A Tool for Advanced Correspondence Checking in Answer-Set Programming*

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

Martina Seidl
Institut für Softwaretechnik 188/3,
Technische Universität Wien,
Favoritenstraße 9-11,
A-1040 Vienna, Austria
seidl@big.tuwien.ac.at

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 previous work, a general framework for specifying correspondences between logic programs under the answer-set semantics has been defined. The framework allows to define different notions of equivalence, including well-known notions like *strong equivalence* as well as refined ones 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 implementation to verify program correspondences in this general framework. The system, called cc⊤, relies on linear-time constructible reductions to *quantified propositional logic* using extant solvers for the latter language as back-end inference engines. We provide some preliminary performance evaluation which shed light on some crucial design issues.

Introduction

Nonmonotonic logic programs under the answer-set semantics (Gelfond & Lifschitz 1991), with which we are dealing with in this paper, represent the canonical and, due to the availability of efficient answer-set solvers, arguably most widely used approach to answer-set programming (ASP). The latter paradigm is based on the idea that problems are encoded in terms of theories such that the solutions of a given problem are determined by the models ("answer sets") of the corresponding theory. Logic programming under the answer-set semantics has become an important host for solving many AI problems, including planning, diagnosis, and inheritance reasoning (cf. Gelfond & Leone (2002) for an overview).

To support engineering tasks of ASP solutions, an important issue is to determine the equivalence of different problem encodings. To this end, various notions of equivalence between programs under the answer-set semantics have been studied in the literature, including the recently proposed framework by Eiter, Tompits, & Woltran (2005), which subsumes most of the previously introduced notions. Within this framework, correspondence between two programs, P and Q, holds iff the answer sets of $P \cup R$ and $Q \cup R$ satisfy certain criteria, for any program R in a specified class, called the *context*. We shall focus here on correspondence problems where both the context and the comparison between answer sets are determined in terms of *alphabets*. This kind of program correspondence includes, as special instances, the well-known notions of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001), *uniform equivalence* (Eiter & Fink 2003), its relativised variants thereof (Woltran 2004), as well as the practicably important case of program comparison under *projected* answer sets. In the last 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.

For illustration, consider the following two programs which both express the selection of exactly one of the atoms a, b. An atom can only be selected if it can be derived together with the context:

$$\begin{aligned} P = \{\ & sel(b) \leftarrow b, not\ \ out(b); \\ & sel(a) \leftarrow a, not\ \ out(a); \\ & out(a) \vee out(b) \leftarrow a, b; \}. \end{aligned}$$

$$\begin{aligned} Q = \{\ & fail \leftarrow sel(a), not\ a, not\ fail; \\ & fail \leftarrow sel(b), not\ b, not\ fail; \\ & sel(a) \vee sel(b) \leftarrow a; \\ & sel(a) \vee sel(b) \leftarrow b; \}. \end{aligned}$$

Both programs use "local" atoms, $out(\cdot)$ and $fail$, respectively, which are expected not to appear in the context. In order to compare the programs, we could specify an alphabet, A, for the context, for instance $A = \{a, b\}$, or, more generally, any set A of atoms not containing the local atoms $out(a)$, $out(b)$, and $fail$. On the other hand, we want to check whether, for each addition of a context program over A, the answer sets correspond when taking only atoms from $B = \{sel(a),\ sel(b)\}$ into account.

In this paper, we report about an implementation of such correspondence problems together with some initial experimental results. The overall approach of the system, which

*This work was partially supported by the Austrian Science Fund (FWF) under grant P18019; the second author was also supported by the Austrian Federal Ministry of Transport, Innovation, and Technology (BMVIT) and the Austrian Research Promotion Agency (FFG) under grant FIT-IT-810806.

we call cc$\top$ ("correspondence-checking tool"), is to reduce the problem of correspondence checking to the satisfiability problem of *quantified propositional logic*, 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.

The motivation to use such an approach is twofold. First, complexity results (Eiter, Tompits, & Woltran 2005) show that correspondence checking within this framework is hard, lying on the fourth level of the polynomial hierarchy. This indicates that implementations of such checks cannot be realised in a straightforward manner using ASP systems themselves. In turn, it is well known that decision problems from the polynomial hierarchy can be efficiently represented in terms of QBFs in such a way that determining the validity of the resultant QBFs is not computationally harder than checking the original problem. In previous work (Tompits & Woltran 2005), such translations from correspondence checking to QBFs have been developed; moreover, they are constructible in *linear time and space*. Second, various practicably efficient solvers for quantified propositional logic are currently available (see, e.g., Le Berre *et al.* (2005)). Hence, such tools are used as back-end inference engines in our system to verify the correspondence problems under consideration.

