\}$. Then $Z' \subseteq Y$, $Z'|_{A\cup B} = X|_{A\cup B}$ and $Z' \models P^Y$. $\square$

Lemma 21
Let P be a program, $A, B, X, Y \subseteq At$ and $y \in (Y \setminus (X \cup At(P))) \cap A$. If $|(Y \setminus (X \cup At(P))) \cap A| > 2$, then $(X, Y) \in SE^B_A(P)$ if and only if $(X, Y \setminus \{y\}) \in SE^B_A(P)$.

Proof
Since $|(Y \setminus (X \cup At(P))) \cap A| > 2$, there are $y', y'' \in (Y \setminus (X \cup At(P))) \cap A$ such that y, y', y'' are all distinct. As before, we will show that each of the conditions (1) - (5) for $(X, Y) \in SE^B_A(P)$ is equivalent to its counterpart for $(X, Y \setminus \{y\}) \in SE^B_A(P)$.

The case of the condition (1) is evident. By our assumptions, neither $X = Y$ nor $X = Y \setminus \{y\}$. Moreover, $X \subseteq Y|_{A\cup B}$ if and only if $X \subseteq (Y \setminus \{y\})|_{A\cup B}$ and $X|_A \subset Y|_A$ if and only if $X|_A \subset (Y \setminus \{y\})|_A$ (since $y, y' \in Y$ and $y, y' \in A$). Thus, the corresponding versions of the condition (2) are also equivalent. The case of the condition (3) can be argued in the same way as it was in Lemma 20.

Let us assume the condition (4) for $(X, Y) \in SE^B_A(P)$. Let $Z \subset Y \setminus \{y\}$ be such that $Z|_B \subset X|_B$ and $Z|_A \supseteq X|_A$, or $Z|_B \subseteq X|_B$ and $Z|_A \supset X|_A$. Clearly, $Z \subset Y$. Consequently, by the condition (4) for $(X, Y) \in SE^B_A(P)$, $Z \not\models P^Y$ and so, $Z \not\models P^{Y\setminus\{y\}}$. Thus the condition (4) for $(X, Y \setminus \{y\}) \in SE^B_A(P)$ holds.

Conversely, let the condition (4) for $(X, Y \setminus \{y\}) \in SE^B_A(P)$ hold. Let $Z \subset Y$ be such that $Z|_B \subset X|_B$ and $Z|_A \supseteq X|_A$, or $Z|_B \subseteq X|_B$ and $Z|_A \supset X|_A$. If $Z \subset Y \setminus \{y\}$, then $Z \not\models P^{Y\setminus\{y\}}$ (as the condition (4) for $(X, Y \setminus \{y\}) \in SE^B_A(P)$ holds). Thus, $Z \not\models P^Y$. Otherwise, i.e. for $Z = Y \setminus \{y\}$, we have $y', y'' \in Z$. Let $Z' = Z \setminus \{y, y'\}$. It follows that $Z' \subset Y \setminus \{y\}$ and $Z'|_A \supset X|_A$ (the former, as $y' \in Y \setminus \{y\} \setminus Z'$; the later, as $y'' \in Z'|_A \setminus X|_A$). Thus, $Z' \subset Y \setminus \{y\}$, $Z'|_B \subseteq X|_B$ and $Z'|_A \supset X|_A$. Consequently, $Z' \not\models P^{Y\setminus\{y\}}$ (again, as the condition (4) for $(X, Y \setminus \{y\}) \in SE^B_A(P)$ holds). Thus, also in that case, $Z \not\models P^Y$. It follows that the condition (4) for $(X, Y) \in SE^B_A(P)$ holds.

