A Generalised Program-Correspondence Framework: Preliminary Report[⋆]

Johannes Oetsch and Hans Tompits

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

Abstract. The study of various notions of equivalence between logic programs in the area of answer-set programming (ASP) gained increasing interest in recent years. The main reason for this undertaking is the failure of ordinary equivalence between answer-set programs to yield a replacement property similar to the one of classical logic. Although many refined program correspondence notions have been introduced in the ASP literature so far, most of these notions were studied for propositional programs only, which limits their practical usability as concrete programming applications require the use of variables. In this paper, we address this issue and introduce a general framework for specifying parameterised notions of program equivalence for non-ground disjunctive logic programs under the answer-set semantics. Our framework is a generalisation of a similar one defined previously for the propositional case and allows the specification of several equivalence notions extending well-known ones studied for propositional programs. We provide semantic characterisations for instances of our framework generalising uniform equivalence, and we study decidability and complexity aspects.

1 Introduction

Logic programs under the answer-set semantics are an established means for declarative knowledge representation and nonmonotonic reasoning as well as for declarative problem solving. Their characteristic feature regarding the way problems are represented, viz. that *models* represent solutions to problems and not *proofs* as in traditional logic-oriented languages (hence, the intended models are the "answer sets" for the problem), led to the coinage of the term *answer-set programming* (ASP) for this particular paradigm. Several sophisticated ASP solvers exist, like DLV [1] or GNT [2], and typical application areas of ASP are configuration, information integration, planning, and the Semantic Web.

A recent line of research in ASP deals with the investigation of different notions of equivalence between answer-set programs, initiated by the seminal paper by Lifschitz, Pearce, and Valverde [3] on *strong equivalence*. The main reason for the introduction of these notions is the fact that *ordinary equivalence*, which checks whether two programs have the same answer sets, is too weak to yield a replacement property similar to the one of classical logic. That is to say, given a program R along with some subprogram $P \subseteq R$,

⋆ This work was partially supported by the Austrian Science Fund (FWF) under grant P18019.

when replacing P with an equivalent program Q it is not guaranteed that $Q \cup (R \setminus P)$ is equivalent to R. Clearly, this is undesirable for tasks like modular programming or program optimisation. Strong equivalence does circumvent this problem, essentially by definition—two programs P, Q are strongly equivalent iff for any program R (the "context program"), $P \cup R$ and $Q \cup R$ have the same answer sets—, but is too restrictive for certain aspects. A more liberal notion is *uniform equivalence* [4], which is defined similar to strong equivalence but where the context programs are restricted to contain facts only.[1] However, both notions do not take standard programming techniques like the use of local predicates into account, which may occur in some subprograms but which are ignored in the final computation. Thus, these notions do not admit the *projection* of answer sets to a set of output letters. For illustration, consider the two programs

$$P = \{r(x) \leftarrow s(x)\} \text{ and } Q = \{aux(x) \leftarrow s(x);\ r(x) \leftarrow aux(x)\}.$$

P expresses that $r(x)$ is selected when $s(x)$ is known, and Q yields the selection of $r(x)$ via the local predicate $aux(x)$. While P and Q are not uniformly equivalent (consider P and Q joined with any fact $aux(\cdot)$), they are so if *aux* does not occur in the context and is also ignored when comparing the respective answer sets.

To accommodate such features, strong and uniform equivalence were further relaxed and generalised. On the one hand, Woltran [7] introduced *relativised* versions thereof, where the alphabet of the context can be parameterised (e.g., in the above example, we could specify contexts disallowing the use of the predicate symbol *aux*). On the other hand, Eiter *et al.* [8] introduced a general framework for specifying parameterisable program correspondence notions, allowing not only relativised contexts but also answer-set projection. Other forms of refined program equivalence are, e.g., *modular equivalence* [9] and *update equivalence* [10].

However, most of the above mentioned notions were introduced and analysed for propositional programs only, which is clearly a limiting factor given that practical programming purposes require the use of variables.

In this paper, we address this point and introduce a general program correspondence framework for the non-ground case, lifting the propositional one by Eiter *et al.* [8]. Similar to the latter, program-correspondence notions can be parameterised along two dimensions: one for specifying the kind of programs that are allowed as context for program comparison, and one for specifying a particular comparison relation between answer sets, determining which predicates should be considered for the program comparison. The framework allows to capture the most important notions of program equivalence, including the propositional ones mentioned above as well as non-ground versions of strong and uniform equivalence studied already in the literature [11–13].

