Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming*

Hans Tompits and Stefan Woltran

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

Abstract. In recent work, a general framework for specifying program correspondences under the answer-set semantics has been defined. The framework allows to define different notions of equivalence, including the well-known notions of *strong* and *uniform equivalence*, as well as refined equivalence notions based on the *projection* of answer sets, where not all parts of an answer set are of relevance (like, e.g., removal of auxiliary letters). In the general case, deciding the correspondence of two programs lies on the fourth level of the polynomial hierarchy and therefore this task can (presumably) not be efficiently reduced to answer-set programming. In this paper, we describe an approach to compute program correspondences in this general framework by means of linear-time constructible reductions to *quantified propositional logic*. We can thus use extant solvers for the latter language as back-end inference engines for computing program correspondence problems. We also describe how our translations provide a method to construct *counterexamples* in case a program correspondence does not hold.

1 Introduction

Answer-set programming (ASP) is widely recognised as a fruitful paradigm for declarative knowledge representation and reasoning. It is based on the idea that problems are encoded in terms of theories of some suitable language, associated with a declarative semantics, such that the solutions of the given problems are determined by the models of the corresponding theories. Among the different instances of the ASP paradigm, the class of nonmonotonic logic programs under the answer-set semantics [14], with which we are concerned with in this paper, represents the canonical and, due to the availability of efficient answer-set solvers, like DLV [18], Smodels [26], and ASSAT [22], arguably most widely used ASP approach.

An important issue for the further deployment of ASP is to provide methods and tools for *engineering ASP solutions*. This includes techniques for the *simplification* and (offline) *optimisation* of programs, tools for supporting the user with *debugging* or *verification* features, and methods for *modular programming*. Crucial for all these issues are mechanisms for determining the *equivalence of* (*parts of*) *logic programs*.

In previous work [13], a general framework for specifying correspondences between logic programs under the answer-set semantics has been introduced. In this framework,

* This work was partially supported by the Austrian Science Fund (FWF) under grant P18019, and by the European Commission via projects FET-2001-37004 WASP, IST-2001-33570 INFOMIX, and IST-2001-33123 CologNeT.

M. Gabbrielli and G. Gupta (Eds.): ICLP 2005, LNCS 3668, pp. 189–203, 2005.
© Springer-Verlag Berlin Heidelberg 2005

the correspondence of two programs is determined in terms of a class $\mathcal{C}$ of *context programs* and a comparison relation ρ such that correspondence between two programs P and Q holds iff the answer sets of $P \cup R$ and $Q \cup R$ satisfy ρ, for any program $R \in \mathcal{C}$. The framework includes as special cases the well-known notions of *strong equivalence* [20], *uniform equivalence* [10], and relativised notions thereof [28], as well as the practicably important case of program comparison under *projected* answer sets. In the latter setting, not a whole answer set of a program is of interest, but only its intersection on a subset of all letters; this includes, in particular, removal of auxiliary letters in computation.

For the case of propositional disjunctive logic programs, correspondence checking in the above framework under projected answer sets is surprisingly hard, viz. Π_4^P-complete in general [13], i.e., lying on the fourth level of the polynomial hierarchy. Hence, this task can (presumably) not be efficiently reduced to propositional answer-set programming. Such an approach (used, e.g., by Oikarinen and Janhunen [23] for ordinary equivalence) reduces equivalence checking to problems like program consistency such that equivalence holds iff the resultant program possesses no answer set. Taking the results of Eiter *et al.* [9] into account, a compact reduction as such cannot even be obtained by using non-ground programs as long as we restrict the arities of predicates to a fixed constant. This indicates that advanced equivalence tests in answer-set programming cannot be straightforwardly solved using ASP systems themselves.

In this paper, we describe an approach to compute program correspondences in the framework of Eiter *et al.* [13] by means of efficient reductions to *quantified propositional logic*. The latter is an extension of classical propositional logic characterised by the condition that its sentences, usually referred to as *quantified Boolean formulas* (QBFs), are permitted to contain quantifications over atomic formulas. More specifically, our reductions enjoy the following properties:

1. a solution correspondence under projected answer sets between two given logic programs holds iff the associated QBF is valid in quantified propositional logic,
2. the reduction is constructible in *linear time and space*, and
3. determining the validity of the resultant QBFs under the translations is not computationally harder than checking the original correspondence problem.

Besides the reduction of correspondence problems, we also describe how our translations provide a method to construct *counterexamples* in case a program correspondence does not hold.

The rationale to consider a reduction approach to QBFs is twofold: On the one hand, complexity results about quantified propositional logic imply that decision problems from the polynomial hierarchy can be efficiently represented in terms of QBFs, and, on the other hand, several practicably efficient solvers for quantified propositional logic are currently available (like, e.g., the solvers QuBE [15] and semprop [19]; for others, cf. [17,16]). Hence, tools of the latter kind can be used as back-end inference engines to compute the correspondence problems under consideration. We note that a similar reduction approach to QBFs has been successfully applied in diverse fields like nonmonotonic reasoning [6,5,12], paraconsistent reasoning [3,1,2], planning [25], and automated deduction [7].

2 Preliminaries

We deal with propositional disjunctive logic programs, which are finite sets of rules of form

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

with $n \geq m \geq l \geq 0$, and where all a_i are propositional atoms from a universe $\mathcal{U}$ and not denotes default negation. We call a rule of the above form a *fact* if $l = 1$ and $m = n = 0$, and a *constraint* if $l = 0$. If all atoms occurring in a program P are from a given set $A \subseteq \mathcal{U}$ of atoms, we say that P is a program *over* A. The set of all programs over A is denoted by $\mathcal{P}_A$.

Let I be an interpretation, i.e., a set of atoms, and P a program. Following Gelfond and Lifschitz [14], I is an *answer set* of P iff it is a minimal model of the *reduct* P^I, resulting from P by (i) deleting all rules containing default negated atoms $\mathit{not}\, a$ such that $a \in I$ and (ii) deleting all default negated atoms in the remaining rules. The set of all answer sets of P is denoted by $\mathcal{AS}(P)$. We also write $I \models P$ to indicate that each rule in P is true under I (in the sense of classical logic).

Under the answer-set semantics, two programs P and Q are regarded as (ordinarily) equivalent iff $\mathcal{AS}(P) = \mathcal{AS}(Q)$. The more restrictive form of *strong equivalence* [20] has recently been generalised as follows [28]: Let P, Q be programs over $\mathcal{U}$, and let $A \subseteq \mathcal{U}$. Then, P and Q are *strongly equivalent relative to* A iff, for any $R \in \mathcal{P}_A$ $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$. If $A = \mathcal{U}$, strong equivalence relative to A reduces to strong equivalence; if $A = \emptyset$, we obtain ordinary equivalence.

We use the following notation: For an interpretation I and a set $\mathcal{S}$ of interpretations (resp., pairs of interpretations), we write $\mathcal{S}|_I = \{Y \cap I \mid Y \in \mathcal{S}\}$ (resp., $\mathcal{S}|_I = \{(X \cap I, Y \cap I) \mid (X, Y) \in \mathcal{S}\}$). If $\mathcal{S} = \{s\}$, we usually write $s|_I$ instead of $\mathcal{S}|_I$.