Finally, for the condition (5) we reason as follows. Let the condition (5) for

$(X, Y) \in SE_A^B(P)$ hold. Thus, there is $Z \subseteq Y$ such that $X|_{A\cup B} = Z|_{A\cup B}$ and $Z \models P^Y$. Clearly, $y \notin Z$ (as $y \notin X$ and $y \in A$). Thus, $Z \subseteq Y \setminus \{y\}$ and so $Z \models P^Y$ follows. Conversely, let the condition (5) for $(X, Y \setminus \{y\}) \in SE_A^B(P)$ hold. Then, there is $Z \subseteq Y \setminus \{y\}$ such that $Z|_{A\cup B} = X|_{A\cup B}$ and $Z \models P^Y$. Clearly, we also have $Z \subseteq Y$ and so, the condition (5) for $(X, Y) \in SE_A^B(P)$ follows. □

Finally, we note that the membership of a pair (X, Y), where $X \subseteq At(P)$, in $SE_{A^c}^B(P)$ does not depend on specific elements in $Y \setminus At(P)$ but only on their number.

Lemma 22
Let P be a program, $A, B \subseteq At$, $X, Y \subseteq At(P)$, and $Y', Y'' \subseteq A \setminus At(P)$. If $|Y'| = |Y''|$ then $(X, Y \cup Y') \in SE_A^B(P)$ if and only if $(X, Y \cup Y'') \in SE_A^B(P)$.

Proof
It is clear that the corresponding conditions (1) - (5) for $(X, Y \cup Y') \in SE_A^B(P)$ and $(X, Y \cup Y'') \in SE_A^B(P)$, respectively are equivalent to each other. □

Lemmas 19 - 22 allow us to prove Lemma 12.

Lemma 12
Let P, Q be programs and $A, B \subseteq At$. If $(X, Y) \in SE_A^B(P) \setminus SE_A^B(Q)$, then there are sets $X', Y' \subseteq At(P \cup Q)$, such that at least one of the following conditions holds:

i. $(X', Y') \in SE_A^B(P) \setminus SE_A^B(Q)$
ii. $A \setminus At(P \cup Q) \neq \emptyset$ *and for every* $y, z \in A \setminus At(P \cup Q)$, $(X', Y' \cup \{y, z\}) \in SE_A^B(P) \setminus SE_A^B(Q)$.

Proof
Since $(X, Y) \in SE_A^B(P)$, $X \subseteq Y \subseteq At(P) \cup A$ (cf. Lemma 19). Thus, $X \subseteq Y \subseteq At(P \cup Q) \cup A$.

By applying repeatedly Lemma 20 and then Lemma 21, we can construct sets $X' \subseteq At(P \cup Q)$ and $Y'' \subseteq A \cup At(P \cup Q)$ such that

a. $(X', Y'') \in SE_A^B(P) \setminus SE_A^B(Q)$, and
b. $|Y'' \setminus At(P \cup Q)| \leq 2$.

If $Y'' \subseteq At(P \cup Q)$, (i) follows (with $Y' = Y''$). Otherwise, (ii) follows (by Lemma 22). □

Next we present a proof of Lemma 13

Lemma 13
Let P be a program and $A, B \subseteq At$. If $At(P) \subseteq A$ and $At(P) \cap B = \emptyset$, then $(X, Y) \in SE_A^B(P)$ if and only if there are $X', Y' \subseteq At(P)$ and $W \subseteq A \setminus At(P)$ such that one of the following conditions holds:

i. $X = X' \cup W$, $Y = Y' \cup W$, *and* $(X', Y') \in SE_A^B(P)$

ii. $X = X' \cup W$, $(X', X') \in SE_A^B(P)$ *and* $Y = X' \cup W \cup \{y\}$, *for some* $y \in A \setminus At(P)$

iii. $X = X' \cup W$, $(X', X') \in SE_A^B(P)$ *and* $Y = X' \cup W \cup D$, *for some* $D \subseteq B \cap (A \setminus At(P))$ *such that* $W \cap D = \emptyset$ *and* $|D| \geq 2$.