We pay particular attention to correspondence problems which we refer to as *generalised query inclusion problems* (GQIPs) and *generalised query equivalence problems* (GQEPs), respectively. These are extensions of notions studied in previous work [14] for propositional programs (there termed PQIPs and PQEPs, standing for *propositional query inclusion problems* and *propositional query equivalence problems*, respectively).

[1] We note that the concept of strong equivalence was actually first studied in the context of (negation free) datalog programs by Maher [5] using the term "equivalence of program segments", and likewise uniform equivalence was first studied for datalog programs by Sagiv [6].

A GQEP basically amounts to *relativised uniform equivalence with projection*, whilst in a GQIP, set equality is replaced by set inclusion. Intuitively, if Q corresponds to P under a GQIP, then Q can be seen as a sound approximation of P under brave reasoning and P as a sound approximation of Q under cautious reasoning. GQEPs and GQIPS are relevant in a setting where programs are seen as queries over databases and one wants to decide whether two programs yield the same output-database on each input-database. Indeed, query equivalence as well as program equivalence emerge as special cases. Also, GQEPs and GQIPs are appropriate in cases where programs are composed out of layered modules, where each module of a certain layer gets its input from higher-layered modules and provides its output to lower-layered modules.

We provide semantic characterisations of GQIPs and GQEPs in terms of structures associated with each program such that a GQIP holds iff the structures meet set inclusion, and a GQEP holds iff the associated structures coincide. Our characterisation differs from the well-known characterisation of (relativised) uniform equivalence in terms of (relativised) UE-models [4, 7] in case the projection set is unrestricted. Furthermore, we study decidability and complexity issues, showing in particular that relativised strong equivalence is undecidable, which is, in some sense, surprising since usual (unrelativised) strong equivalence is known to be decidable.

2 Preliminaries

We deal with *disjunctive logic programs* (DLPs) formulated in a function-free first-order language. Such a language is based on a vocabulary $\mathscr{V}$ defined as a pair $(\mathbb{P}, \mathbb{D})$, where $\mathbb{P}$ fixes a countable set of predicate symbols and $\mathbb{D}$, called the *domain*, fixes a countable and non-empty set of constants. As usual, each $p \in \mathbb{P}$ has an associated non-negative integer, called the *arity* of p. An *atom over* $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ is an expression of form $p(t_1, \dots, t_n)$, where p is a n-ary predicate symbol from $\mathbb{P}$ and each t_i, $0 \leq i \leq n$, is either from $\mathbb{D}$ or a variable. An atom is *ground* if it does not contain variables. By $HB_{\mathscr{V}}$ we denote the *Herbrand base* of $\mathscr{V}$, i.e., the set of all ground atoms over $\mathscr{V}$. For a set $A \subseteq \mathbb{P}$ of predicate symbols and a set $C \subseteq \mathbb{D}$ of constants, we also write $HB_{A,C}$ to denote the set of all ground atoms constructed from the predicate symbols from $A \subseteq \mathbb{P}$ and the constants from C. A *rule over* $\mathscr{V}$ is an expression of form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \dots, a_m, \text{not } a_{m+1}, \dots, \text{not } a_n, \qquad (1)$$

where $n \geq m \geq l \geq 0$, $n > 0$, and all a_i, $0 \leq i \leq n$, are atoms over $\mathscr{V}$. A rule is *positive* if $m = n$, *normal* if $l = 1$, and *Horn* if it is positive and normal. A rule of form (1) with $l = m = n = 1$ is a fact; we usually identify a fact $a \leftarrow$ with the atom a. For each rule r of form (1), we call $\{a_1, \dots, a_l\}$ the *head of* r. Moreover, we call $\{a_{l+1}, \dots, a_m\}$ the *positive body of* r and $\{a_{m+1}, \dots, a_n\}$ the *negative body of* r. In case $l = 0$, we identify the head of the rule with the symbol $\perp$. A rule r is *safe* iff all variables that occur in the head of r or in the negative body of r also occur in the positive body of r. Finally, a rule is ground if it contains only ground atoms.

A program, P, is a finite set of safe rules over $\mathscr{V}$. We say that P is positive (resp., normal, Horn, ground) if all rules in P are positive (resp., normal, Horn, ground). Sometimes, we will call ground programs (resp., rules, atoms) *propositional*. For any program P over $\mathscr{V}$, we call a predicate symbol p that does not occur in the head of any rule (excluding facts) in P *extensional* (in P) and we call p *intensional* otherwise. The *Herbrand universe* of P, HU_P, is the set of all constant symbols occurring in P. For technical reasons, we assume that HU_P contains an arbitrary element from the domain of $\mathscr{V}$ in case P does not contain any constant symbol.