For any $A \subseteq \mathcal{U}$, a pair (X, Y) of interpretations, where $Y \subseteq \mathcal{U}$, is an *A-SE-interpretation* (over $\mathcal{U}$) iff either $X = Y$ or $X \subset Y|_A$. (X, Y) is an *A-SE-model* of a program P iff (i) $Y \models P$, (ii) for all $Y' \subset Y$ with $Y'|_A = Y|_A$, $Y' \not\models P^Y$, and (iii) $X \subset Y$ implies the existence of an $X' \subseteq Y$ with $X'|_A = X$ such that $X' \models P^Y$ holds. A pair (X, Y) is *total* iff $X = Y$, and *non-total* otherwise. The set of all A-SE-models of P is denoted by $SE^A(P)$.

For $A = \mathcal{U}$, the notion of an A-SE-interpretation (resp., A-SE-model) coincides with the notion of an *SE-interpretation* (resp., *SE-model*) as defined by Turner [27], and we write $SE(P)$ instead of $SE^{\mathcal{U}}(P)$. Thus, $(X, Y) \in SE(P)$ iff $X \subseteq Y$, $Y \models P$, and $X \models P^Y$.

Proposition 1 ([28]). *Two programs P and Q are strongly equivalent relative to A iff $SE^A(P) = SE^A(Q)$.*

Example 1. Consider the following two programs, P and Q:

$$P = P_0 \cup \{c \vee d \leftarrow a;\ c \vee d \leftarrow b\},$$
$$Q = P_0 \cup \{c \vee d \leftarrow a, b;\ d \leftarrow b, \mathit{not}\, c;\ c \leftarrow a, \mathit{not}\, d\},$$

for $P_0 = \{a \leftarrow c;\ b \leftarrow c;\ a \leftarrow d;\ b \leftarrow d;\ \leftarrow \mathit{not}\, c, \mathit{not}\, d\}$.

They have the following SE-models:[1]

$$SE(P) = \{(\emptyset, abc), (\emptyset, abd), (\emptyset, abcd), (abcd, abcd), (abc, abcd), (abd, abcd), (abc, abc), (abd, abd)\},$$
$$SE(Q) = SE(P) \cup \{(b, abc), (a, abd), (b, abcd), (a, abcd)\}.$$

Hence, P and Q are not strongly equivalent. On the other hand, $\mathcal{AS}(P) = \mathcal{AS}(Q) = \emptyset$, i.e., P and Q are (ordinarily) equivalent. Moreover, P and Q are strongly equivalent relative to A precisely if $A \cap \{a, b\} = \emptyset$. For $A = \{a, b\}$, we get

$$SE^A(P) = \{(\emptyset, abc),\ (\emptyset, abd),\ (abc, abc),\ (abd, abd)\},$$
$$SE^A(Q) = SE^A(P) \cup \{(b, abc),\ (a, abd)\}.$$

Thus, P and Q are not strongly equivalent relative to $A = \{a, b\}$. For instance, adding a fact $a \leftarrow$ yields $\mathcal{AS}(P \cup \{a \leftarrow\}) = \{abc, abd\}$, while $\mathcal{AS}(Q \cup \{a \leftarrow\}) = \{abc\}$. □

A set $\mathcal{S}$ of SE-interpretations is *complete* iff, for each $(X, Y) \in \mathcal{S}$, also $(Y, Y) \in \mathcal{S}$ as well as $(X, Z) \in \mathcal{S}$, for any Z such that $Y \subseteq Z$ and $(Z, Z) \in \mathcal{S}$. It can be shown that, for any program P, the set $SE(P)$ of all SE-models of P is always complete. Conversely, any complete set $\mathcal{S}$ of SE-interpretations can be represented by some program P. As a general result, taking also a restricted alphabet A into account, the following result holds:

Proposition 2 ([13]). *Let $\mathcal{S}$ be a complete set of SE-interpretations, and let A be a set of atoms. Then, there exists a program $P_{\mathcal{S},A} \in \mathcal{P}_A$ such that $SE(P_{\mathcal{S},A})|_A = \mathcal{S}|_A$.*

One possibility to obtain $P_{\mathcal{S},A}$ from $\mathcal{S}$ is as follows:

1. for each $Y \subseteq A$ with $(Y, Y) \notin \mathcal{S}|_A$, add rules $\bot \leftarrow Y, not\ (A \setminus Y)$, and
2. for each $X \subset Y$ with $(X, Y) \notin \mathcal{S}|_A$ and $(Y, Y) \in \mathcal{S}|_A$, add rules $\bigvee_{p \in (Y \setminus X)} p \leftarrow X, not\ (A \setminus Y)$.

3 Correspondence Checking

In order to deal with differing notions of program equivalence in a uniform manner, taking in particular strong equivalence and its relativised version, as well as equivalence notions based on the projection of answer sets into account, Eiter *et al.* [13] introduced a general framework for specifying differing notions of equivalence. In this framework, one parameterises, on the one hand, the class of programs used to be added to the programs under consideration, and, on the other hand, the relation that has to hold between the collection of answer sets of the extended programs.

Definition 1. *A* correspondence frame*, or simply* frame*, $\mathcal{F}$, is a triple $(\mathcal{U}, \mathcal{C}, \rho)$, where (i) $\mathcal{U}$ is a set of atoms, called the* universe of $\mathcal{F}$*, (ii) $\mathcal{C} \subseteq \mathcal{P}_\mathcal{U}$, called the* context of $\mathcal{F}$*, and (iii) $\rho \subseteq 2^{2^\mathcal{U}} \times 2^{2^\mathcal{U}}$.*

For every program $P, Q \in \mathcal{P}_\mathcal{U}$, P and Q are $\mathcal{F}$-corresponding*, in symbols $P \simeq_\mathcal{F} Q$, iff, for all $R \in \mathcal{C}$, $(\mathcal{AS}(P \cup R), \mathcal{AS}(Q \cup R)) \in \rho$.*

[1] We write abc instead of $\{a, b, c\}$, a instead of $\{a\}$, etc.

It is quite obvious that the equivalence notions presented in the previous section are special cases of $\mathcal{F}$-correspondence. Indeed, for any universe $\mathcal{U}$ and any $A \subseteq \mathcal{U}$, strong equivalence relative to A coincides with $(\mathcal{U}, \mathcal{P}_A, =)$-correspondence, and ordinary equivalence coincides with $(\mathcal{U}, \{\emptyset\}, =)$-correspondence.

Following Eiter *et al.* [13], we are mainly concerned with correspondence frames of form $(\mathcal{U}, \mathcal{P}_A, \subseteq_B)$ and $(\mathcal{U}, \mathcal{P}_A, =_B)$, where $A, B \subseteq \mathcal{U}$ are sets of atoms, and $\subseteq_B$ and $=_B$ are projections of the standard subset and set-equality relation, respectively, defined as follows: for any set $\mathcal{S}, \mathcal{S}'$ of interpretations, $\mathcal{S} \subseteq_B \mathcal{S}'$ iff $\mathcal{S}|_B \subseteq \mathcal{S}'|_B$, and $\mathcal{S} =_B \mathcal{S}'$ iff $\mathcal{S}|_B = \mathcal{S}'|_B$.

A *correspondence problem*, Π, (*over* $\mathcal{U}$) is a quadruple $(P, Q, \mathcal{C}, \rho)$, where $P, Q \in \mathcal{P}_\mathcal{U}$ and $(\mathcal{U}, \mathcal{C}, \rho)$ is a frame. We say that Π *holds* iff $P \simeq_{(\mathcal{U},\mathcal{C},\rho)} Q$ holds. For a correspondence problem $\Pi = (P, Q, \mathcal{C}, \rho)$ over $\mathcal{U}$, we usually leave $\mathcal{U}$ implicit, assuming that it consists of all atoms occurring in P, Q, and $\mathcal{C}$. We call Π an *equivalence problem* if ρ is given by $=_B$, and an *inclusion problem* if ρ is given by $\subseteq_B$, for some $B \subseteq \mathcal{U}$. Note that $(P, Q, \mathcal{C}, =_B)$ holds iff $(P, Q, \mathcal{C}, \subseteq_B)$ and $(Q, P, \mathcal{C}, \subseteq_B)$ jointly hold.

