Program Correspondence under the Answer-Set Semantics: The Non-ground Case*

Johannes Oetsch and Hans Tompits

Institut für Informationssysteme,
Arbeitsbereich Wissensbasierte Systeme 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 that ordinary equivalence between answer-set programs fails 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. Furthermore, we consider axiomatisations of such correspondence problems by means of polynomial translations into second-order logic.

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 highly sophisticated ASP solvers exist, like DLV [1] or GNT [2], and typical application areas of ASP are configuration, information integration, security analysis, agent systems, Semantic Web, and planning.

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

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

M. Garcia de la Banda and E. Pontelli (Eds.): ICLP 2008, LNCS 5366, pp. 591–605, 2008.
© Springer-Verlag Berlin Heidelberg 2008

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 $P \subseteq R$, 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 subprograms but which are ignored in the final computation. Thus, these notions do not admit the *projection* of answer sets to a set of designated output letters. For example, consider the two programs $P = \{r(x) \leftarrow s(x)\}$ and $Q = \{aux(x) \leftarrow s(x);\ r(x) \leftarrow aux(x)\}$. While P expresses that $r(x)$ is selected whenever $s(x)$ is known, Q yields the selection of $r(x)$ via the auxiliary predicate $aux(x)$. P and Q are not uniformly equivalent (as P and Q conjoined with any fact $aux(\cdot)$ have differing answer sets), and hence they are not strongly equivalent, but they are if the context programs do not contain aux which also has to be ignored in the comparison of the 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 a range of different program equivalence notions, including the propositional ones mentioned above as well as non-ground versions of strong and uniform equivalence studied already in the literature [11,12,13].

We pay particular attention to correspondence problems which we refer to as *generalised query inclusion problems* (GQIPs) and *generalised query equivalence problems*

[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 as program segments", and likewise uniform equivalence was first studied for datalog programs by Sagiv [6].

(GQEPs), respectively. These are extensions of notions studied in previous work [14] for propositional programs (there termed PQIPs and PQEPs—*propositional query inclusion problems* and *propositional query equivalence problems*). 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, such versions of generalised uniform equivalence are appropriate in cases where programs are composed out of modules, arranged in layers, such that 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 for correspondence problems, showing in particular that relativised strong equivalence is undecidable, which is, in some sense, surprising because usual (unrelativised) strong equivalence is decidable both in the propositional and the non-ground case.

Finally, we study axiomatisations of GQIPs and GQEPs in terms of second-order logic (SOL), which is in the spirit of providing logical characterisations of program equivalence, following the seminal result by Lifschitz, Pearce, and Valverde [3] that strong equivalence between propositional programs coincides with equivalence in the logic of here-and-there, which is intermediate between classical logic and intuitionistic logic. Indeed, the use of SOL recently gained increasing interest in ASP as Ferraris, Lee, and Lifschitz [15] defined an answer-set semantics for general first-order theories in terms of SOL, avoiding the need of an explicit grounding step. As well, our SOL characterisations generalise results for axiomatising PQIPs and PQEPs by means of quantified propositional logic, which is just a restricted form of SOL.

2 Preliminaries

2.1 Logic Programs

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 *LP-domain*, or simply *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 $p(t_1,\ldots,t_n)$, 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 and the constants from C.

A *rule over* $\mathscr{V}$ is an expression of form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n, \tag{1}$$