For any set $C \subseteq \mathbb{D}$, we define the *grounding of a rule r with respect to C*, denoted by $grd(r,C)$, as the set of all rules that can be obtained from r by uniformly replacing all variables occurring in r by constants from C. Furthermore, the *grounding of a program P with respect to C*, denoted by $grd(P,C)$, is $\bigcup_{r\in P} grd(r,C)$. An *interpretation I* over a vocabulary $\mathscr{V}$ is a set of ground atoms over $\mathscr{V}$. Moreover, I is a *model* of a ground rule r, symbolically $I \models r$, iff, whenever the positive body of r is a subset of I and no elements from the negative body of r are in I, then some element from the head of r is in I. I is a model of a ground program P, in symbols $I \models P$, iff $I \models r$, for all r in P.

Following Gelfond and Lifschitz [15], the *reduct P^I*, where I is an interpretation and P is a ground program, is the program obtained from P by (i) deleting all rules containing a default negated atom not a such that $a \in I$ and (ii) deleting all default negated atoms in the remaining rules. An interpretation I is an *answer set* of a ground program P iff I is a subset-minimal model of the reduct P^I. The answer sets of a non-ground program P are the answer sets of the grounding of P with respect to the Herbrand universe of P. The collection of all answer sets of a program P is denoted by $AS(P)$.

For a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$ and a set $A \subseteq \mathbb{P}$, by $\mathscr{P}^A_{\mathscr{V}}$ we denote the set of all programs over $\mathscr{V}$ that contain only predicate symbols from A. The set $\mathscr{F}^A_{\mathscr{V}}$ is defined analogously except that the programs are additionally required to contain only facts. Note that since we assume safety of rules in programs, the facts in the elements of $\mathscr{F}^A_{\mathscr{V}}$ are therefore ground. If we omit the set A, we assume that $A = \mathbb{P}$. In the sequel, we use the following notation: For an interpretation I and a set S of interpretations, $S|_I$ is defined as $\{Y \cap I \mid Y \in S\}$. For a singleton set $S = \{Y\}$, we also write $Y|_I$ instead of $S|_I$.

We recall some basic equivalence notions for logic programs. To begin with, two programs P and Q are *ordinarily equivalent* iff $AS(P) = AS(Q)$. The more restrictive notions of *strong* and *uniform equivalence*, originally defined for propositional programs by Lifschitz, Pearce, and Valverde [3] and Eiter and Fink [4], respectively, are given as follows [12]: Let $\mathscr{V}$ be a vocabulary and P, Q two finite programs over $\mathscr{V}$. Then, P and Q are strongly equivalent iff $AS(P \cup R) = AS(Q \cup R)$, for any program $R \in \mathscr{P}_{\mathscr{V}}$, and P and Q are uniformly equivalent iff $AS(P \cup F) = AS(Q \cup F)$, for any $F \in \mathscr{F}_{\mathscr{V}}$. We also recall two further equivalence notions, especially important in the context of deductive databases: *query equivalence* and *program equivalence*. Let $\mathscr{E}$ be the set of the extensional predicates of $P \cup Q$. Then, P and Q are query equivalent with respect to a predicate p iff, for any set $F \in \mathscr{F}^{\mathscr{E}}_{\mathscr{V}}$, $AS(P \cup F)|_{HB_{\{p\},\mathbb{D}}} = AS(Q \cup F)|_{HB_{\{p\},\mathbb{D}}}$. Furthermore, P and Q are program equivalent iff, for any set $F \in \mathscr{F}^{\mathscr{E}}_{\mathscr{V}}$, $AS(P \cup F) = AS(Q \cup F)$.

3 A Unifying Correspondence-Framework

In this section, we introduce the central concept of our work, a general framework for specifying parameterised notions of program correspondence for non-ground programs, which is a lifting of a correspondence framework introduced by Eiter *et al.* [8] for propositional programs to the non-ground setting. This framework allows to capture a range of different equivalence notions defined in the literature in a uniform manner. We first discuss the basic definitions and some elementary properties, afterwards we concentrate on a specific instance of our method, namely on correspondence notions generalising uniform equivalence. We then provide model-theoretic characterisations for this particular family of correspondences.