For inclusion problems, we define the concept of a *counterexample*, which is easily extended to equivalence problems.

Definition 2. *A pair* (Y, R)*, where* Y *is an interpretation and* $R \in \mathcal{C}$*, is a* counterexample *for* $(P, Q, \mathcal{C}, \subseteq_B)$ *iff* (*i*) $Y \in \mathcal{AS}(P \cup R)$ *and* (*ii*) $Z \notin \mathcal{AS}(Q \cup R)$*, for each* Z *with* $Z =_B Y$*.*

Example 2. We have already seen that for P, Q from Example 1, $(P, Q, \mathcal{P}_A, \subseteq_\mathcal{U})$ does not hold for $A = \{a, b\}$ and $\mathcal{U} = \{a, b, c, d\}$. What happens if we restrict the comparison of answer sets from $\mathcal{U}$ to A, i.e., does $(P, Q, \mathcal{P}_A, \subseteq_A)$ hold? Note that, e.g., $\mathcal{AS}(P \cup \{a \leftarrow\})|_A = \mathcal{AS}(Q \cup \{a \leftarrow\})|_A = \{ab\}$. Hence, the counterexample $(abc, \{a \leftarrow\})$ from Example 1 is no longer a counterexample for $(P, Q, \mathcal{P}_A, \subseteq_A)$. As we shall see below, there still exist counterexamples for this problem, but these are more involving. □

As shown by Eiter *et al.* [13], inclusion problems with projection may possess only counterexamples which are exponential in the size of the compared programs. Hence, instead of guessing concrete programs and checking whether they are counterexamples for a given inclusion problem, Eiter *et al.* provide a semantical structure, called *spoiler*, which operates on the compared programs alone, together with the notion of a *partial spoiler*.

Definition 3. *Let* $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ *be an inclusion problem,* Y *an interpretation, and* $\mathcal{S} \subseteq SE^A(Q) \cap \{(X, Z) \mid Z =_{A \cup B} Y\}$ *a complete set of* A*-SE-interpretations. The pair* $(Y, \mathcal{S})$ *is a* spoiler for Π *iff*

1. $(Y, Y) \in SE^A(P)$*,*
2. *each* $(Z, Z) \in SE^A(Q)$ *such that* $Z =_{A \cup B} Y$ *is also in* $\mathcal{S}$*,*
3. *for each* $(Z, Z) \in \mathcal{S}$*, some non-total* $(X, Z) \in \mathcal{S}$ *exists, and*
4. *for each non-total* $(X, Z) \in \mathcal{S}$*,* $(X, Y) \notin SE^A(P)$*.*

For a spoiler $(Y, \mathcal{S})$*, the interpretation* Y *is referred to as a* partial spoiler *for* Π*.*

Intuitively, in a spoiler $(Y, \mathcal{S})$ for $(P, Q, \mathcal{P}_A, \subseteq_B)$, the interpretation Y is an answer set of $P \cup R$ but not of $Q \cup R$, where R is a program which is semantically given by $\mathcal{S}$.

We collect and rephrase the main results from [13].

Proposition 3. *Let* $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ *be an inclusion problem. Then,* Π *holds iff there exists no spoiler for* Π.

As an immediate consequence, we obtain that a correspondence problem Π holds iff there exists no partial spoiler for Π. Moreover, we are able to connect spoilers to counterexamples using the generic programs $P_{\mathcal{S},A}$, as introduced in Section 2.

Proposition 4. *If* $(Y, \mathcal{S})$ *is a spoiler for an inclusion problem* $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$, *then* $(Y, P_{\mathcal{S},A})$ *is a counterexample for* Π.

Example 3. For P and Q from Example 1 and $A = \{a, b\}$, the pairs $(Y_1, \mathcal{S})$ and $(Y_2, \mathcal{S})$ are the only spoilers for $(P, Q, \mathcal{P}_A, \subseteq_A)$, where $Y_1 = \{abc\}$ and $Y_2 = \{abd\}$ are the partial spoilers for $(P, Q, \mathcal{P}_A, \subseteq_A)$, and $\mathcal{S} = \{(a, abd), (b, abc), (abc, abc), (abd, abd)\}$. Invoking our program construction, we obtain $P_{\mathcal{S},A} = \{\bot \leftarrow a, not\, b;\ \bot \leftarrow b, not\, a;\ \bot \leftarrow not\, a, not\, b;\ a \vee b \leftarrow\}$. One can verify that both Y_1 and Y_2 are contained in $\mathcal{AS}(P_1 \cup P_{\mathcal{S},A})$, while no interpretation Z with $Z =_A Y_1$ is an answer set of $Q \cup P_{\mathcal{S},A}$. □

Finally, we recall the computational complexity of checking whether an equivalence or inclusion problem holds. As shown by Eiter *et al.* [13], deciding $(P, Q, \mathcal{P}_A, =_B)$ is of a significantly higher complexity compared to more restricted notions of equivalence, like strong equivalence (which is coNP-complete) or ordinary equivalence and relativised strong equivalence (which both are Π_2^P-complete).

Proposition 5 ([13]). *Given programs* P, Q, *sets of atoms* A, B, *and* $\rho \in \{\subseteq_B, =_B\}$, *deciding whether a correspondence problem* $(P, Q, \mathcal{P}_A, \rho)$ *holds is* Π_4^P*-complete.*

4 Reductions

In this section, we provide two approaches to map inclusion problems into quantified Boolean formulas. By combining the reduction for a problem $(P, Q, \mathcal{P}_A, \subseteq_B)$ with that of $(Q, P, \mathcal{P}_A, \subseteq_B)$, we straightforwardly obtain a method to check whether an equivalence problem $(P, Q, \mathcal{P}_A, =_B)$ holds. We start with a brief recapitulation of the basic facts about the quantified version of propositional logic.

4.1 Quantified Propositional Logic

Quantified propositional logic is an extension of classical propositional logic in which formulas are permitted to contain quantifications over propositional variables. More formally, formulas of quantified propositional logic are built from atomic formulas using the primitive sentential connectives $\neg$ and $\wedge$, the logical constant $\top$, and unary operators of form $\forall p$ (where p is some atom), called *universal quantifiers*. The operators $\vee$, $\rightarrow$, and $\leftrightarrow$, as well as the symbol $\bot$, are defined from the primitive ones, $\neg$, $\wedge$, and $\top$, as usual. Furthermore, similar to first-order logic, $\exists p$ is defined as the operator

$\neg\forall p\neg$, referred to as an *existential quantifier*. Formulas of this language are also called *quantified Boolean formulas* (QBFs) and we denote them by Greek upper-case letters.

An occurrence of an atom p is *free* in a QBF Φ if it does not occur in the scope of a quantifier $\mathsf{Q}p$, $\mathsf{Q} \in \{\exists, \forall\}$. In what follows, we tacitly assume that every subformula $\mathsf{Q}p\Phi$ of a QBF contains a free occurrence of p in Φ, and for two different subformulas $\mathsf{Q}p\Phi$, $\mathsf{Q}q\Psi$ of a QBF we require $p \neq q$. Moreover, given a finite set P of atoms, $\mathsf{Q}P\Psi$ stands for any QBF $\mathsf{Q}p_1\mathsf{Q}p_2 \ldots \mathsf{Q}p_n\Psi$ such that the variables $p_1, \ldots, p_n$ are pairwise distinct and $P = \{p_1, \ldots, p_n\}$.