We note that reduction methods to QBFs have been successfully applied already in the field of nonmonotonic reasoning (Egly *et al.* 2000; Delgrande *et al.* 2004), paraconsistent reasoning (Besnard *et al.* 2005; Arieli & Denecker 2003), and planning (Rintanen 1999).

Previous systems implementing different forms of equivalence, being special cases of correspondence notions in the framework of Eiter, Tompits, & Woltran (2005), also based on a reduction approach, are SELP (Chen, Lin, & Li 2005) and DLPEQ (Oikarinen & Janhunen 2004). Concerning SELP, here the problem of checking strong equivalence is reduced to propositional logic, making use of SAT solvers as back-end inference engines. Our system generalises SELP in the sense that cc$\top$ handles a correspondence problem which coincides with a test for strong equivalence by the same reduction as used in SELP. The system DLPEQ, on the other hand, is capable of comparing disjunctive logic programs under ordinary equivalence. Here, the reduction of a correspondence problem results in further logic programs such that the latter have no answer set iff the encoded problem holds. Hence, this system uses answer-set solvers themselves in order to check for equivalence.

The methodologies of both of the above systems have in common that their range of applicability is restricted to very special forms of program correspondences, while our new system cc$\top$ provides a wide range of more fine-grained equivalence notions, allowing practical comparisons useful for debugging and modular programming.

The outline of the paper is as follows. We start with recapitulating the basic facts about logic programs under the answer-set semantics and quantified propositional logic. In describing how to implement correspondence problems, we first give a detailed review of the encodings, followed by a discussion how these encodings (and thus the present system) behave in the case the specified correspondence coincides with special equivalence notions. Then, we address some technical questions which arise when applying the encodings to QBF solvers which require its input to be in a certain normal form. Finally, we present the concrete system cc$\top$ and illustrate its usage. The penultimate section is devoted to experimental evaluation and comparisons. We conclude with some final remarks and pointers to future work.

Preliminaries

Throughout the paper, we use the following notation: For an interpretation I (i.e., a set of atoms) and a set $\mathcal{S}$ of interpretations, we write $\mathcal{S}|_I = \{Y \cap I \mid Y \in \mathcal{S}\}$. For a singleton set $\mathcal{S} = \{Y\}$, we write $Y|_I$ instead of $\mathcal{S}|_I$, if convenient.

Logic Programs

We are concerned with *propositional disjunctive logic programs* (DLPs) which are finite sets of rules of form

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

$n \geq m \geq l \geq 0$, where all a_i are propositional atoms from some fixed universe $\mathcal{U}$ and not denotes default negation. 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 $\mathcal{P}_A$.

Following Gelfond & Lifschitz (1991), an interpretation I is an *answer set* of a program P iff it is a minimal model of the *reduct* P^I, resulting from P by

- deleting all rules containing default negated atoms $not\, a$ such that $a \in I$, and
- deleting all default negated atoms in the remaining rules.

The collection of all answer sets of a program P is denoted by $\mathcal{AS}(P)$.

In order to semantically compare programs, different notions of equivalence have been introduced in the context of the answer-set semantics. Besides *ordinary equivalence* between programs, which checks whether two programs have the same answer sets, the more restrictive notions of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001) and *uniform equivalence* (Eiter & Fink 2003) have been introduced. Two programs, P and Q, are strongly equivalent iff $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$, for any program R, and they are uniformly equivalent iff $\mathcal{AS}(P \cup R) = \mathcal{AS}(Q \cup R)$, for any set R of *facts*, i.e., rules of form $a \leftarrow$, for some atom a. Also, relativised equivalence notions, taking the alphabet of the extension set R into account, have been defined (Woltran 2004).

In abstracting from these notions, Eiter, Tompits, & Woltran (2005) introduced a general framework for specifying differing notions of program correspondence. In this framework, one parameterises, on the one hand, the *context*, i.e., 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. More formally, the following definition has been introduced:

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

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

Clearly, the equivalence notions mentioned above 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, Tompits, & Woltran (2005), we are 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 correspondence 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.

The next proposition summarises the complexity landscape within this framework (Eiter, Tompits, & Woltran 2005; Pearce, Tompits, & Woltran 2001; Woltran 2004).

Proposition 1 *Given programs* P *and* Q*, sets of atoms* A *and* B*, and* $\rho \in \{\subseteq_B, =_B\}$*, deciding whether a correspondence problem* $(P, Q, \mathcal{P}_A, \rho)$ *holds is:*

1. Π_4^P*-complete, in general;*
2. Π_3^P*-complete, for* $A = \emptyset$*;*
3. Π_2^P*-complete, for* $B = \mathcal{U}$*;*
4. coNP*-complete for* $A = \mathcal{U}$.