Definition 1. *By a* correspondence frame*, or simply a* frame*, F, we understand a triple $(\mathscr{V},\mathscr{C},\rho)$, where (i) $\mathscr{V}$ is a vocabulary, (ii) $\mathscr{C} \subseteq \mathscr{P}_{\mathscr{V}}$, called the context class of F, and (iii) $\rho \subseteq 2^{2^{HB_{\mathscr{V}}}} \times 2^{2^{HB_{\mathscr{V}}}}$. For every program $P,Q \in \mathscr{P}_{\mathscr{V}}$, we say that P and Q are* F-corresponding*, symbolically $P \simeq_F Q$, iff, for all $R \in \mathscr{C}$, $(AS(P \cup R), AS(Q \cup R)) \in \rho$.*

In a frame $F = (\mathscr{V},\mathscr{C},\rho)$, $\mathscr{V}$ fixes the language under consideration, $\mathscr{C}$ is the class of possible extensions of the programs that are to be compared, and ρ represents the comparison relation between the sets of answer-sets of the two programs.

Following Eiter *et al.* [8], a *correspondence problem*, Π, over a vocabulary $\mathscr{V}$ is a tuple $(P,Q,\mathscr{C},\rho)$, where P and Q are programs over $\mathscr{V}$ and $(\mathscr{V},\mathscr{C},\rho)$ is a frame. We say that $(P,Q,\mathscr{C},\rho)$ *holds* iff $P \simeq_{(\mathscr{V},\mathscr{C},\rho)} Q$.

Important instances of frames are those where the comparison relation is given by projective versions of set equality and set inclusion, where projection allows to ignore auxiliary ("local") atoms during comparison, and the context class is parameterised by a set of predicate symbols. More formally, for sets S,S' of interpretations, an interpretation B, and $\odot \in \{\subseteq,=\}$, we define $S \odot_B S'$ as $S|_B \odot S'|_B$. For any vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$ and any set $B \subseteq \mathbb{P}$, the relation $\sqsubseteq^B_{\mathscr{V}}$ is defined as $\subseteq_{HB_{B,\mathbb{D}}}$, and $\equiv^B_{\mathscr{V}}$ denotes $=_{HB_{B,\mathbb{D}}}$. If B is omitted, this coincides with the case $B = \mathbb{P}$. We call a correspondence problem over $\mathscr{V}$ of form $(P,Q,\mathscr{P}^A_{\mathscr{V}},\sqsubseteq^B_{\mathscr{V}})$ an *inclusion problem* and one of form $(P,Q,\mathscr{P}^A_{\mathscr{V}},\equiv^B_{\mathscr{V}})$ an *equivalence problem*, for $A,B \subseteq \mathbb{P}$. Furthermore, for a correspondence problem of form $\Pi = (P,Q,\mathscr{C},\equiv^B_{\mathscr{V}})$, we set $\Pi_{\sqsubseteq} =_{df} (P,Q,\mathscr{C},\sqsubseteq^B_{\mathscr{V}})$ and $\Pi_{\sqsupseteq} =_{df} (Q,P,\mathscr{C},\sqsubseteq^B_{\mathscr{V}})$. Clearly, Π holds iff $\Pi_{\sqsubseteq}$ and $\Pi_{\sqsupseteq}$ jointly hold.

Example 1. Consider the programs P and Q from the introduction. The claim is that, for any program M with $P \subseteq M$, P can be replaced by Q within M without changing the answer sets of M, provided $aux(\cdot)$ does not occur in M and is ignored regarding answer-set comparison. This is represented in our framework by the fact that the correspondence problem $\Pi = (P,Q,\mathscr{P}^A_{\mathscr{V}},\equiv^B_{\mathscr{V}})$, with $A = B = \{r,s\}$, holds. $\square$

With the above notation at hand, it is quite straightforward to express the equivalence notions introduced earlier:

Theorem 1. *Let P,Q be programs over a vocabulary $\mathscr{V}$ and $\mathscr{E}$ the set of the extensional predicates of $P \cup Q$. Then, P and Q are*

- *ordinarily equivalent iff $(P,Q,\{\emptyset\},=)$ holds,*
- *strongly equivalent iff $(P,Q,\mathscr{P}_{\mathscr{V}},=)$ holds,*
- *uniformly equivalent iff $(P,Q,\mathscr{F}_{\mathscr{V}},=)$ holds,*
- *query equivalent with respect to a predicate p iff $(P,Q,\mathscr{F}^{\mathscr{E}}_{\mathscr{V}},\equiv^{\{p\}}_{\mathscr{V}})$ holds, and*
- *program equivalent iff $(P,Q,\mathscr{F}^{\mathscr{E}}_{\mathscr{V}},=)$ holds.*