Towards the definition of the semantics of QBFs, we introduce the following notation: For an atom p (resp., a set P of atoms) and a set I of atoms, $\Phi[p/I]$ (resp., $\Phi[P/I]$) denotes the QBF resulting from Φ by replacing each free occurrence of p (resp., each $p \in P$) in Φ by $\top$ if $p \in I$ and by $\bot$ otherwise.

For an interpretation I and a QBF Φ, the relation $I \models \Phi$ is inductively defined as follows:

1. $I \models \top$,
2. $I \models p$ iff $p \in I$,
3. $I \models \neg\Phi$ iff $I \not\models \Phi$,
4. $I \models \Phi_1 \wedge \Phi_2$ iff $I \models \Phi_1$ and $I \models \Phi_2$, and
5. $I \models \forall p\, \Phi$ iff $I \models \Phi[p/\{p\}]$ and $I \models \Phi[p/\emptyset]$.

From these conditions, corresponding ones for $\bot$, $\vee$, $\rightarrow$, $\leftrightarrow$, and $\exists p$, for any p, follow in the usual way.

A QBF Φ is *true under* I iff $I \models \Phi$, otherwise Φ is *false under* I. A QBF is *valid* iff it is true under any interpretation. Note that a *closed* QBF, i.e., a QBF without free variable occurrences, is either true under any I or false under any I.

A QBF Φ is said to be in *prenex normal form* iff it is closed and of the form

$$\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1\, \phi, \tag{2}$$

where $n \geq 0$, ϕ is a propositional formula, $\mathsf{Q}_i \in \{\exists, \forall\}$ such that $\mathsf{Q}_i \neq \mathsf{Q}_{i+1}$ for $1 \leq i \leq n-1$, $(P_1, \ldots, P_n)$ is a partition of the propositional variables occurring in ϕ, and $P_i \neq \emptyset$, for each $1 \leq i \leq n$. We call a QBF of the form (2) an (n, Q_n)-*QBF*.

Without going into details, we mention that any closed QBF Φ is easily transformed into an equivalent QBF in prenex normal form such that each quantifier occurrence from the original QBF corresponds to a quantifier occurrence in the prenex normal form. Let us call such a QBF the *prenex normal form of* Φ. Similar as in first-order logic, however, there are different ways how to obtain an equivalent prenex QBF (cf. [8] for more details on this issue).

The following property is essential:

Proposition 6. *For every $k \geq 0$, deciding the truth of a given $(k, \exists)$-QBF (resp., $(k, \forall)$-QBF) is Σ_k^P-complete (resp., Π_k^P-complete).*

Hence, any decision problem $\mathcal{D}$ in Σ_k^P (resp., Π_k^P) can be mapped in polynomial time to a $(k, \exists)$-QBF (resp., $(k, \forall)$-QBF) Φ such that $\mathcal{D}$ holds iff Φ is valid. In particular, Proposition 5 implies thus that any correspondence problem $(P, Q, \mathcal{P}_A, \rho)$, for $\rho \in \{\subseteq_B, =_B\}$, can be reduced in polynomial time to a $(4, \forall)$-QBF. In what follows, we construct two such mappings which are actually constructible in *linear space and time*.

4.2 Encodings

For our encodings, we use the following building blocks. The idea hereby is to use sets of globally new atoms in order to refer to different assignments of the atoms from the compared program within a single formula. More formally, given an indexed set V of atoms, we assume (pairwise) disjoint copies $V_i = \{v_i \mid v \in V\}$, for every $i \geq 1$. Furthermore, we introduce the following abbreviations:

1. $(V_i \leq V_j) := \bigwedge_{v \in V}(v_i \rightarrow v_j)$,
2. $(V_i < V_j) := (V_i \leq V_j) \wedge \neg(V_j \leq V_i)$, and
3. $(V_i = V_j) := (V_i \leq V_j) \wedge (V_j \leq V_i)$.

Observe that the latter is clearly equivalent to $\bigwedge_{v \in V}(v_i \leftrightarrow v_j)$.

Proposition 7. *Let I be an interpretation, and $A, X, Y \subseteq V$ such that, for some i, j, $I|_{V_i} = X_i$ and $I|_{V_j} = Y_j$. Then,*

1. *$X|_A \subseteq Y|_A$ iff $I \models (A_i \leq A_j)$,*
2. *$X|_A \subset Y|_A$ iff $I \models (A_i < A_j)$, and*
3. *$X|_A = Y|_A$ iff $I \models (A_i = A_j)$.*

In accordance to the above renaming of atoms, we use subscripts as a general renaming schema for formulas and rules. That is, for each $i \geq 1$, α_i expresses the result of replacing each occurrence of an atom p in α by p_i, where α is any formula or rule. Furthermore, for a rule r 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 $\top$. Finally, for a program P, we define $P_{i,j} = \bigwedge_{r \in P}\big((B^+(r_i) \wedge B^-(r_j)) \rightarrow H(r_i)\big)$.

Proposition 8. *Let P be a program over atoms V, I an interpretation, and $X, Y \subseteq V$ such that, for some i, j, $I|_{V_i} = X_i$ and $I|_{V_j} = Y_j$. Then, $X \models P^Y$ iff $I \models P_{i,j}$.*

Intuitively, this allows to refer to the reduct of P (in case that $i \neq j$) and to the classical formula associated to P (in case that $i = j$) simultaneously.

The central characterisation towards our encodings, given next, replaces the concept of an A-SE-model in Definition 3 by tests over program reducts.

Proposition 9. *An interpretation Y is a partial spoiler for $(P, Q, \mathcal{P}_A, \subseteq_B)$ iff*

(a) $Y \models P$,
(b) for each $Y' \subset Y$ with $Y' =_A Y$, $Y' \not\models P^Y$, and
(c) for each $Z =_{A \cup B} Y$, $Z \models Q$ implies the existence of an $X \subset Z$ such that $X \models Q^Z$ and, if $X \subset Z|_A = Y|_A$, then, for each $X' \subseteq Y$ with $X' =_A X$, $X' \not\models P^Y$.

Definition 4. *Let P, Q be programs over V and let $A, B \subseteq V$. Furthermore, consider $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$. Then,*

$$\boldsymbol{S}_\Pi(V_1) := P_{1,1} \wedge \boldsymbol{S}^1(P, A) \wedge \forall V_3\big(\boldsymbol{S}^2(Q, A, B) \rightarrow \boldsymbol{S}^3(P, Q, A)\big), \text{ where}$$
$$\boldsymbol{S}^1(P, A) := \forall V_2\big(((A_2 = A_1) \wedge (V_2 < V_1)) \rightarrow \neg P_{2,1}\big),$$
$$\boldsymbol{S}^2(Q, A, B) := \big((A \cup B)_3 = (A \cup B)_1\big) \wedge Q_{3,3}, \text{ and}$$
$$\boldsymbol{S}^3(P, Q, A) := \exists V_4\big((V_4 < V_3) \wedge Q_{4,3} \wedge \big((A_4 < A_1) \rightarrow \forall V_5\big(((A_5 = A_4) \wedge (V_5 \leq V_1)) \rightarrow \neg P_{5,1}\big)\big)\big).$$

Lemma 1. *Let P and Q be programs over V, and let $A, B, Y \subseteq V$. Then, Y is a partial spoiler for $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ iff $Y_1 \models \boldsymbol{S}_\Pi(V_1)$.*