While Case 1 provides the result in the general setting, for the other cases we have the following: Case 2 amounts to *ordinary equivalence with projection*, i.e., the answer sets of two programs relative to a specified set B of atoms are compared. Case 3 amounts to *strong equivalence relative to* A and includes, as a special case, viz. for $A = \emptyset$, *ordinary equivalence*. Finally, Case 4 includes *strong equivalence* (for $B = \mathcal{U}$) as well as strong equivalence with projection.

The Π_4^P-hardness result shows that, in general, checking the correspondence of two programs cannot (presumably) be efficiently encoded in terms of ASP, which has its basic reasoning tasks located at the second level of the polynomial hierarchy (i.e., they are contained in Σ_2^P or Π_2^P). However, correspondence checking can be efficiently encoded in terms of *quantified propositional logic*, whose basic concepts we recapitulate next.

Quantified Propositional Logic

Quantified propositional logic is an extension of classical propositional logic in which formulas are permitted to contain quantifications over propositional variables. In particular, this language contains, for any atom p, unary operators of form $\forall p$ and $\exists p$, called *universal* and *existential quantifiers*, respectively, where $\exists p$ is defined as $\neg\forall p\neg$. Formulas of this language are also called *quantified Boolean formulas* (QBFs), and we denote them by Greek upper-case letters.

Given a QBF $\mathsf{Q}p\,\Psi$, for $\mathsf{Q} \in \{\exists, \forall\}$, we call Ψ the *scope* of $\mathsf{Q}p$. An occurrence of an atom p is *free* in a QBF Φ if it does not occur in the scope of a quantifier $\mathsf{Q}p$ in Φ. 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 $P = \{p_1, \ldots, p_n\}$. Finally, 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 in classical propositional logic, whereby universal quantifiers are evaluated as follows:

$$I \models \forall p\ \Phi \text{ iff } I \models \Phi[p/\{p\}] \text{ and } I \models \Phi[p/\emptyset].$$

A QBF Φ is *true under* I iff $I \models \Phi$, otherwise Φ is *false under* I. A QBF is *satisfiable* iff it is true under at least one interpretation. 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 interpretation or false under any interpretation.

A QBF Φ is said to be in *prenex normal form* (PNF) 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 say that Φ is in *prenex conjunctive normal form* (PCNF) iff Φ is of the form (2) and ϕ is in conjunctive normal form. Furthermore, a QBF of form (2) is also referred to as an (n, Q_n)*-QBF*. 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. However, there are different ways to obtain an equivalent prenex QBF (cf. Egly *et al.* (2004) for more details on this issue). The following property is essential:

Proposition 2 *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 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. Our implemented tool, described next, relies on two such mappings, which are actually constructible in *linear space and time*.

Computing Correspondence Problems

We now describe the system cc$\top$, which allows to verify the correspondence of two programs. It relies on efficient reductions from correspondence problems to QBFs as developed by Tompits & Woltran (2005). These encodings are presented in the first subsection. Then, we discuss how the encodings behave if the specified correspondence problem coincides with special forms of inclusion or equivalence problems, viz. those restricted cases discussed in Proposition 1. Afterwards, we give details concerning the transformation of the resultant QBFs into PCNF, which is necessary because most extant QBF solvers rely on input of this form. Finally, we give some details concerning the general syntax and invocation of the cc$\top$ tool.

Basic Encodings

Following Tompits & Woltran (2005), we consider two different reductions from inclusion problems to QBFs, $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, where $\mathsf{T}[\cdot]$ can be seen as an explicit optimisation of $\mathsf{S}[\cdot]$. Recall that equivalence problems can be decided by the composition of two inclusion problems. Thus, a composed encoding for equivalence problems is easily obtained via a conjunction of two particular instantiations of $\mathsf{S}[\cdot]$ (or $\mathsf{T}[\cdot]$).

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 programs 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 equivalent to $\bigwedge_{v \in V}(v_i \leftrightarrow v_j)$.

Roughly speaking, these three "operators" allow us to compare different subsets of atoms from a common set, V, under subset inclusion, proper-subset inclusion, and equality, respectively. The comparison takes place within a *single* interpretation while evaluating a formula. As an example, consider $V = \{v, w, u\}$ and an interpretation $I = \{v_1, v_2, w_2\}$, implicitly representing sets $X = \{v\}$ (via the relation $I|_{V_1} = \{v_1\}$) and $Y = \{v, w\}$ (via the relation $I|_{V_2} = \{v_2, w_2\}$). Then, we have that $(V_1 \leq V_2)$ as well as $(V_1 < V_2)$ are true under I which matches the observation that X is indeed a proper subset of Y, while $(V_1 = V_2)$ is false under I reflecting the fact that $X \neq Y$.