We mentioned that our definition of correspondence frames lifts that of Eiter *et al.* [8] introduced for propositional programs. Let us elaborate this point in more detail. A frame in the sense of Eiter *et al.* [8] (which we henceforth will refer to as *propositional*) is a tuple $F = (\mathscr{U}, \mathscr{C}, \rho)$, where $\mathscr{U}$ is a set of propositional atoms, $\mathscr{C}$ is a set of propositional programs over $\mathscr{U}$, and $\rho \subseteq 2^{2^{\mathscr{U}}} \times 2^{2^{\mathscr{U}}}$ is a comparison relation. As in our case, two programs P and Q are F-corresponding, in symbols $P \simeq_F Q$, iff, for any $R \in \mathscr{C}$, $(AS(P \cup R), AS(Q \cup R)) \in \rho$. Consider now a propositional frame $F = (\mathscr{U}, \mathscr{C}, \rho)$ and define the vocabulary $\mathscr{V}_0 = (\mathscr{U}, \mathbb{D})$, where the arity of each $p \in \mathscr{U}$ is 0, thus $\mathbb{D}$ has no influence on the language and can be fixed arbitrarily (but, of course, $\mathbb{D}$ is assumed to be non-empty). Then, $HB_{\mathscr{V}_0} = \mathscr{U}$, and thus $F_0 = (\mathscr{V}_0, \mathscr{C}, \rho)$ is a frame in the sense of Definition 1. Furthermore, for every program P, Q over $\mathscr{U}$, $P \simeq_F Q$ iff $P \simeq_{F_0} Q$.

Consequently, every equivalence notion expressible in terms of propositional frames is also expressible in terms of frames in our sense. Indeed, propositional frames capture the following notions from the literature defined for propositional programs only so far:

- *Relativised strong and uniform equivalence* [7]: Let $\mathscr{U}$ be a set of propositional atoms, $A \subseteq \mathscr{U}$, $\mathscr{P}^A$ the set of all propositional programs constructed from atoms in A, and $\mathscr{F}^A$ the subclass of $\mathscr{P}^A$ containing all programs over A comprised of facts only. Then, two programs P and Q over $\mathscr{U}$ are strongly equivalent relative to A iff they are $(\mathscr{U}, \mathscr{P}^A, =)$-corresponding. Moreover, they are uniformly equivalent relative to A iff they are $(\mathscr{U}, \mathscr{F}^A, =)$-corresponding. Clearly, if $A = \mathscr{U}$, relativised strong and uniform equivalence collapse to the usual versions of strong and uniform equivalence, respectively.
- *Propositional query equivalence and inclusion problems* [14]: Let $\mathscr{U}$ be a set of propositional atoms and $A, B \subseteq \mathscr{U}$. A propositional query inclusion problem (PQIP) over $\mathscr{U}$ is defined as a correspondence problem of form $(P, Q, \mathscr{F}^A, \subseteq_B)$ over $\mathscr{U}$, where $P, Q \in \mathscr{P}_{\mathscr{U}}$, and a propositional query equivalence problem (PQEP) over $\mathscr{U}$ is defined as a correspondence problem of form $(P, Q, \mathscr{F}^A, =_B)$ over $\mathscr{U}$. Clearly, for $B = \mathscr{U}$, the latter problems turn into checking uniform equivalence relative to A.

Hence, these notions are then just special cases of correspondence problems according to Definition 1. In what follows, we will in particular be interested in analysing non-ground pendants of PQIPs and PQEPs. We thus define:

Definition 2. *Let $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ be a vocabulary, $A, B \subseteq \mathbb{P}$, and $P, Q \in \mathscr{P}_{\mathscr{V}}$. Then, $(P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$ is a* generalised query inclusion problem (GQIP) over $\mathscr{V}$ *and* $(P, Q, \mathscr{F}^A_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$ *is a* generalised query equivalence problem (GQEP) over $\mathscr{V}$.

Hence, for a GQEP $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$, if $A = B = \mathbb{P}$, Π coincides with testing uniform equivalence between P and Q, and if just $B = \mathbb{P}$, Π amounts to testing the extension of relativised uniform equivalence for non-ground programs.

Example 2. Consider the following two programs:

$$P = \left\{ \begin{array}{l} sel(x) \vee nsel(x) \leftarrow s(x);\ \bot \leftarrow sel(x), sel(y), \text{not } eq(x,y); \\ some \leftarrow sel(x);\ \bot \leftarrow \text{not } some;\ eq(x,x) \leftarrow s(x) \end{array} \right\},$$

$$Q = \left\{ \begin{array}{l} nsel(x) \vee nsel(y) \leftarrow s(x), s(y), \text{not } eq(x,y); \\ sel(x) \leftarrow s(x), \text{not } nsel(x);\ eq(x,x) \leftarrow s(x) \end{array} \right\}.$$

Program P takes as input facts over s and non-deterministically selects one element of the set implicitly defined by s, i.e., each answer set of P contains exactly one fact $sel(c)$ where c is from that set. Program Q, being more compact than P, aims at the same task, but does it the same job for every input? To answer this question, let us consider the GQEP $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$, for $A = \{s\}$ and $B = \{sel\}$. Since $AS(P) = \emptyset$ but $AS(Q) = \{\emptyset\}$, it follows that Π does not hold. Having only *non-empty* input regarding s in mind, we can verify that $\Pi' = (P \cup \{s(a)\}, Q \cup \{s(a)\}, \mathscr{F}^A_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$ holds. $\square$

The next properties generalise similar results from the propositional setting [8].