We do not give a formal proof here, but just provide the following explanations. The subformula $P_{1,1} \wedge \boldsymbol{S}^1(P, A)$ of $\boldsymbol{S}_\Pi(V_1)$ takes care of Conditions (a) and (b) from Proposition 9; we use atoms V_1 to refer to Y, and atoms V_2 to refer to the Y' therein. Note that $(A_2 = A_1) \wedge (V_2 < V_1)$ thus guarantees that we take only those Y' for testing $Y' \models P^Y$ into account, where $Y' \subset Y$ and $Y' =_A Y$. The next subformula, $\boldsymbol{S}^2(Q, A, B)$, "returns" all Z (via assignments to V_3) such that $Z =_{A \cup B} Y$ and $Z \models Q$. Finally, for each such Z, $\boldsymbol{S}^3(P, Q, A)$ has to be true. On account of $(V_4 < V_3)$, we let the assignments to V_4 (which refer to the X in Item (c) of Proposition 9) be a proper subset of those to V_3, i.e., we require $X \subset Z$. Then we test whether $X \models Q^Z$ via $Q_{4,3}$, as follows from Proposition 8, and in case that $X|_A \subset Y|_A$ (checked via $A_4 < A_3$), the remaining formula encodes the test whether for all X' (assignments to V_5) with $X' =_A X$ and $X' \subseteq Y$, $X' \not\models P^Y$, i.e., $P_{5,1}$ is false under the current assignment to V_1 and V_5.

In what follows, we give a more compact encoding, which in particular reduces the number of universal quantifications. The idea is to save on the fixed assignments as, e.g., in $\boldsymbol{S}^2(Q, A, B)$ where we have $(A \cup B)_3 = (A \cup B)_1$. That is, in $\boldsymbol{S}^2(Q, A, B)$, we implicitly ignore all assignments to V_3 where atoms from A or B have different truth values as those in V_1. Therefore, it makes sense to consider only atoms from $V_3 \setminus (A_3 \cup B_3)$ and using $A_1 \cup B_1$ instead of $A_3 \cup B_3$ in $Q_{3,3}$.

This calls for a more subtle renaming schema for programs, however. Let $\mathcal{V}$ be a set of indexed atoms, and let r be a rule. Then, $r^{\mathcal{V}}_{i,k}$ results from r by replacing each atom x in r by x_i, providing $x_i \in \mathcal{V}$, and by x_k otherwise. For a program P, we define

$$P^{\mathcal{V}}_{i,j,k} := \bigwedge_{r \in P} \big((B^+(r^{\mathcal{V}}_{i,k}) \wedge B^-(r^{\mathcal{V}}_{j,k})) \rightarrow H(r^{\mathcal{V}}_{i,k})\big).$$

Moreover, for every $i \geq 1$, every set V of atoms, and every set C, $V_i^C := (V \setminus C)_i$.

Definition 5. *Let P, Q be programs over V and $A, B \subseteq V$. Furthermore, let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem and $\mathcal{V} = V_1 \cup V_2^A \cup V_3^{A \cup B} \cup V_4 \cup V_5^A$. Then,*

$$\boldsymbol{T}_\Pi(V_1) := P_{1,1} \wedge \boldsymbol{T}^1(P, A, \mathcal{V}) \wedge \forall V_3^{A \cup B}\big(Q^{\mathcal{V}}_{3,1,1} \rightarrow \boldsymbol{T}^3(P, Q, A, \mathcal{V})\big), \text{ where}$$
$$\boldsymbol{T}^1(P, A, \mathcal{V}) := \forall V_2^A\big((V_2^A < V_1^A) \rightarrow \neg P^{\mathcal{V}}_{2,1,1}\big) \text{ and}$$
$$\boldsymbol{T}^3(P, Q, A, \mathcal{V}) := \exists V_4\big(\big(V_4 < ((A \cup B)_1 \cup V_3^{A \cup B})\big) \wedge Q^{\mathcal{V}}_{4,3,1} \wedge \big((A_4 < A_1) \rightarrow \forall V_5^A((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4})\big)\big).$$

Note that the subformula $V_4 < ((A \cup B)_1 \cup V_3^{A \cup B})$ in $\boldsymbol{T}^3(P, Q, A, \mathcal{V})$ denotes

$$\big(\big((A \cup B)_4 \leq (A \cup B)_1\big) \wedge (V_4 \leq V_1)\big) \wedge \neg\big(\big((A \cup B)_1 \leq (A \cup B)_4\big) \wedge (V_1 \leq V_4)\big).$$

Also note that, compared to our first encoding $\boldsymbol{S}_\Pi(V_1)$, we do not have a pendant to subformula $\boldsymbol{S}^2$ here, which reduces simply to $Q^{\mathcal{V}}_{3,1,1}$ due to the new renaming schema.

Lemma 2. *Let P, Q be programs over V, and let $A, B, Y \subseteq V$. Then, Y is a partial spoiler for $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ iff $Y_1 \models \boldsymbol{T}_\Pi(V_1)$.*

Example 4. Consider the two programs $P = \{a \vee b \leftarrow c\}$ and $Q = \{a \leftarrow c, not\, b\}$, $A = \{a\}$, and $B = \{b\}$. The encodings for the problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ are as follows:

$$\begin{aligned}
\boldsymbol{S}_\Pi(V_1) &= (c_1 \rightarrow a_1 \vee b_1) \wedge \boldsymbol{S}^1(P, A) \wedge \\
&\quad \forall a_3 b_3 c_3 (\boldsymbol{S}^2(Q, A, B) \rightarrow \boldsymbol{S}^3(P, Q, A)), \\
\boldsymbol{S}^1(P, A) &= \forall a_2 b_2 c_2 \big((a_2 \leftrightarrow a_1) \wedge (\{b_2, c_2\} < \{b_1, c_1\}) \rightarrow \neg(c_2 \rightarrow a_2 \vee b_2)\big), \\
\boldsymbol{S}^2(Q, A, B) &= (a_3 \leftrightarrow a_1) \wedge (b_3 \leftrightarrow b_1) \wedge (c_3 \wedge \neg b_3 \rightarrow a_3), \\
\boldsymbol{S}^3(P, Q, A) &= \exists a_4 b_4 c_4 \big((\{a_4, b_4, c_4\} < \{a_3, b_3, c_3\}) \wedge (c_4 \wedge \neg b_3 \rightarrow a_4) \wedge \\
&\qquad ((\{a_4\} < \{a_1\}) \rightarrow \forall a_5 b_5 c_5 ((a_5 \leftrightarrow a_4) \wedge \\
&\qquad (\{a_5, b_5, c_5\} \leq \{a_1, b_1, c_1\}) \rightarrow \neg(c_5 \rightarrow a_5 \vee b_5)))\big); \\
\boldsymbol{T}_\Pi(V_1) &= (c_1 \rightarrow a_1 \vee b_1) \wedge \boldsymbol{T}^1(P, A, \mathcal{V}) \wedge \\
&\quad \forall c_3 \big((c_3 \wedge \neg b_1 \rightarrow a_1) \rightarrow \boldsymbol{T}^3(P, Q, A, \mathcal{V})\big), \\
\boldsymbol{T}^1(P, A, \mathcal{V}) &= \forall b_2 c_2 ((\{b_2, c_2\} < \{b_1, c_1\}) \rightarrow \neg(c_2 \rightarrow a_1 \vee b_2), \\
\boldsymbol{T}^3(P, Q, A, \mathcal{V}) &= \exists a_4 b_4 c_4 \big((\{a_4, b_4, c_4\} < \{a_1, b_1, c_3\}) \wedge (c_4 \wedge \neg b_1 \rightarrow a_4) \wedge \\
&\qquad ((\{a_4\} < \{a_1\}) \rightarrow \forall b_5 c_5 ((\{b_5, c_5\} \leq \{b_1, c_1\}) \rightarrow \\
&\qquad \neg(c_5 \rightarrow a_4 \vee b_5)))\big).
\end{aligned}$$