In accordance to this 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

$$P_{i,j} = \bigwedge_{r \in P} \big((B^+(r_i) \wedge B^-(r_j)) \rightarrow H(r_i)\big).$$

Formally, we have the following relation: 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 $I \models P_{i,j}$. Hence, we are able to characterise models of P (in case that $i = j$) as well as models of certain reducts of P (in case that $i \neq j$).

Having defined these building blocks, we proceed with the first encoding.

Definition 2 *Let P, Q be programs over V, let $A, B \subseteq V$, and let $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ be an inclusion problem. Then,*

$$\mathsf{S}[\Pi] := \neg\exists V_1 \Big(P_{1,1} \wedge \mathsf{S}^1(P, A) \wedge \forall V_3 \big(\mathsf{S}^2(Q, A, B) \rightarrow \mathsf{S}^3(P, Q, A)\big)\Big),$$

where

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

In fact, the scope, Φ, of $\exists V_1$ encodes the conditions for deciding whether a so-called *partial spoiler* (Eiter, Tompits, & Woltran 2005) for the inclusion problem Π exists. Such spoilers test certain relations on the reducts of the two programs involved, in order to avoid an explicit enumeration of all $R \in \mathcal{P}_A$ for deciding whether Π holds. Such a spoiler for Π exists iff Π does *not* hold. Hence, the resulting encoding Φ is unsatisfiable iff Π holds, and thus the closed QBF $\mathsf{S}[\Pi] = \neg\exists V_1 \Phi$ is valid iff Π holds.

In more concrete terms, given a correspondence problem Π and its encoding $\mathsf{S}[\Pi] = \neg\exists V_1 \Phi$, the general task of the QBF Φ is to test, for an answer-set candidate X of P, that no Y with $Y|_B = X|_B$ becomes an answer set of Q under some implicitly considered extension (in fact, it is sufficient to check only potential candidates Y of the form $Y|_{A \cup B} = X|_{A \cup B}$). Now, the subformula $P_{1,1} \wedge \mathsf{S}^1(P, A)$ tests whether X is such a candidate for P, with X being represented by V_1. In the remaining part of the encoding, $\mathsf{S}^2(Q, A, B)$ returns as its models those potential candidates Y (represented by V_3) for being answer set of Q. These candidates are now checked to be non-minimal and whether there is a further model (represented by V_4) of the reduct of Q with respect to Y surviving an extension of Q, for which X turns into an answer set of the extension of P.

In what follows, we review 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 $\mathsf{S}^2(Q, A, B)$, where we have $(A \cup B)_3 = (A \cup B)_1$. That is, in $\mathsf{S}^2(Q, A, B)$, we implicitly ignore all assignments to V_3 where atoms from A or B have different truth values as the corresponding assignments to 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 3 *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,*

$$\mathsf{T}[\Pi] := \neg\exists V_1 \Big(P_{1,1} \wedge \mathsf{T}^1(P, A, \mathcal{V}) \wedge \forall V_3^{A \cup B} \big(Q^{\mathcal{V}}_{3,3,1} \rightarrow \mathsf{T}^3(P, Q, A, \mathcal{V}) \big) \Big),$$

where

$$\mathsf{T}^1(P, A, \mathcal{V}) := \forall V_2^A \big((V_2^A < V_1^A) \rightarrow \neg P^{\mathcal{V}}_{2,1,1} \big) \text{ and}$$

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

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

$$\big(\big((A \cup B)_4 \leq (A \cup B)_1 \big) \wedge (V_4^{A \cup B} \leq V_3^{A \cup B}) \big) \wedge \neg \big(\big((A \cup B)_1 \leq (A \cup B)_4 \big) \wedge (V_3^{A \cup B} \leq V_4^{A \cup B}) \big).$$

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

Proposition 3 (Tompits & Woltran 2005) *For any inclusion problem Π, the following statements are equivalent: (i) Π holds; (ii) $\mathsf{S}[\Pi]$ is valid; and (iii) $\mathsf{T}[\Pi]$ is valid.*

In what follows, let, for every equivalence problem $\Pi = (P, Q, \mathcal{P}_A, =_B)$, Π' and Π'' denote the associated inclusion problems $(P, Q, \mathcal{P}_A, \subseteq_B)$ and $(Q, P, \mathcal{P}_A, \subseteq_B)$, respectively.