Theorem 2 (Anti-Monotony). *Let F be a correspondence frame of form $(\mathscr{V}, \mathscr{C}, \equiv^B_{\mathscr{V}})$. If two programs are F-corresponding, then they are also F'-corresponding, for any frame $F' = (\mathscr{V}, \mathscr{C}', \equiv^{B'}_{\mathscr{V}})$ with $\mathscr{C}' \subseteq \mathscr{C}$ and $B' \subseteq B$.*

Theorem 3 (Projective Invariance). *Let $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ be a vocabulary and F a correspondence frame of form $(\mathscr{V}, \mathscr{P}_{\mathscr{V}}, =)$. If two programs are F-corresponding, then they are also F'-corresponding, for any frame $F' = (\mathscr{V}, \mathscr{P}_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$ with $B \subseteq \mathbb{P}$.*

Note that the last result states that strong equivalence coincides with strong equivalence with projection, no matter what projection set is used.

We now provide necessary and sufficient conditions for deciding GQIPs and GQEPs, based on model-theoretic concepts. This is along the lines of characterising strong equivalence in terms of *SE-models* [16] and of uniform equivalence in terms of *UE-models* [4]. However, our characterisations of GQIPs and GQEPs are based on structures which are, in some sense, orthogonal to SE- and UE-models, extending the concepts introduced in previous work for propositional programs [14].

We start with introducing objects witnessing the failure of a GQIP or GQEP.

Definition 3. *Let $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$ be a GQIP over some vocabulary $\mathscr{V}$ and X, Y interpretations over $\mathscr{V}$. Then, (X, Y) is a* counterexample *to Π iff (i) $X \in \mathscr{F}^A_{\mathscr{V}}$, (ii) $Y \in AS(P \cup X)$, and (iii) there exists no interpretation Z over $\mathscr{V}$ such that $Z \equiv^B_{\mathscr{V}} Y$ and $Z \in AS(Q \cup X)$. Furthermore, (X, Y) is a counterexample to a GQEP Π iff (X, Y) is a counterexample to one of $\Pi_{\sqsubseteq}$ or $\Pi_{\sqsupseteq}$.*

We note that, for any problem Π, Π does not hold iff some counterexample to Π exists.

Example 3. Recall problem Π from Example 2, which does not hold. The pair $(\emptyset, \emptyset)$ is a counterexample to $\Pi_{\sqsupseteq}$ since $AS(Q \cup \emptyset) = \{\emptyset\}$ but $AS(P \cup \emptyset) = \emptyset$. On the other hand, $\Pi' = (P \cup \{s(a)\}, Q \cup \{s(a)\}, \mathscr{F}^A_{\mathscr{V}}, \equiv^B_{\mathscr{V}})$ from the same example does hold and, accordingly, has no counterexamples. $\square$

Next, we introduce structures assigned to programs rather than to correspondence problems as counterexamples are, yielding our desired model-theoretic characterisation.

Definition 4. *Let $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ be a vocabulary, $A, B \subseteq \mathbb{P}$, P a program over $\mathscr{V}$, and X, Y interpretations over $\mathscr{V}$. Then, (X, Y) is an* A-B-wedge *of* P *over* $\mathscr{V}$ *iff $X \in \mathscr{F}^A_{\mathscr{V}}$ and $Y \in AS(P \cup X)|_{HB_{B,\mathbb{D}}}$. The set of all A-B-wedges of P is denoted by $\omega_{A,B}(P)$.*

It is straightforward to check that there exists a counterexample to a GQIP $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$ iff an A-B-wedge of P exists that is not an A-B-wedge of Q. Hence:

Theorem 4. *Consider $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \odot^B_{\mathscr{V}})$, for $\odot \in \{\sqsubseteq, \equiv\}$. Then, (i) Π holds iff $\omega_{A,B}(P) \subseteq \omega_{A,B}(Q)$, if $\odot$ is $\sqsubseteq$, and (ii) Π holds iff $\omega_{A,B}(P) = \omega_{A,B}(Q)$, if $\odot$ is $\equiv$.*

Example 4. Consider again problem Π' from Example 3. Then, $\omega_{A,B}(P) = \omega_{A,B}(Q) = \{(\emptyset, \{sel(a)\}),\ (\{s(a)\}, \{sel(a)\}), (\{s(a), s(b)\}, \{sel(a)\}), (\{s(a), s(b)\}, \{sel(b)\}), \ldots\}$, witnessing that Π' holds. $\square$

We finally give a characterisation of wedges in terms of classical models of program reducts, extending a similar one for the propositional case [14].