As mentioned before, the optimised encoding $\boldsymbol{T}_\Pi(\cdot)$ saves "fixed assignments", like $(a_2 \leftrightarrow a_1)$, which occur in $\boldsymbol{S}_\Pi(\cdot)$, by employing the advanced renaming schema in such a way that, instead of atom a_2, atom a_1 is used in the encoding. One effect of this refinement is the decrease of universally quantified atoms. □

Theorem 1. *For any inclusion problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$, the following statements are equivalent: (i) Π holds; (ii) $\neg\exists V_1 \boldsymbol{S}_\Pi(V_1)$ is valid; and (iii) $\neg\exists V_1 \boldsymbol{T}_\Pi(V_1)$ is valid.*

Corollary 1. *Let $\Pi = (P, Q, \mathcal{P}_A, =_B)$ be an equivalence problem. Then, for $\Pi' = (P, Q, \mathcal{P}_A, \subseteq_B)$ and $\Pi'' = (Q, P, \mathcal{P}_A, \subseteq_B)$, the following statements are equivalent: (i) Π holds; (ii) $\neg\exists V_1 \boldsymbol{S}_{\Pi'}(V_1) \wedge \neg\exists V_1 \boldsymbol{S}_{\Pi''}(V_1)$ is valid; and (iii) $\neg\exists V_1 \boldsymbol{T}_{\Pi'}(V_1) \wedge \neg\exists V_1 \boldsymbol{T}_{\Pi''}(V_1)$ is valid.*

4.3 Applicability and Adequacy of the Encodings

In order to employ off-the-shelves QBF-solvers for deciding answer-set correspondence, we have to transform above encodings into prenex normal form. The propositional part of these prenex QBFs additionally has to be reduced to CNF, which can be accomplished by usual techniques. We thus focus here just on possible prenex normal forms of our encodings.

Recall that there are several ways to transform a QBF into prenex normal form. For our encodings, the situation is as follows. Take, e.g., the existential closure of $\boldsymbol{S}_\Pi(V_1)$, given by $\exists V_1 \boldsymbol{S}_\Pi(V_1)$: for this closed QBF, different prenex forms can be obtained, e.g.,

$$\exists V_1 \forall (V_2 \cup V_3) \exists V_4 \forall V_5\, \phi \quad \text{or} \quad \exists V_1 \forall V_3 \exists V_4 \forall (V_5 \cup V_2)\, \phi,$$

where ϕ represents the so-called *propositional skeleton* of the QBF $\boldsymbol{S}_\Pi(V_1)$ (cf. [8]), which, roughly speaking, results from $\boldsymbol{S}_\Pi(V_1)$ by deleting all quantifiers. For later purposes, we use in the following the second variant, and define $\boldsymbol{S}^{\mathsf{p}}_\Pi := \exists V_1 \forall V_3 \exists V_4 \forall (V_5 \cup V_2)\, \phi$. Likewise, we use $\boldsymbol{T}^{\mathsf{p}}_\Pi := \exists V_1 \forall V_3^{A \cup B} \exists V_4 \forall (V_5^A \cup V_2^A)\, \psi$ as a prenex form for $\exists V_1 \boldsymbol{T}_\Pi(V_1)$, where ψ is the propositional skeleton of $\boldsymbol{T}_\Pi(V_1)$.

Theorem 2. *For any inclusion problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$, the following statements are equivalent: (i) Π holds; (ii) $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ is valid; (iii) $\neg \boldsymbol{T}^{\mathsf{p}}_\Pi$ is valid.*

These prenex forms also give evidence that our encodings are *adequate* in a certain theoretical sense: Following [3], given decision problems $\mathcal{D} \subseteq \mathcal{L}$ and $\mathcal{D}' \subseteq \mathcal{L}'$ in languages $\mathcal{L}$ and $\mathcal{L}'$, respectively, we call an encoding $f : \mathcal{L} \to \mathcal{L}'$ *adequate* iff, for each $s \in \mathcal{L}$, (i) $s \in \mathcal{D}$ iff $f(s) \in \mathcal{D}'$, (ii) $f(s)$ is constructible in polynomial time from s, and (iii) deciding whether $f(s) \in \mathcal{D}'$ is not computationally harder than deciding whether $s \in \mathcal{D}$.

Now, both $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ and $\neg \boldsymbol{T}^{\mathsf{p}}_\Pi$ obviously satisfy these conditions, for every inclusion problem Π. Indeed, by the above theorem, we have that Π holds iff $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ is valid. Moreover, $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ is computable in polynomial time (indeed, in linear time) in the size of Π (as easily verified from the definitions). Finally, $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ can be transformed into a $(4, \forall)$-QBF in polynomial time. Hence, Proposition 6 implies that determining the truth of $\neg \boldsymbol{S}^{\mathsf{p}}_\Pi$ is thus in the same complexity class (viz. Π^P_4) as the encoded problem. All these properties hold for $\neg \boldsymbol{T}^{\mathsf{p}}_\Pi$ as well. This proves the adequacy of our encodings.

5 Obtaining Counterexamples

In this section, we provide a theoretical basis how to use our encodings to obtain counterexamples for an inclusion problem $(P, Q, \mathcal{P}_A, \subseteq_B)$. To this end, we use the concept of *policies* for prenex QBFs, along the lines of Coste-Marquis *et al.* [4].

Definition 6. *The set $P(k, \mathsf{Q}, X_k, \ldots, X_1)$ of* policies *for a (k, Q)-QBF of the form $\mathsf{Q}_k X_k \ldots \mathsf{Q}_1 X_1 \phi$ is inductively defined as follows:*

1. $P(0, \mathsf{Q}) = \{\lambda\}$,
2. $P(k, \exists, X_k, \ldots, X_1) = \{(I, \pi) \mid I \subseteq X_k,\ \pi \in P(k-1, \forall, X_{k-1}, \ldots, X_1)\}$, *and*
3. $P(k, \forall, X_k, \ldots, X_1) = \{\pi \mid \pi : 2^{X_k} \to P(k-1, \exists, X_{k-1}, \ldots, X_1)\}$,

where λ represents the empty policy.

Note that policies for $(k, \exists)$-QBFs are pairs (I, π), where I is an interpretation over atoms from the outermost group of quantifiers and π is a policy itself, whereas policies for $(k, \forall)$-QBFs are functions assigning to each interpretation over atoms from the outermost group of quantifiers a policy.

Definition 7. *A (k, Q)-QBF $\Phi = \mathsf{Q}_k X_k \ldots \mathsf{Q}_1 X_1 \phi$ is* satisfied *by a policy π (for Φ) iff one the following conditions applies (inductively):*

1. *$k = 0$, $\pi = \lambda$, and ϕ is true,*
2. *$k > 0$, $\mathsf{Q} = \exists$, $\pi = (I, \pi')$, and $\forall X_{k-1} \ldots \mathsf{Q}_1 X_1 \phi[X_k / I]$ is satisfied by π',*
3. *$k > 0$, $\mathsf{Q} = \forall$, and for any $I \subseteq X_k$, $\exists X_{k-1} \ldots \mathsf{Q}_1 X_1 \phi[X_k / I]$ is satisfied by $\pi(I)$.*

Denote by $SP(\Phi)$ the set of satisfying policies for a prenex QBF Φ.