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 $\{a_1, \ldots, a_l\}$ the *head of* r. Moreover, we call $\{a_{l+1}, \ldots, a_m\}$ the *positive body of* r and $\{a_{m+1}, \ldots, a_n\}$ the *negative body of* r. 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, a predicate symbol p that does not occur in the head of any rule in P is *extensional* (in P), and *intensional* otherwise. We denote by $PS(P)$ the set of all predicate symbols occurring in a program P, and say that P is *over* A if $PS(P) \subseteq A$. 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 $C \subseteq \mathbb{D}$ and any rule r, we define the *grounding of* r *with respect to* C, denoted by $grd(r,C)$, as the set of all rules obtained from r by uniformly replacing all variables occurring in r by constants from C. Furthermore, for a program P, the *grounding of* 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 [16], 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. When omitting A in $\mathscr{P}^A_{\mathscr{V}}$ or $\mathscr{F}^A_{\mathscr{V}}$, we assume that $A = \mathbb{P}$. We also 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 $\mathscr{V} = (\mathbb{P},\mathbb{D})$ be a vocabulary and P, Q two 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 $F \in \mathscr{F}^{\mathscr{E}}_{\mathscr{V}}$, $AS(P \cup F) = AS(Q \cup F)$.

2.2 Second-Order Logic

Classical second-order logic (SOL) extends classical first-order logic (FOL) by admitting quantifications over predicate and function variables. For our purposes, it suffices to consider a version of SOL without function variables. More formally, we assume that the vocabulary of SOL contains denumerable sets of *individual variables*, *predicate variables*, *predicate constants*, and *individual constants*, as well as the truth constants $\top$ and $\bot$, the usual connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$, and the quantifiers $\forall$ and $\exists$. Each predicate variable and predicate constant is assumed to have a non-negative arity.

We use $\mathrm{Q}X_1 \cdots X_n \Phi$, for $\mathrm{Q} \in \{\forall,\exists\}$, as an abbreviation of $\mathrm{Q}X_1 \cdots \mathrm{Q}X_n \Phi$. Moreover, for any set $X = \{X_1,\ldots,X_n\}$ of variables, $\mathrm{Q}X\Phi$, $\mathrm{Q} \in \{\forall,\exists\}$, abbreviates $\mathrm{Q}X_1 \cdots X_n \Phi$.

The semantics of SOL is defined in terms of *SOL-frames*. A SOL-frame is a tuple $\langle D,m\rangle$, where D is a non-empty set, called the *SOL-domain*, and m is an *interpretation function* that maps any predicate constant with arity n to a subset of D^n and any individual constant to an element of D. An *assignment over* M is a function a that maps any predicate variable with arity n to a subset of D^n and any individual variable to an element of D. If the domain of a SOL-frame M is countable and M meets the unique-names assumption, i.e., different constant symbols are interpreted by different objects from the domain, then M is a *Herbrand frame*. Constant symbols are interpreted by themselves in a Herbrand frame. The *truth-value* of a formula Φ in a SOL-frame M with respect to an assignment a over M is denoted by $V^M_a(\Phi)$ and is defined as usual.

A formula Φ is *satisfied* in a SOL-frame M by an assignment a over M iff $V^M_a(\Phi) = 1$. Moreover, Φ is *true* in M iff Φ is satisfied in M by every assignment over M. If Φ is true in M, M is a *model* of Φ. A formula is *satisfiable* iff it possesses at least one model and *unsatisfiable* otherwise. A formula that is true in all frames is called *valid*. A formula Φ is *consequence* of a set A of formulae, symbolically $A \models \Phi$ iff, for each SOL-frame M, Φ is true in M whenever each formula in A is true in M.

We tacitly assume in what follows that each variable in a program vocabulary $\mathscr{V}$ is an individual variable in our SOL-vocabulary, each predicate symbol in $\mathscr{V}$ is a predicate variable, and each constant in $\mathscr{V}$ is an individual constant.

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 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, viz. on correspondence notions generalising uniform equivalence. We then provide model-theoretic characterisations for this particular family of correspondences and analyse some computational properties.

3.1 The Basic Framework and Its Instances

We start with our central definition, which is a lifting of a correspondence framework introduced by Eiter *et al.* [8] for propositional programs to the non-ground setting.