Corollary 1 *For any equivalence problem Π, the following statements are equivalent: (i) Π holds; (ii) $\mathsf{S}[\Pi'] \wedge \mathsf{S}[\Pi'']$ is valid; and (iii) $\mathsf{T}[\Pi'] \wedge \mathsf{T}[\Pi'']$ is valid.*

Special Cases

We now analyse how our encodings behave in certain instances of the equivalence framework which are located at lower levels of the polynomial hierarchy (cf. Proposition 1). We point out that the following simplifications are correspondingly implemented within our system.

In the case of *strong equivalence* (Lifschitz, Pearce, & Valverde 2001), i.e., problems of form $\Pi = (P, Q, \mathcal{P}_A, =_A)$ with $A = \mathcal{U}$, the encodings $\mathsf{T}[\Pi']$ and $\mathsf{T}[\Pi'']$ can be drastically simplified, since $V_2^A = V_3^A = V_5^A = \emptyset$. In particular, $\mathsf{T}[\Pi']$ is equivalent to

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

Now, the composed encoding for strong equivalence, i.e., the QBF $\mathsf{T}[\Pi'] \wedge \mathsf{T}[\Pi'']$, amounts to a single propositional unsatisfiability test, witnessing the coNP-completeness complexity for checking strong equivalence (Pearce, Tompits, & Woltran 2001; Lin 2002). This holds also for problems of the form $(P, Q, \mathcal{P}_\mathcal{U}, =_B)$ with arbitrary B. One can show that similar reductions (Pearce, Tompits, & Woltran 2001; Lin 2002) for testing strong equivalence in terms of propositional logic are simple variants thereof. Indeed, the methodology of the tool SELP (Chen, Lin, & Li 2005) is basically mirrored in our approach, in case the parameterisation of the given problem corresponds to a test for strong equivalence.

Strong equivalence *relative* to a set A of atoms (Woltran 2004), i.e., problems of form $(P, Q, \mathcal{P}_A, =_B)$ with $B = \mathcal{U}$, also yields simplifications within $\mathsf{T}[\Pi']$ and $\mathsf{T}[\Pi'']$, since $V_3^{A \cup B} = \emptyset$. In fact, $\mathsf{T}[\Pi']$ can be rewritten to

$$\neg\exists V_1 \big(P_{1,1} \wedge \forall V_2^A \big((V_2^A < V_1^A) \rightarrow \neg P^{\mathcal{V}}_{2,1,1} \big) \wedge \big(Q_{1,1} \rightarrow \exists V_4 \big(\big(V_4 < V_1 \big) \wedge Q_{4,1} \wedge \big((A_4 < A_1) \rightarrow \forall V_5^A ((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4}) \big) \big) \big) \big).$$

When putting this QBF into prenex normal form (see below), it turns out that the resulting QBF amounts to a $(2, \forall)$-QBF, again reflecting the complexity of the encoded task. Notice that for equivalence problems $(P, Q, \mathcal{P}_A, =_B)$ with $A \cup B = \mathcal{U}$, we also have that $V_3^{A \cup B} = \emptyset$. Thus, the same simplifications also apply for this special case.

The case of ordinary equivalence, i.e., considering problems of form $\Pi = (P, Q, \mathcal{P}_A, =)$ with $A = \emptyset$, is, indeed, a special case of relativised strong equivalence. As an additional optimisation we can drop the subformula

$$(A_4 < A_1) \rightarrow \forall V_5^A \big((V_5^A \leq V_1^A) \rightarrow \neg P^{\mathcal{V}}_{5,1,4} \big) \qquad (3)$$

from part T^3 of $\mathsf{T}[\Pi']$. This is because $A = \emptyset$, and therefore

$$(A_4 < A_1) := \bigwedge_{a \in A} (a_4 \rightarrow a_1) \wedge \neg \bigwedge_{a \in A} (a_1 \rightarrow a_4)$$

reduces to $\top \wedge \neg\top$, and thus to $\bot$. Hence, the validity of the implication (3) follows. However, this does not affect the number of quantifier alternations compared to the case of relativised strong equivalence. Indeed, this is in accord with the Π_2^P-completeness for ordinary equivalence. Putting things together, and observing that for $A = \emptyset$ we have $V_2^A = V_2$, the encoding $\mathsf{T}[\Pi']$ results for ordinary equivalence in

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

This encoding is related to encodings for computing answer sets via QBFs, as discussed by Egly *et al.* (2000). Indeed, taking the two main conjuncts from $\mathsf{T}[\Pi']$, viz.

$$P_{1,1} \wedge \forall V_2 ((V_2 < V_1) \rightarrow \neg P_{2,1}) \text{ and} \qquad (4)$$

$$Q_{1,1} \rightarrow \exists V_4 ((V_4 < V_1) \wedge Q_{4,1}), \qquad (5)$$

we get, for any assignment $Y_1 \subseteq V_1$, that Y_1 is a model of (4) iff Y is an answer set of P, and Y_1 is a model of (5) only if Y is not an answer set of Q.