Proof

($\Leftarrow$) If (i) holds, then $(X, Y) \in SE_A^B(P)$ follows from Lemma 20. Thus, let us assume that (ii) or (iii) holds. Then $X' \models P$ and so, $X' \cup \{y\} \cup W \models P$ (respectively, $X' \cup W \cup D \models P$). Moreover, $X \subset Y$. Thus, since $Y \subseteq A$, the condition (2) for $(X, Y) \in SE_A^B(P)$ holds. Next, it is evident that the condition (3) is vacuously true. The condition (4) is also vacuously true. To see it, let us consider $Z \subset Y$ such that $Z|_B \subset X|_B$ and $Z|_A \supseteq X|_A$, or $Z|_B \subseteq X|_B$ and $Z|_A \supset X|_A$. Since $X \subseteq Y \subseteq A$, $X \subseteq Z$. Thus, $X|_B \subseteq Z|_B$, and so $Z|_B \subset X|_B$ is impossible. Consequently, $Z|_B \subseteq X|_B$ and $Z|_A \supset X|_A$. The latter implies $X \subset Z$. We also have $Z \subset Y$. Thus, $|Y \setminus X| \geq 2$, contradicting (ii). It follows that (iii) holds. Consequently, $X' \cup W \subset Z \subset X' \cup W \cup D$. Since $Z|_B \subseteq X|_B$, $D = \emptyset$, a contradiction.

Finally, let Z be a set verifying the condition (5) for $(X', X') \in SE_A^B(P)$ (which holds under either (ii) or (iii)). Clearly, the set $Z \cup W$ demonstrates that the condition (5) for $(X, Y) \in SE_A^B(P)$ holds.

($\Rightarrow$) Let $W = X \cap (A \setminus At(P))$. We define $X' = X \setminus W$ and $Y' = Y \setminus W$. Clearly, $X' \subseteq At(P)$. Moreover, by Lemma 20, $(X', Y') \in SE_A^B(P)$. If $Y' \subseteq At(P)$, then (i) follows.

Thus, let us assume that $Y' \setminus At(P) \neq \emptyset$. Next, let us assume that $X' \subset Y' \cap At(P)$ and let $Z = Y' \cap At(P)$. Clearly, $Z \subset Y'$, $Z|_B = \emptyset$ and $X|_A = X \subset Z = Z|_A$. By the condition (4) for $(X', Y') \in SE_A^B(P)$, $Z \not\models P^{Y'}$. On the other hand, by the condition (1) for $(X', Y') \in SE_A^B(P)$, $Y' \models P$. Consequently, $Y' \models P^{Y'}$. It follows that $Z \models P^{Y'}$, a contradiction.

It follows that $X' = Y' \cap At(P)$. If there are $y', y'' \in Y' \setminus At(P)$ such that $y' \neq y''$ and $y' \notin B$, then let us define $Z = X' \cup \{y'\}$. It is easy to verify that Z contradicts the condition (4). If $|Y' \setminus At(P)| = 1$, then (ii) follows (with the only element of $Y' \setminus At(P)$ as y). Otherwise, $|Y' \setminus At(P)| \geq 2$ and $Y' \setminus At(P) \subseteq B$. In this case, (iii) follows (with $D = Y' \setminus At(P)$). □

References

Apt, K. 1990. Logic programming. In *Handbook of theoretical computer science*, J. van Leeuven, Ed. Elsevier, Amsterdam, 493–574.

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

Brass, S. and Dix, J. 1997. Characterizations of the disjunctive stable semantics by partial evaluation. *Journal of Logic Programming 32(3)*, 207–228.

Cabalar, P., Odintsov, S., Pearce, D., and Valverde, A. 2006. Analysing and extending well-founded and partial stable semantics using partial equilibrium logic. In *Proceedings of the 22nd International Conference (ICLP 2006)*, S. Etalle and M. Truszczyński, Eds. LNCS, vol. 4079. Springer, Berlin, New York, 346–360.

Clark, K. 1978. Negation as failure. In *Logic and data bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York-London, 293–322.

de Jongh, D. and Hendriks, L. 2003. Characterizations of strongly equivalent logic programs in intermediate logics. *Theory and Practice of Logic Programming 3,* 3, 259–270.