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, it is assumed that $B = \mathbb{P}$. 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 M does not contain $aux(\cdot)$ which is also ignored concerning the answer sets of M. 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. □

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 a *propositional frame*) 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 so $F_0=(\mathscr{V}_0,\mathscr{C},\rho)$ 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. 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 $\mathscr{U}$, 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 $\mathscr{U}$, where P and Q are propositional programs over $\mathscr{U}$, $A,B\subseteq\mathscr{U}$, and $\mathscr{F}^A$ is as above. 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, 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. □

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 $(\mathscr{V}, \mathscr{P}_{\mathscr{V}}, =)$. Then, two programs are F-corresponding iff they are 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.

3.2 Model-Based Characterisations of GQIPs and GQEPs

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* [17] and 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. *A* counterexample *for a GQIP $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$ over some vocabulary $\mathscr{V}$ is a pair (X, Y) such that (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, a counterexample for a GQEP Π is a pair which is a counterexample for one of $\Pi_{\sqsubseteq}$ or $\Pi_{\sqsupseteq}$.*

Obviously, given a problem $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \odot^B_{\mathscr{V}})$, where $\odot \in \{\sqsubseteq, \equiv\}$, Π does not hold iff there exists a counterexample for Π.

Example 3. Recall problem Π from Example 2, which does not hold. The pair $(\emptyset, \emptyset)$ is a counterexample for $\Pi_{\sqsupseteq}$ 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. □

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

Definition 4. *Let $\mathcal{V} = (\mathbb{P}, \mathbb{D})$ be a vocabulary, $A, B \subseteq \mathbb{P}$, and P a program over $\mathcal{V}$. An A-B-*wedge of P over $\mathcal{V}$ *is a pair (X, Y) such that $X \in \mathscr{F}^A_{\mathcal{V}}$ 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 for a GQIP $\Pi = (P, Q, \mathscr{F}^A_{\mathcal{V}}, \sqsubseteq^B_{\mathcal{V}})$ iff an A-B-wedge of P exists that is not an A-B-wedge of Q. Hence:

Theorem 4. *For $\Pi = (P, Q, \mathscr{F}^A_{\mathcal{V}}, \odot^B_{\mathcal{V}})$, with $\odot \in \{\sqsubseteq, \equiv\}$, we have that (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. □

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

3.3 Computability Issues

We continue with an analysis of the computational complexity of deciding correspondence problems. In particular, our primary aim is to draw a border between decidable and undecidable instances of the framework.

It was shown by Shmueli [18] that query equivalence for Horn programs over infinite domains is undecidable. This undecidability result was subsequently 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]. Moreover, 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 are 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

complexity bound for 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$^{\Sigma_3^P}$ complexity, and checking GQIPs and GQEPs over $\mathscr{V}$ has co-NEXPTIME$^{\Sigma_2^P}$ 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, checking ordinary equivalence is decidable, being co-NEXPTIME$^{\mathrm{NP}}$-complete [19]. Furthermore, 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* $(P, Q, \mathscr{P}_{\mathscr{V}}, \odot_{\mathscr{V}}^{B})$, *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 parametrisation, 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,* $P \simeq_{(\mathscr{V}, \mathscr{F}_{\mathscr{V}}^{\mathscr{E}}, =)} Q$ *iff* $P \simeq_{(\mathscr{V}, \mathscr{P}_{\mathscr{V}}^{\mathscr{E}}, =)} Q$.

As $(\mathscr{V}, \mathscr{F}_{\mathscr{V}}^{\mathscr{E}}, =)$-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* $(P, Q, \mathscr{P}_{\mathscr{V}}^{A}, =)$ *over some vocabulary* $\mathscr{V}$ *holds is undecidable.*

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

Finally, 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.* [20] give a detailed exposition of the complexity of strong and uniform equivalence with respect to restricted syntactic classes of programs. Although lower bounds for analogous classes could be obtained from these results, they do not shift the border between decidable and undecidable instantiations of our framework.