Proposition 10. *A prenex QBF Φ is valid iff $SP(\Phi) \neq \emptyset$.*

Example 5. Consider $\phi = (p \rightarrow q) \wedge (q \rightarrow p)$ and the following QBFs:[2]

$$\Phi_1 = \exists pq\, \phi, \quad \Phi_2 = \forall pq\, \phi, \quad \Phi_3 = \exists p \forall q\, \phi, \quad \text{and} \quad \Phi_4 = \forall p\, \exists q\, \phi.$$

The set of policies for Φ_1 is given by $\{(I, \lambda) \mid I \subseteq \{p, q\}\}$, i.e., the satisfying policies for Φ_1 are in a one-to-one correspondence to the models of ϕ, and are given by $(\emptyset, \lambda)$ and $(\{p, q\}, \lambda)$. For Φ_2, the only policy is the function π assigning to each $I \subseteq \{p, q\}$ the empty policy λ. Note that π is not satisfying Φ_2 since, for instance, with $I = \{p\}$, we get $\pi(I) = \lambda$, but $\phi[\{p, q\}/I] = (\top \rightarrow \bot) \wedge (\bot \rightarrow \top)$ is not true. For Φ_3, we get as policies $\pi_1 = (\{p\}; \pi')$ and $\pi_2 = (\emptyset; \pi')$, where π' is defined as $\pi'(\{q\}) = \pi'(\emptyset) = \lambda$. It can be shown that neither π_1 nor π_2 satisfy Φ_3, by similar arguments as for the case of Φ_2. Finally, Φ_4 yields four policies, given as follows:

$$\begin{array}{llll} \pi(p) = (q, \lambda), & \pi(\emptyset) = (q, \lambda); & \pi'(p) = (q, \lambda), & \pi'(\emptyset) = (\emptyset, \lambda); \\ \pi''(p) = (\emptyset, \lambda), & \pi''(\emptyset) = (q, \lambda); & \pi'''(p) = (\emptyset, \lambda), & \pi'''(\emptyset) = (\emptyset, \lambda). \end{array}$$

One can verify that π' is the only satisfying policy for Φ_4. □

We now use the concept of policies to obtain the counterexamples from the satisfying policies of our encodings. Note that, in the definition below, we make use of our renaming schema as used in the encodings; e.g., $Z_3 = \{z_3 \mid z \in Z\}$.

Definition 8. *Let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem. Furthermore, let $\boldsymbol{S}^{\mathsf{p}}_{\Pi}$ and $\boldsymbol{T}^{\mathsf{p}}_{\Pi}$ be as in Subsection 4.3 and $\Omega \in \{\boldsymbol{S}, \boldsymbol{T}\}$. Then,*

$$\sigma(\Omega, \Pi) := \{(Y, \Sigma_{\Omega, Y, \pi}) \mid (Y_1, \pi) \in SP(\Omega^{\mathsf{p}}_{\Pi})\},$$

where

$$\begin{array}{rl} \Sigma_{\boldsymbol{S}, Y, \pi} := & \{(X, Z), (Z, Z) \mid Z =_{A \cup B} Y, (Z, Z) \in SE^A(Q), \\ & \quad \pi(Z_3) = (X_4, \pi'), \textit{ for some } \pi'\} \quad \textit{and} \\ \Sigma_{\boldsymbol{T}, Y, \pi} := & \{(X, Y \dot{+} Z), (Y \dot{+} Z, Y \dot{+} Z) \mid (Y \dot{+} Z, Y \dot{+} Z) \in SE^A(Q), \\ & \quad \pi(Z_3) = (X_4, \pi'), \textit{ for some } \pi'\}, \end{array}$$

and $Y \dot{+} Z$ stands for $Y|_{A \cup B} \cup Z$.

These two projections, $\sigma(\boldsymbol{S}, \cdot)$ and $\sigma(\boldsymbol{T}, \cdot)$, on the satisfying policies for our two encodings are actually identical. Hence, our final two results in this section apply to both encodings.

Theorem 3. *Let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem and $\Omega \in \{\boldsymbol{S}, \boldsymbol{T}\}$. Then, each $(Y, \Sigma) \in \sigma(\Omega, \Pi)$ is a spoiler for Π.*

In view of the construction of Proposition 2, we can thus construct counterexamples directly from the satisfying policies of our encodings.

[2] In what follows, we sometimes omit brackets "{" and "}" for ease of notation.

Corollary 2. *Let* $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ *be an inclusion problem and* $\Omega \in \{\boldsymbol{S}, \boldsymbol{T}\}$. *Then, each* $(Y, \Sigma) \in \sigma(\Omega, \Pi)$ *induces a counterexample* $(Y, P_{\Sigma,A})$ *for* Π.

From Proposition 10 and Theorem 2, in turn, we obtain that in case no satisfying policy for our encodings exists, the considered inclusion problem holds, and therefore does not possess any counterexample.

6 Special Cases

Finally, we analyse our encodings in the light of special instantiations of correspondence problems and give pointers to related work.

In what follows, for every equivalence problem $\Pi = (P, Q, \mathcal{P}_A, =_B)$, let $\Pi' = (P, Q, \mathcal{P}_A, \subseteq_B)$ and $\Pi'' = (Q, P, \mathcal{P}_A, \subseteq_B)$ be the associated inclusion problems (see also Corollary 1).

In case of *strong equivalence* [20], i.e., for problems of form $\Pi = (P, Q, \mathcal{P}_A, =_A)$ with $A = \mathcal{U}$, the encodings $\boldsymbol{T}_{\Pi'}(V_1)$ and $\boldsymbol{T}_{\Pi''}(V_1)$, as defined in Definition 5, can be drastically simplified since $V_2^A = V_3^A = V_5^A = \emptyset$. In particular, $\boldsymbol{T}_{\Pi'}(V_1)$ is equivalent to

$$P_{1,1} \wedge \big(Q_{1,1} \rightarrow \exists V_4\big((V_4 < V_1) \wedge Q_{4,1} \wedge \neg P_{4,1}\big)\big).$$

Note that the composed encoding for deciding strong equivalence, i.e., the closed QBF $\neg\exists V_1 \boldsymbol{T}_{\Pi'}(V_1) \wedge \neg\exists V_1 \boldsymbol{T}_{\Pi''}(V_1)$, amounts to a propositional unsatisfiability test, witnessing the coNP-completeness complexity for checking strong equivalence [24]. One can show that the reductions due to Pearce *et al.* [24] and Lin [21] for testing strong equivalence in terms of propositional logic are simple variants thereof.

For strong equivalence *relative* to a set A of atoms [28], i.e., for Π being of form $(P, Q, \mathcal{P}_A, =_B)$ with $B = \mathcal{U}$ but with arbitrary A, our encodings $\boldsymbol{T}_{\Pi'}(V_1)$ and $\boldsymbol{T}_{\Pi''}(V_1)$ can still be simplified since $V_3^{A \cup B} = \emptyset$. Indeed, $\boldsymbol{T}^{\mathsf{p}}_{\Pi'}$ and $\boldsymbol{T}^{\mathsf{p}}_{\Pi''}$ are then $(2, \exists)$-QBFs, reflecting the complexity of strong equivalence relative to A, which is on the second level of the polynomial hierarchy [28].