Theorem 5. *Let $\mathscr{V}$ be a vocabulary and P a program over $\mathscr{V}$. Then, (X, Y) is an A-B-wedge of P iff (i) $X \in \mathscr{F}^A_{\mathscr{V}}$, (ii) $Y \in \mathscr{F}^B_{\mathscr{V}}$, and (iii) there exists an interpretation Y' over $\mathscr{V}$ such that $X \subseteq Y'$, $Y' \equiv^B_{\mathscr{V}} Y$, $Y' \models grd(P, HU_{P \cup X})$, and for each X' with $X \subseteq X' \subset Y'$, $X' \not\models grd(P, HU_{Q \cup X})^{Y'}$.*

4 Computability Issues

Next, we analyse the computational complexity of deciding correspondence problems, and we draw a border between decidable and undecidable instances of the framework.

It was shown by Shmueli [17] that query equivalence for Horn programs over infinite domains is undecidable. This undecidability result was extended by Eiter *et al.* [12] to program equivalence and uniform equivalence for disjunctive logic programs under the answer-set semantics. Since these notions can be formulated as special instances within our framework, we immediately get the following result:

Theorem 6. *The problem of determining whether a given correspondence problem over some vocabulary holds is undecidable in general, even if the programs under consideration are positive or normal.*

Some important decidable instances of the framework are obtained by imposing certain restrictions on the language of the programs under consideration. First of all, if we only consider propositional frames, checking inclusion and equivalence problems is decidable; in fact, this task is Π^P_4-complete in general [8]. Likewise, the complexity of checking PQIPs and PQEPs is Π^P_3-complete [14].

Also, for vocabularies with a finite domain, correspondence problems are decidable. In such a setting, programs can be seen as compact representations of their groundings over that domain. Since the size of a grounding of a program is, in general, exponential in the size of the program, for problems over a finite domain, we immediately obtain an upper bound for the complexity of correspondence checking which is increased by one exponential compared to the propositional case. That is, for a vocabulary $\mathscr{V}$ with a finite domain, checking inclusion and equivalence problems over $\mathscr{V}$ has co-NEXPTIME$^{\Sigma^P_3}$ complexity, and checking GQIPs and GQEPs over $\mathscr{V}$ has co-NEXPTIME$^{\Sigma^P_2}$ complexity.

More relevant in practice than frames over finite domains or propositional vocabularies are frames over infinite domains. For this setting, some decidable instantiations were already singled out in the literature. For example, ordinary equivalence checking is decidable, being co-NEXPTIME$^{\mathrm{NP}}$-complete [18]. It was shown that deciding strong equivalence of two programs over an infinite domain is co-NEXPTIME-complete [12]. If we factor in Theorem 3, we obtain the following result:

Theorem 7. *Deciding whether a problem of form $(P,Q,\mathscr{P}_{\mathscr{V}},\odot^{B}_{\mathscr{V}})$, for $\odot \in \{\sqsubseteq,\equiv\}$, over some $\mathscr{V}=(\mathbb{P},\mathbb{D})$ holds is co-NEXPTIME-complete, for any $B\subseteq\mathbb{P}$.*

One could conjecture that relativised strong equivalence is decidable as well, since both "ends" of the parameterisation, viz. ordinary equivalence and strong equivalence, are decidable. Interestingly, this is not the case.

Lemma 1. *Let P and Q be two programs over a vocabulary $\mathscr{V}$ and $\mathscr{E}$ the set of the extensional predicates of $P\cup Q$. Then, $P\simeq_{(\mathscr{V},\mathscr{F}^{\mathscr{E}}_{\mathscr{V}},=)} Q$ iff $P\simeq_{(\mathscr{V},\mathscr{P}^{\mathscr{E}}_{\mathscr{V}},=)} Q$.*

As $(\mathscr{V},\mathscr{F}^{\mathscr{E}}_{\mathscr{V}},=)$-correspondence coincides with program equivalence (cf. Theorem 1), and program equivalence is undecidable, we immediately obtain the following result:

Theorem 8. *The problem of determining whether a given correspondence problem of form $(P,Q,\mathscr{P}^{\mathbb{A}}_{\mathscr{V}},=)$ over some vocabulary $\mathscr{V}$ holds is undecidable.*

In other words, although testing relativised strong equivalence is decidable in the propositional case [7], it is undecidable in the general non-ground setting.