4 Translations into Second-Order Logic

In this section, we introduce encodings of GQIPs and GQEPs in terms of second-order logic (SOL). We start with some aspects concerning the choice of SOL as target formalism, rather than of FOL, and afterwards we present the encodings.

4.1 A Case for Second-Order Logic

SOL is a highly expressive formalism—too expressive for effective computing—yet with important applications in knowledge representation (KR) and nonmonotonic reasoning. Besides McCarthy's well-known circumscription formalism [21], SOL recently gained increased interest in the ASP area by using it to define a generalised answer-set semantics

for first-order theories (which is actually closely related to circumscription) [15]. Also, many encodings of different KR formalisms into quantified propositional logic were developed, which is just a decidable fragment of SOL. Indeed, in previous work [14], polynomial translations of PQIPs and PQEPs into quantified propositional logic were developed, and therefore translating GQIPs and GQEPs into SOL is a natural lifting. Nevertheless, one may seek translations into FOL for them. In what follows, we show some undesired feature of such an endeavour.

Let us understand by an *FOL-translation* of a class S of correspondence problems a function assigning to each element of S a FOL-formula. An FOL-translation T of S is *faithful* iff, for any $\Pi \in S$, $T(\Pi)$ is valid iff Π holds.

Now consider the class U of problems of form $(P,Q,\mathscr{F}_{\mathscr{V}},=)$, i.e., problems testing for uniform equivalence, where the domain of $\mathscr{V}$ is denumerable, and assume that T_U is an FOL-translation of S_U which is both faithful and computable. Define $C \subseteq S_U$ as the set containing all the problems in S_U that hold and let $\bar{C} = S_U \setminus C$. We first show that C is recursively enumerable. Consider the following algorithm: Let $\Pi \in S_U$. (i) Compute $\varphi = T_U(\Pi)$. (ii) Take any sound and complete calculus for FOL, and start to enumerate all proofs. If a proof for φ is found, terminate. If φ is valid, this method will terminate after a finite number of steps (by completeness of the calculus), otherwise the method will loop forever (by soundness of the calculus). Hence, C is indeed recursively enumerable.

Next, we show that $\bar{C}$ is recursively enumerable as well. Consider the following algorithm: Take a problem $\Pi = (P,Q,\mathscr{F}_{\mathscr{V}},=)$ from S_U. Start enumerating $\mathscr{F}_{\mathscr{V}}$. For each such $X \in \mathscr{F}_{\mathscr{V}}$, if $AS(P \cup X) \neq AS(Q \cup X)$, terminate. If Π does not hold, there exists a counterexample for Π and the algorithm will terminate after a finite number of steps, otherwise it will loop forever. Thus, $\bar{C}$ is also recursively enumerable. Since both C and $\bar{C}$ are recursively enumerable, S_U must be decidable. But this is a contradiction since S_U is not decidable. We showed the following result:

Theorem 9. *Let S_U be the class of problems of form $(P,Q,\mathscr{F}_{\mathscr{V}},=)$, where the domain of $\mathscr{V}$ is denumerable. Then, any FOL-translation of S_U is either not faithful or not computable.*

Theorem 9 applies for any logic possessing sound and complete calculi and extends to any class of correspondence problems that allows to decide uniform equivalence. Hence, characterising correspondence problems is indeed a case for full SOL; not only undecidability but also incompleteness with respect to deduction is inherent for verifying program correspondences in general.

4.2 Translating GQIPs and GQEPs

In the sequel, we will make use of superscripts as a renaming schema for predicate variables and formulae. Formally, for any set V of predicate variables, we assume pairwise disjoint sets $V^i = \{v^i \mid v \in V\}$ of predicate variables, for every $i \geq 1$. For a formula ϕ and every $i \geq 1$, ϕ^i is the formula obtained from ϕ by uniformly replacing each occurrence of an atomic formula $a(t_1,\ldots,t_n)$ in ϕ by $a^i(t_1,\ldots,t_n)$. Consider a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$. For any rule r over $\mathscr{V}$ of form (1), we define $H(r) = a_1 \vee \cdots \vee a_l$, $B^+(r) = a_{l+1} \wedge \cdots \wedge a_m$, and $B^-(r) = \neg a_{m+1} \wedge \cdots \wedge \neg a_n$. We identify empty disjunctions with $\bot$ and empty conjunctions with $\top$.