Next, we address the case of *bounded* relativised strong equivalence, as investigated by Eiter *et al.* [11]. This notion applies to problems of form $\Pi = (P, Q, \mathcal{P}_A, =)$, where the cardinality of $(\mathcal{U} \setminus A)$, i.e., the number of atoms missing in A, is bounded by a constant. Hereby, the sets V_2^A and V_5^A, which build the only universal quantifiers in the encoding $\boldsymbol{T}_{\Pi'}(V_1)$ for relativised strong equivalence, are sets of a fixed size. Hence, we can eliminate these quantifiers according to the semantics and still get an adequate encoding for this particular notion of equivalence. Consequently, bounded relativised strong equivalence can be checked with a polynomial unsatisfiability test, once again reflecting the coNP-complexity of this problem [11].

Finally, we address the case of ordinary equivalence, i.e., considering problems of form $\Pi = (P, Q, \mathcal{P}_A, =)$ with $A = \emptyset$, which is well known to be Π_2^P-complete [23]. Here, the encoding $\boldsymbol{S}_{\Pi'}(V_1)$ from Definition 4 can be simplified as follows:

$$P_{1,1} \wedge \forall V_2\big((V_2 < V_1) \rightarrow \neg P_{2,1}\big) \wedge \big(Q_{1,1} \rightarrow \exists V_4((V_4 < V_1) \wedge Q_{4,1})\big).$$

One can observe that this encoding is related to encodings for computing stable models via QBFs, as discussed by Egly *et al.* [6] and Pearce *et al.* [24]. Indeed, taking the two

main conjuncts from $\boldsymbol{S}_{\Pi'}(V_1)$, $\Phi = P_{1,1} \wedge \forall V_2\big((V_2 < V_1) \rightarrow \neg P_{2,1}\big)$ and $\Psi = Q_{1,1} \rightarrow \exists V_4\big((V_4 < V_1) \wedge Q_{4,1})\big)$, we get, for any assignment $Y_1 \subseteq V_1$, $Y_1 \models \Phi$ iff Y is an answer set of P, and $Y_1 \models \Psi$ iff Y is not an answer set of Q. Note that once more the encodings reflect the inherent complexity of the reduced equivalence checking task, viz. the Π_2^P-completeness for ordinary equivalence in this case.

7 Conclusion

In this paper, we discussed a novel decision procedure for advanced program comparison in answer-set programming (ASP) via encodings into quantified propositional logic. This approach was motivated by the high computational complexity we have to face for this task, making a direct realisation via ASP hard to accomplish. Furthermore, we showed how to obtain counterexamples from policies, which satisfy these encodings, and discussed special instances of the considered correspondence problems. Since currently practicably efficient solvers for quantified propositional logic are available, they can be used as back-end inference engines to compute the correspondence problems under consideration using the proposed encodings. Moreover, since these correspondence problems are one of the few natural problems lying above the second level of the polynomial hierarchy, yet still part of the polynomial hierarchy, we believe that our encodings also provide valuable benchmarks for evaluating QBF-solvers, for which there is currently a lack of structured problems with more than one quantifier alternation (see [17,16]).

References

1. O. Arieli. Paraconsistent Preferential Reasoning by Signed Quantified Boolean Formulae. In *Proc. ECAI'04*, pages 773–777. IOS Press, 2004.
2. O. Arieli and M. Denecker. Reducing Preferential Paraconsistent Reasoning to Classical Entailment. *Journal of Logic and Computation*, 13(4):557–580, 2003.
3. P. Besnard, T. Schaub, H. Tompits, and S. Woltran. Representing Paraconsistent Reasoning via Quantified Propositional Logic. In *Inconsistency Tolerance*, volume 3300 of *LNCS*, pages 84–118. Springer, 2005.
4. S. Coste-Marquis, H. Fargier, J. Lang, D. Le Berre, and P. Marquis. Function Problems for Quantified Boolean Formulas. Technical Report 2003-15-R, Institut de Recherche en Informatique de Toulouse (IRIT), 2003. Available at `http://www.cril.univ-artois.fr/asqbf/pub/files/qbfeng7.pdf`.
5. J. Delgrande, T. Schaub, H. Tompits, and S. Woltran. On Computing Solutions to Belief Change Scenarios. *Journal of Logic and Computation*, 14(6):801–826, 2004.
6. U. Egly, T. Eiter, H. Tompits, and S. Woltran. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In *Proc. AAAI'00*, pages 417–422. AAAI Press/MIT Press, 2000.
7. U. Egly, R. Pichler, and S. Woltran. On Deciding Subsumption Problems. *Annals of Mathematics and Artificial Intelligence*, 43(1–4):255–294, 2005.
8. U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. SAT'03, Selected Revised Papers*, volume 2919 of *LNCS*, pages 214–228. Springer, 2004.

9. T. Eiter, W. Faber, M. Fink, G. Pfeifer, and S. Woltran. Complexity of Answer Set Checking and Bounded Predicate Arities for Non-ground Answer Set Programming. In *Proc. KR'04*, pages 377–387. AAAI Press, 2004.
10. T. Eiter and M. Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proc. ICLP'03*, number 2916 in LNCS, pages 224–238. Springer, 2003.
11. T. Eiter, M. Fink, and S. Woltran. Semantical Characterizations and Complexity of Equivalences in Answer Set Programming. Technical Report INFSYS RR-1843-05-01, Institut für Informationssysteme, Technische Universität Wien, Austria, 2005. To appear in *ACM Transactions on Computational Logic*.
12. T. Eiter, V. Klotz, H. Tompits, and S. Woltran. Modal Nonmonotonic Logics Revisited: Efficient Encodings for the Basic Reasoning Tasks. In *Proc. TABLEAUX'02*, volume 2381 of *LNCS*, pages 100–114. Springer, 2002.
13. T. Eiter, H. Tompits, and S. Woltran. On Solution Correspondences in Answer Set Programming. In *Proc. IJCAI'05*, 2005.
14. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.
15. E. Giunchiglia, M. Narizzano, and A. Tacchella. Backjumping for Quantified Boolean Logic Satisfiability. *Artificial Intelligence*, 145:99–120, 2003.
16. D. Le Berre, M. Narizzano, L. Simon, and A. Tacchella. The Second QBF Solvers Comparative Evaluation, 2004. Available at `http://www.qbflib.org/`.
17. D. Le Berre, L. Simon, and A. Tacchella. Challenges in the QBF Arena: the SAT'03 Evaluation of QBF Solvers. In *Proc. SAT'03, Selected Revised Papers*, volume 2919 of *LNCS*, pages 468–485. Springer, 2004.
18. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. Technical Report cs.AI/0211004, arXiv.org. To appear in *ACM Transactions on Computational Logic*.
19. R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proc. TABLEAUX'02*, volume 2381 of *LNCS*, pages 160–175. Springer, 2002.
20. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
21. F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. KR'02*, pages 170–176. Morgan Kaufmann, 2002.
22. F. Lin and Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In *Proc. AAAI'02*, pages 112–117. AAAI Press / MIT Press, 2002.
23. E. Oikarinen and T. Janhunen. Verifying the Equivalence of Logic Programs in the Disjunctive Case. In *Proc. LPNMR'04*, volume 2923 of *LNCS*, pages 180–193. Springer, 2004.
24. D. Pearce, H. Tompits, and S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA'01*, volume 2258 of *LNCS*, pages 306–320. Springer, 2001.
25. J. Rintanen. Constructing Conditional Plans by a Theorem Prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.
26. P. Simons, I. Niemelä, and T. Soininen. Extending and Implementing the Stable Model Semantics. *Artificial Intelligence*, 138:181–234, 2002.
27. H. Turner. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *Theory and Practice of Logic Programming*, 3(4-5):602–622, 2003.
28. S. Woltran. Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In *Proc. JELIA'04*, volume 3229 of *LNCS*, pages 161–173. Springer, 2004.