It is to mention that the important case of uniform equivalence, which is undecidable in general, is decidable for Horn programs [6]. Furthermore, Eiter *et al.* [12] give a detailed exposition of the complexity of strong and uniform equivalence with respect to restricted syntactic classes of programs. Although lower bounds could be obtained from these results, they do not shift the border between decidable and undecidable instantiations of our framework.

5 Conclusion

The focus of our work is on notions of program correspondence for non-ground logic programs under the answer-set semantics. Previously, refined equivalence notions, taking answer-set projection and relativised contexts into account, were defined and studied for propositional programs only. Indeed, the framework introduced in this paper is a lifting of a framework due to Eiter *et al.* [8] defined for specifying parameterised program correspondence notions for propositional programs. Our framework allows to capture several well-known equivalence notions in a uniform manner and allows to directly extend notions studied so far for propositional programs only to the non-ground case. We mention in passing that generality relations on logic programs, as studied by Inoue and Sakama [19], are also expressible by suitable instantiations of the comparison relation.

We introduced GQIPs and GQEPs, which generalise uniform equivalence as well as query and program equivalence, and provided model-theoretic characterisations for them quite in the spirit of UE-models for characterising uniform equivalence.

Concerning future work, we plan to provide characterisations, similar to those given for GQIPs and GQEPs in this paper, for inclusion and equivalence problems capturing relativised strong equivalence with projection (recall that GQEPs amount to relativised uniform equivalence with projection). A further point will be to study the generalised answer-set semantics recently introduced by Ferraris, Lee, and Lifschitz [20] in connection with our correspondence framework—in particular, to characterise equivalence notions for the generalised answer-set semantics in terms of second-order logic, as that semantics is itself defined by means of second-order logic.

References

1. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL **7**(3) (2006) 499–562
2. Janhunen, T., Niemelä, I., Seipel, D., Simons, P.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL **7**(1) (2006) 1–37
3. Lifschitz, V., Pearce, D., Valverde, A.: Strongly Equivalent Logic Programs. ACM TOCL **2**(4) (2001) 526–541
4. Eiter, T., Fink, M.: Uniform Equivalence of Logic Programs under the Stable Model Semantics. In: Proc. ICLP 2003. Vol. 2916 of LNCS, Springer (2003) 224–238
5. Maher, M.J.: Equivalences of Logic Programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1988) 627–658
6. Sagiv, Y.: Optimizing Datalog Programs. In Minker, J., ed.: Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann (1988) 659–698
7. Woltran, S.: Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In: Proc. JELIA 2004. Vol. 3229 of LNCS, Springer (2004) 161–173
8. Eiter, T., Tompits, H., Woltran, S.: On Solution Correspondences in Answer-Set Programming. In: Proc. IJCAI 2005. (2005) 97–102
9. Oikarinen, E., Janhunen, T.: Modular Equivalence for Normal Logic Programs. In: Proc. ECAI 2006. IOS (2006) 412–416
10. Inoue, K., Sakama, C.: Equivalence of Logic Programs under Updates. In: Proc. JELIA 2004. Vol. 3229 of LNCS, Springer (2004) 174–186
11. Lin, F.: Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In: Proc. KR 2002. Morgan Kaufmann (2002) 170–176
12. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. In: Proc. AAAI 2005. AAAI Press (2005) 695–700
13. Lifschitz, V., Pearce, D., Valverde, A.: A Characterization of Strong Equivalence for Logic Programs with Variables. In: Proc. LPNMR 2007. Vol. 4483 of LNCS, Springer (2007) 188–200
14. Oetsch, J., Tompits, H., Woltran, S.: Facts do not Cease to Exist Because They are Ignored: Relativised Uniform Equivalence with Answer-Set Projection. In: Proc. AAAI 2007. AAAI Press (2007) 458–464
15. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
16. Turner, H.: Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. TCLP **3**(4-5) (2003) 602–622
17. Shmueli, O.: Decidability and Expressiveness Aspects of Logic Queries. In: Proc. PODS 1987. ACM (1987) 237–249
18. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys **33**(3) (2001) 374–425
19. Inoue, K., Sakama, C.: Generality Relations in Answer Set Programming. In: Proc. ICLP 2006. Vol. 4079 of LNCS, Springer (2006) 211–225
20. Ferraris, P., Lee, J., Lifschitz, V.: A New Perspective on Stable Models. In: Proc. IJCAI 2007. (2007) 372–379