Next, we introduce some building blocks for our encodings. These are basically lifted versions of ones used to characterise different notions of program correspondence for propositional programs by means of quantified propositional logic [22,14].

We will relate sets of ground atoms with assignments over Herbrand frames in the following way: Let X be a set of predicate variables and a an assignment over a Herbrand frame M. Moreover, consider a program vocabulary $\mathscr{V}$. Then, for any integer i, $a|X^i| = \{x(t_1,\ldots,t_n) \mid x^i \in X^i, (t_1,\ldots,t_n) \in a(x^i)\}$, and, syntactically, $a|X^i|$ is assumed to be a set of ground atoms over $\mathscr{V}$.

Definition 5. *Let P be a program over a vocabulary $\mathscr{V}$ and $i,j \geq 1$. Then, $P^{\langle i,j\rangle} =_{df} \bigwedge_{r\in P} r^{\langle i,j\rangle}$, where $r^{\langle i,j\rangle} =_{df} \forall X\big((B^+(r^i) \wedge B^-(r^j)) \rightarrow H(r^i)\big)$ and X is the set of all variables occurring in r.*

Theorem 10. *Let P be a program over a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$, $V = PS(P)$, M a Herbrand frame along with an assignment a over M, and $X,Y \subseteq HB_{\mathscr{V}}$ such that for some i,j, $X = a|V^i|$ and $Y = a|V^j|$. Then, $X \models grd(P,\mathbb{D})^Y$ iff $P^{\langle i,j\rangle}$ is satisfied in M by a.*

Example 5. Let $P = \{p(x) \leftarrow q(x), \mathrm{not}\, q(a);\ q(x) \leftarrow p(x), \mathrm{not}\, q(b)\}$ be a program over a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$, $\mathbb{P} = \{p,q\}$, $\mathbb{D} = \{a,b\}$, and let $X = \{p(a),q(a),q(b)\}$ and $Y = \{q(a)\}$ be interpretations over $\mathscr{V}$. Observe that $X \models grd(P,\mathbb{D})^Y$. Furthermore, consider a Herbrand frame M and an assignment a over M such that $a|\{p,q\}^1| = X$ and $a|\{p,q\}^2| = Y$. It can be verified that $P^{\langle 1,2\rangle} = \forall x(q^1(x) \wedge \neg q^2(a) \rightarrow p^1(x)) \wedge \forall x(p^1(x) \wedge \neg q^2(b) \rightarrow q^1(x))$ is satisfied in M by a.

We also make use of the following abbreviations for sets of predicate variables: Let p be a predicate variable of arity n. For any two integers $i,j \geq 1$, define $p^i \leq p^j =_{df} \forall x_1 \cdots x_n(p^i(x_1,\ldots x_n) \rightarrow p^j(x_1,\ldots x_n))$. Furthermore, let P be a finite set of predicate variables and let $i,j \geq 1$ be integers. Then, $(P^i \leq P^j) =_{df} \bigwedge_{p\in P}(p^i \leq p^j)$, $(P^i < P^j) =_{df} (P^i \leq P^j) \wedge \neg(P^j \leq P^i)$, and $(P^i = P^j) =_{df} (P^i \leq P^j) \wedge (P^j \leq P^i)$.

We proceed with our central encoding, which captures the notion of an A-B-wedge.