Finally, we discuss the case of ordinary equivalence with projection, i.e., problems of form $(P, Q, \mathcal{P}_A, =_B)$ with $A = \emptyset$. Problems of this form are Π_3^P-complete, and thus we expect our system (after transformation to prenex form) to

yield $(3,\forall)$-QBFs. Here, the only simplification is to get rid off the subformula (3). We can do this for the same reason, viz. since $A = \emptyset$, as above. The simplifications are then as follows (once again using the fact that $V_2^A = V_2$ as well as $V_3^{A\cup B} = V_3^B$):

$$\neg\exists V_1\big(P_{1,1} \wedge \forall V_2\big((V_2 < V_1) \rightarrow \neg P_{2,1}\big) \wedge \forall V_3^B\big(Q^{\mathcal{V}}_{3,3,1} \rightarrow \exists V_4\big(\big(V_4 < (B_1 \cup V_3^B)\big) \wedge Q^{\mathcal{V}}_{4,3,1}\big)\big)\big).$$

Compared to the case of relativised equivalence, as discussed above, this time we have $V_3^{A\cup B} \neq \emptyset$ and thus an additional quantifier alternation "survives" the simplification. After bringing the encoding into its prenex form, we therefore get $(3,\forall)$-QBFs, once again reflecting the intrinsic complexity of the encoded problem.

For the encoding $\mathsf{T}[\cdot]$, the structure of the resulting QBF always reflects the complexity of the correspondence problem according to Proposition 1. This does not hold for formulas stemming from $\mathsf{S}[\cdot]$, however. In any case, our tool implements both encodings in order to provide interesting benchmarks for QBF solvers with respect to their capability to find implicit optimisations for equivalent QBFs.

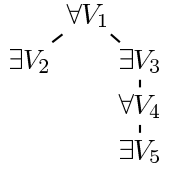
Transformations into Normal Forms

Most available QBF solvers require its input formula to be in a certain normal form, viz. in prenex conjunctive normal form (PCNF). Hence, in order to employ these solvers for our tool, the translations described above have to be transformed by a further two-phased normalisation step:

1. translation of the QBF into prenex normal form (PNF);
2. translation of the propositional part of the formula in PNF into CNF.

Both steps require to address different design issues. In what follows, we describe the fundamental problems, and then briefly provide our solutions in some detail.

First, the step of prenexing is not deterministic. As shown by Egly *et al.* (2004), there are numerous so-called *prenexing strategies*. The concrete selection of such a strategy (also depending on the concrete solver used) crucially influences the running times (see also our results below). In prenexing a QBF, certain *dependencies* between quantifiers have to be respected, when combining the quantifiers of different subformulas to one linear prefix. For our encodings, these dependencies are rather simple and analogous for both encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$. First, observe, however, that both encodings have a negation as their main connective which has to be shifted into the formula by applying the usual equivalence preserving transformations as known from first-order logic. In what follows, we implicitly assume that this step has already been performed. This allows us to consider the quantifier dependencies cleansed with respect to their polarities. The dependencies for the encoding $\mathsf{S}[\cdot]$ can then be illustrated as follows:



Here, the left branch results from the subformula S^1 and the right one results from the subformula $\forall V_3(\mathsf{S}^2(Q,A,B) \rightarrow \mathsf{S}^3(P,Q,A))$.

Inspecting these quantifier dependencies, we can group $\exists V_2$ either together with $\exists V_3$ or with $\exists V_5$. This yields the following two basic ways for prenexing our encodings:

$$\uparrow\colon\ \forall V_1\exists(V_2 \cup V_3)\forall V_4\exists V_5;\ \text{and}\ \downarrow\colon\ \forall V_1\exists V_3\forall V_4\exists(V_5 \cup V_2).$$

Together with the two encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, we thus get four different alternatives to represent an inclusion problem in terms of a prenex QBF; we will denote them by $\mathsf{S}_\uparrow[\cdot]$, $\mathsf{S}_\downarrow[\cdot]$, $\mathsf{T}_\uparrow[\cdot]$, and $\mathsf{T}_\downarrow[\cdot]$, respectively. Our experiments below show their different performance behaviour (relative to the employed QBF solver and the benchmarks).