Eiter, T. and Fink, M. 2003. Uniform equivalence of logic programs under the stable model semantics. In *Proceedings of the 19th International Conference on Logic Programming (ICLP 2003)*, C. Palamidessi, Ed. LNCS, vol. 2916. Springer, Berlin, New York, 224–238.

Eiter, T., Fink, M., and Woltran, S. 2007. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Transactions on Computational Logic 8,* 3. 53 pages.

Eiter, T. and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence 15,* 3/4, 289–323.

Eiter, T., Tompits, H., and Woltran, S. 2005. On solution correspondences in answer-set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005)*, L. P. Kaelbling and A. Saffiotti, Eds. Professional Book Center, 97–102.

Erdogan, S. and Lifschitz, V. 2004. Definitions in answer set programming: (extended abstract). In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2004)*, V. Lifschitz and I. Niemelä, Eds. LNCS, vol. 2916. Springer, Berlin, New York, 483–484.

Gaifman, H. and Shapiro, E. 1989. Fully abstract compositional semantics for logic programs. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages (POPL 1989)*. 134–142.

Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., and Truszczyński, M. 2007. The first answer set programming system competition. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, C. Baral, G. Brewka, and J. Schlipf, Eds. LNCS, vol. 4483. Springer, Berlin, New York, 3–17.

Gelfond, M. 2002. Representing knowledge in A-Prolog. In *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert A. Kowalski, Part II*, A. Kakas and F. Sadri, Eds. LNCS, vol. 2408. Springer, Berlin, New York, 413–451.

Gelfond, M. and Leone, N. 2002. Logic programming and knowledge representation – the A-prolog perspective. *Artificial Intelligence 138*, 3–38.

Inoue, K. and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming 35*, 39–78.

Inoue, K. and Sakama, C. 2004. Equivalence of logic programs under updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, J. Alferes and J. Leite, Eds. LNCS, vol. 3229. Springer, Berlin, New York, 174–186.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics 16,* 1-2, 35–86.

Janhunen, T., Oikarinen, E., Tompits, H., and Woltran, S. 2007. Modularity aspects of disjunctive stable models. In *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2007)*, C. Baral, G. Brewka, and J. Schlipf, Eds. LNAI, vol. 4483. Springer, Berlin, New York, 175–187.

Lifschitz, V., Pearce, D., and Valverde, A. 2001. Strongly equivalent logic programs. *ACM Transactions on Computational Logic 2(4)*, 526–541.

Lifschitz, V. and Turner, H. 1994. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming (ICLP 1994)*, P. V. Hentenryck, Ed. MIT Press, 23–37.

Lin, F. 2002. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, D. Fensel, D. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, 170–176.

Marek, V. and Truszczyński, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, K. Apt, W. Marek, M. Truszczyński, and D. Warren, Eds. Springer, Berlin, New York, 375–398.

Niemelä, I. 1999. Logic programming with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3-4, 241–273.

Oetsch, J., Tompits, H., and Woltran, S. 2007. Facts do not cease to exist because they are ignored: Relativised uniform equivalence with answer-set projection. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*. AAAI Press, 458–464.

Oikarinen, E. and Janhunen, T. 2006. Modular equivalence for normal logic programs. In *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI 2006*), G. Brewka, S. Coradeschi, A. Perini, and P. Traverso, Eds. IOS Press, Amsterdam, 412–416.

Sagiv, Y. 1988. Optimizing datalog programs. In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, 659–698.

Truszczyński, M. and Woltran, S. 2008. Hyperequivalence of logic programs with respect to supported models. *Annals of Mathematics and Artificial Intelligence 53,* 1-4, 331–365.

Turner, H. 2003. Strong equivalence made easy: nested expressions and weight constraints. *Theory and Practice of Logic Programming 3*, 609–622.

Woltran, S. 2004. Characterizations for relativized notions of equivalence in answer set programming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, J. Alferes and J. Leite, Eds. LNCS, vol. 3229. Springer, Berlin, New York, 161–173.

Woltran, S. 2008. 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.