Definition 6. *Let P be a program over a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$. Furthermore, let A,B be finite subsets of $\mathbb{P}$ and $V = PS(P) \cup A \cup B$. Then, $TR_{\mathscr{V}}^{(A,B)}(P)$ is given by*

$$\exists V^3\Big((B^3 = B^1) \wedge (A^2 \leq A^3) \wedge P^{\langle 3,3\rangle} \wedge \forall V^4\big(((A^2 \leq A^4) \wedge (V^4 < V^3)) \rightarrow \neg P^{\langle 4,3\rangle}\big)\Big).$$

Lemma 2. *Let P be a program over a vocabulary $\mathscr{V} = (\mathbb{P},\mathbb{D})$, A,B finite subsets of $\mathbb{P}$, and $V = PS(P) \cup A \cup B$. Then, for any finite Herbrand frame M and an assignment a over M such that $X = a|A^2|$ and $Y = a|B^1|$, it holds that (X,Y) is an A-B-wedge of P iff $TR_{\mathscr{V}}^{(A,B)}(P)$ is satisfied in M by a.*

To give an intuition of the above encoding, recall the characterisation of wedges in terms of interpretations from Theorem 5. If we take the semantics of our building blocks into account, Definition 6 basically encodes Conditions (i)–(iii) of Theorem 5 by means of SOL. In particular, the interpretations Y, X, Y', and X' from Theorem 5 are captured by the sets B^1, A^2, V^3, and V^4 of variables in $TR_{\mathscr{V}}^{(A,B)}(P)$.

Next, we introduce two axioms that make some assumptions explicit. The first one captures the well-known *unique-names assumption*.

Definition 7. *For any program P over vocabulary $\mathscr{V}$, the* unique-names axiom *for P is given by* $UNA(P) =_{df} \bigwedge_{a,b \in HU_P, a \neq b} \neg(a = b)$.

We consider only finite sets in the context class of frames, hence we need as second axiom one which imposes finite SOL-domains for the models which will correspond to the domains of the programs and the sets of facts from the context class.

Definition 8. *Let $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ be a vocabulary. Then, $FDA(\mathbb{D})$ is given as follows:*

1. *If $\mathbb{D}$ is finite, then* $FDA(\mathbb{D}) =_{df} \forall x(\bigvee_{c \in \mathbb{D}}(x = c))$.
2. *If $\mathbb{D}$ is denumerable, then* $FDA(\mathbb{D}) =_{df} \exists S\big(\forall x S(x) \wedge \forall R\big(PO(R,S) \rightarrow MAX(R,S)\big)\big)$, *where*

$$PO(R,S) =_{df} \forall xyz\Big(S(x) \wedge S(y) \wedge S(z) \rightarrow (R(x,x) \wedge (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge (R(x,y) \wedge R(y,x) \rightarrow x = y))\Big) \text{ and}$$

$$MAX(R,S) =_{df} \exists m\big(S(m) \wedge \forall x(S(x) \wedge R(m,x) \rightarrow m = x)\big).$$

The intuition of the first item should be clear: The SOL-domain of any model does not contain more objects than enumerated by the finite disjunction over the elements from the LP-domain $\mathbb{D}$. Hence, $FDA(\cdot)$ turns into the customary *domain-closure axiom*. If $\mathbb{D}$ is denumerable, the intuition of the second item is the following: $PO(R,S)$ is true if R is a partial order on S, and $MAX(R,S)$ is true if S contains a maximal element with respect to R. Hence, $FDA(\mathbb{D})$ encodes that there exists a set S such that each element of the SOL-domain in any model is in S and, for each partial order on S, S contains a maximal element with respect to that order. The latter is a well-known set-theoretic condition that ensures that S is finite. Since, for each model, any element from the SOL-domain is in S, it follows that each model is finite as well.

We are now in a position to state the main characterisation in SOL.