Concerning the transformation of the propositional part of a prenex QBF into CNF, we use a method following Tseitin (1968) in which new atoms are introduced abbreviating subformula occurrences and which has the property that the resultant CNFs are always polynomial in the size of the input formula. Recall that a standard translation of a propositional formula into CNF based on distributivity laws yields formulas of exponential size in the worst case. However, the normal form translation into CNF using labels is not validity preserving like the one based on distributivity laws but only *satisfiability equivalent*. In the case of closed QBFs, the following result holds:

Proposition 4 *Let $\Phi = \mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \phi$, for $\mathsf{Q}_i \in \{\exists,\forall\}$ and $n > 0$, be either an $(n,\forall)$-QBF with n being even or an $(n,\exists)$-QBF with n being odd. Furthermore let ϕ' be the CNF resulting from the propositional part ϕ of Φ by introducing new labels following Tseitin (1968). Then, Φ and $\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \exists V \phi'$ are logically equivalent, where V are the new labels introduced by the CNF transformation.*

Note that for Φ as in the above proposition, we have that $\mathsf{Q}_1 = \exists$. Hence, in this case, $\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \exists V \phi'$ is the desired PCNF, equivalent to Φ, used as input for QBF solvers requiring PCNF format for evaluating Φ. In order to transform a QBF $\Psi = \mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \psi$ which is an $(n,\forall)$-QBF with n being odd or an $(n,\exists)$-QBF with n being even, we just apply the above proposition to $\overline{\mathsf{Q}}_n P_n \ldots \overline{\mathsf{Q}}_1 P_1 \neg\psi$, where $\overline{\mathsf{Q}}_i = \exists$ if $\mathsf{Q}_i = \forall$ and $\overline{\mathsf{Q}}_i = \forall$ otherwise, which is equivalent to $\neg\Psi$. That is, in order to evaluate Ψ by means of a QBF solver requiring PCNF input, we compute $\overline{\mathsf{Q}}_n P_n \ldots \overline{\mathsf{Q}}_1 P_1 \neg\psi$ and "reverse" the output. This is accommodated in cc$\top$ that either the original correspondence problem or the complementary problem is encoded whenever an input yields a QBF like Ψ.

For the entire normal-form transformation, one can use the quantifier-shifting tool `qst` (Zolda 2004). It accepts arbitrary QBFs in `boole` format (see below) as input and returns an equivalent PCNF QBF in *qdimacs* format, which is nowadays a de-facto standard for PCNF-QBF solvers. The tool `qst` implements 14 different strategies (among them $\uparrow$ and $\downarrow$ we use here) to obtain a PCNF and uses the mentioned structure-preserving normal-form transformation for the transformation to CNF.

The Implemented Tool

The system cc$\top$ implements the reductions from inclusion problems $(P, Q, \mathcal{P}_A, \subseteq_B)$ and equivalence problems $(P, Q, \mathcal{P}_A, =_B)$ to corresponding QBFs, together with the potential simplifications discussed above. It takes as input two programs, P and Q, and two sets of atoms, A and B, where A specifies the alphabet of the context and B the set of atoms for projection on the correspondence relation. The reduction ($\mathsf{S}[\cdot]$ or $\mathsf{T}[\cdot]$) and the type of correspondence problem ($\subseteq_B$ or $=_B$) are specified via command-line arguments: `-S`, `-T` to select the kind of reduction; and `-i`, `-e` to check for inclusion or equivalence between the two programs.

In general, the syntax to specify the programs in cc$\top$ corresponds to the basic `DLV` syntax.[1] Propositional `DLV` programs can be passed to cc$\top$ and programs processible for cc$\top$ can be handled by `DLV`. Considering the example from the introduction, the two programs would be expressed as:

P:
```
sel(b) :- b, not out(b).
sel(a) :- a, not out(a).
out(a) v out(b) :- a, b.
```

Q:
```
fail :- sel(a), not a, not fail.
fail :- sel(b), not b, not fail.
sel(a) v sel(b) :- a.
sel(a) v sel(b) :- b.
```

We suppose that file `P.dl` contains the code for program P and, accordingly, file `Q.dl` contains the code for Q. If we want to check whether P is equivalent to Q with respect to the projection to the output predicate $sel(\cdot)$, and restricting the context to programs over $\{a, b\}$, then we need to specify

- the context set, stored in a file, say `A`, containing the string "`(a, b)`", and
- the projection set, also stored in a file, say `B`, containing the string "`(sel(a), sel(b))`".