Theorem 11. *Let Π be a correspondence problem of form $(P, Q, \mathscr{F}^A_{\mathscr{V}}, \odot^B_{\mathscr{V}})$, where $\mathscr{V} = (\mathbb{P}, \mathbb{D})$ and $\odot \in \{\sqsubseteq, \equiv\}$, and let $V = PS(P \cup Q)$, $A' = A|_V$, and $B' = B|_V$. Then:*

(*i*) *Π holds iff* $UNA(P \cup Q), FDA(\mathbb{D}) \models TR^{(A',B')}_{\mathscr{V}}(P) \rightarrow TR^{(A',B')}_{\mathscr{V}}(Q)$, *if $\odot$ is $\sqsubseteq$.*

(*ii*) *Π holds iff* $UNA(P \cup Q), FDA(\mathbb{D}) \models TR^{(A',B')}_{\mathscr{V}}(P) \leftrightarrow TR^{(A',B')}_{\mathscr{V}}(Q)$, *if $\odot$ is $\equiv$.*

That $FDA(\cdot)$ is indeed required can be seen as follows: For a vocabulary with a denumerable domain, define the programs

$$P = \left\{ \begin{array}{l} s(a);\ r(x,y) \vee r(y,x) \leftarrow s(x), s(y); \\ r(x,z) \leftarrow r(x,y), r(y,z);\ p(x,y) \leftarrow r(x,y), \text{not } r(y,x); \\ \bar{g}(x) \leftarrow p(x,y);\ g \leftarrow s(x), \text{not } \bar{g}(x) \end{array} \right\},\ Q = P \cup \{ \leftarrow \text{not } g \}.$$

The intuition behind program P is that, for any set X of facts over s, each answer set from $P \cup X$ contains g (the greatest element) iff X is finite. This is realised by formalising the property that any strict total order on a non-empty set S induces a greatest element with respect to this order relation iff S is finite. On the other hand, program $Q \cup X$ has no answer sets iff X is infinite. This is realised by defining Q as P augmented with the rule $\leftarrow$ not g that eliminates any answer set not containing g. In a nutshell, programs P and Q show that the answer-set semantics is expressible enough to distinguish whether the

domain of X is finite. Now consider the problem $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$, where $A = \{s\}$, $B = \{g\}$, and $\mathscr{V}$ is a vocabulary with a denumerable domain. It is easy to see that Π holds since any answer set of P and Q contains g when joined with finite sets of facts over $\{s\}$. Nevertheless, $\mathit{UNA}(P \cup Q) \models \mathit{TR}^{(A,B)}_{\mathscr{V}}(P) \rightarrow \mathit{TR}^{(A,B)}_{\mathscr{V}}(Q)$ does not hold since there exist models of infinite size that correspond, roughly speaking, to infinite counterexamples.

Finally, one may wonder why $\mathit{FDA}(\cdot)$ comes in two guises, depending on the cardinality of the LP-domain. Let us assume that $\mathit{FDA}(\cdot)$ would be defined only by Item 2 of Definition 8, irrespective of the cardinality of $\mathbb{D}$. We use FDA' to refer to this version of $\mathit{FDA}(\cdot)$. Consider the following two programs: $P = \{eq(x,x) \leftarrow s(x)\}$ and $Q = P \cup \{\leftarrow s(x), s(y), \mathrm{not}\ eq(x,y)\}$. Program P expresses that any element in the set implicitly defined by s is equal to itself; program Q is defined as P plus a rule expressing that Q has no answer set iff the set implicitly defined by s has more than one element. Hence, the problem $\Pi = (P, Q, \mathscr{F}^A_{\mathscr{V}}, \sqsubseteq^B_{\mathscr{V}})$, for $A = \{s\}$ and $B = \emptyset$, holds only if the vocabulary $\mathscr{V}$ has a domain with less than two elements. Assume that $\mathscr{V}$ is such a vocabulary. However, $\mathit{UNA}(P \cup Q), \mathit{FDA}' \models \mathit{TR}^{(A,B)}_{\mathscr{V}}(P) \rightarrow \mathit{TR}^{(A,B)}_{\mathscr{V}}(Q)$ does not hold since FDA' only ensures that all models are finite but it makes no commitment about their cardinalities.

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, moreover, allows to directly extend notions studied so far for propositional programs only to the general non-ground case. In particular, we introduced GQIPs and GQEPs as generalisations of PQIPs and PQEPs, respectively, which in turn generalise uniform equivalence as well as query and program equivalence. We provided model-theoretic characterisations for GQIPs and GQEPs—in the spirit of characterising uniform equivalence in terms of UE-models—and axiomatised them in terms of second-order logic.

We plan to provide characterisations similar to those given for GQIPs and GQEPs to correspondence 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 [15] in connection with our correspondence framework—in particular, to characterise equivalence notions for this 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), 499–562 (2006)
2. Janhunen, T., Niemelä, I., Seipel, D., Simons, P.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL 7(1), 1–37 (2006)

3. Lifschitz, V., Pearce, D., Valverde, A.: Strongly Equivalent Logic Programs. ACM TOCL 2(4), 526–541 (2001)
4. Eiter, T., Fink, M.: Uniform Equivalence of Logic Programs under the Stable Model Semantics. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 224–238. Springer, Heidelberg (2003)
5. Maher, M.J.: Equivalences of logic programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 627–658. Morgan Kaufmann, San Francisco (1988)
6. Sagiv, Y.: Optimizing Datalog Programs. In: Minker, J. (ed.) Foundations of Deductive Databases and Logic Programming, pp. 659–698. Morgan Kaufmann, San Francisco (1988)
7. Woltran, S.: Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 161–173. Springer, Heidelberg (2004)
8. Eiter, T., Tompits, H., Woltran, S.: On Solution Correspondences in Answer-Set Programming. In: 19th International Joint Conference on Artificial Intelligence, pp. 97–102 (2005)
9. Oikarinen, E., Janhunen, T.: Modular Equivalence for Normal Logic Programs. In: 17th European Conference on Artificial Intelligence, pp. 412–416. IOS Press, Amsterdam (2006)
10. Inoue, K., Sakama, C.: Equivalence of Logic Programs Under Updates. In: Alferes, J.J., Leite, J. (eds.) JELIA 2004. LNCS, vol. 3229, pp. 174–186. Springer, Heidelberg (2004)
11. Lin, F.: Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In: 8th International Conference on Principles of Knowledge Representation and Reasoning, pp. 170–176. Morgan Kaufmann, San Francisco (2002)
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: 20th National Conference on Artificial Intelligence, pp. 695–700. AAAI Press, Menlo Park (2005)
13. Lifschitz, V., Pearce, D., Valverde, A.: A Characterization of Strong Equivalence for Logic Programs with Variables. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS, vol. 4483, pp. 188–200. Springer, Heidelberg (2007)
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: 22nd National Conference on Artificial Intelligence, pp. 458–464. AAAI Press, Menlo Park (2007)
15. Ferraris, P., Lee, J., Lifschitz, V.: A New Perspective on Stable Models. In: 20th International Joint Conference on Artificial Intelligence, pp. 372–379 (2007)
16. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)
17. Turner, H.: Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. Theory and Practice of Logic Programming 3(4-5), 602–622 (2003)
18. Shmueli, O.: Decidability and Expressiveness Aspects of Logic Queries. In: 6th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 237–249. ACM, New York (1987)
19. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys 33(3), 374–425 (2001)
20. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Complexity Results for Checking Equivalence of Stratified Logic Programs. In: 20th International Joint Conference on Artificial Intelligence, pp. 330–335 (2007)
21. McCarthy, J.: Circumscription—A Form of Non-Monotonic Reasoning. Artificial Intelligence 13, 27–39 (1980)
22. Tompits, H., Woltran, S.: Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In: Gabbrielli, M., Gupta, G. (eds.) ICLP 2005. LNCS, vol. 3668, pp. 189–203. Springer, Heidelberg (2005)