The invocation syntax for cc$\top$ is as follows:

```
ccT -e P.dl Q.dl A B.
```

By default, the encoding $\mathsf{T}[\cdot]$ is chosen. Note that the order of the arguments is important: first, the programs P and Q appear, then the context set A, and at last the projection set B. An alternative call of cc$\top$ for our example would be

```
cc⊤ -e -A "(a,b)" -B "(sel(a),sel(b))"
P.dl Q.dl
```

specifying sets A and B directly from the command line. After invocation, the resulting QBF is written to the standard output device and can be processed further by QBF solvers. The output can be piped, e.g., directly to the BDD-based QBF solver `boole`[2] by means of the command

```
ccT -e P.dl Q.dl A B | boole
```

which yields `0` or `1` as answer for the correspondence problem (in our case, the correspondence holds and the output is `1`). To employ further QBF solvers, the output has to be processed according to their input syntax.

If the set A (resp., B) is omitted in invocation, then each variable occurring in P or Q is assumed to be in A (resp., B); if "`0`" is passed instead of a filename, then the empty set is assumed for set A (resp., B). Thus, checking for strong equivalence between P and Q is done by

```
ccT -e P.dl Q.dl | boole
```

while ordinary equivalence (with projection over B) by

```
ccT -e P.dl Q.dl 0 B | boole.
```

We developed cc$\top$ entirely in *ANSI C*; hence, it is highly portable. The parser for the input data was written using *LEX* and *YACC*. The complete package in its current version consists of more than 2000 lines of code. For further information about cc$\top$ and the benchmarks below, see

```
http://www.kr.tuwien.ac.at/research/eq/.
```

Experimental Results

Our experiments were conducted to determine the behaviour of different QBF solvers in combination with the encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$ for inclusion checking, or, if the employed QBF solver requires the input in prenex form, with $\mathsf{S}_{\uparrow}[\cdot]$, $\mathsf{S}_{\downarrow}[\cdot]$, $\mathsf{T}_{\uparrow}[\cdot]$, and $\mathsf{T}_{\downarrow}[\cdot]$. To this end, we implemented a generator of inclusion problems which emanate from the proof of the Π_4^P-hardness of inclusion checking (Eiter, Tompits, & Woltran 2005), and thus provides us with benchmark problems capturing the intrinsic complexity of this task.

The strategy to generate such instances is as follows:

1. generate a $(4, \forall)$-QBF Φ in PCNF by random;
2. reduce Φ to an inclusion problem $\Pi = (P, Q, \mathcal{P}_A, \subseteq_B)$ such that Π holds iff Φ is valid;
3. apply cc$\top$ to derive the corresponding encoding Ψ for Π.

Incidentally, this procedure also yields a simple method for verifying the correctness of the overall implementation by simply checking whether Ψ is equivalent to Φ. We use here a parameterisation for the generation of random QBFs such that the benchmark set yields a nearly 50% distribution between the true and false instances. Therefore, the set is neither over- nor underconstrained and thus presumably located in a hard region, having easy-hard-easy patterns in mind.

The reduction from the generated QBF Φ to the corresponding inclusion problem is obtained as follows: Consider Φ of form $\forall W \exists X \forall Y \exists Z \phi$, where $\phi = \bigwedge_{i=1}^{n} C_i$ is a formula in CNF over atoms $V = (W \cup X \cup Y \cup Z)$ with $C_i = c_{i,1} \vee \cdots \vee c_{i,k_i}$. Now, let $\bar{V} = \{\bar{v} \mid v \in V\}$ be a set of new atoms, and define $C_i^* = c_{i,1}^*, \ldots, c_{i,k_i}^*$, $v^* = \bar{v}$, and $(\neg v)^* = v$. We generate

$$
\begin{aligned}
P = \{ & v \vee \bar{v} \leftarrow \mid v \in V\} \cup \\
& \{v \leftarrow u, \bar{u};\ \bar{v} \leftarrow u, \bar{u} \mid v, u \in V \backslash W\} \cup \\
& \{\leftarrow not\, v;\ \leftarrow not\, \bar{v} \mid v \in V \backslash W\} \cup \\
& \{v \leftarrow C_i^*;\ \bar{v} \leftarrow C_i^* \mid v \in V \backslash W;\ 1 \leq i \leq n\}.
\end{aligned}
$$

For program Q we use further atoms $X' = \{x' \mid x \in X\}$, $\bar{X}' = \{\bar{x}' \mid x \in X\}$ and generate:

[1] See `http://www.dlvsystem.com/` for more information about `DLV`.

[2] This solver is available at `http://www.cs.cmu.edu/~modelcheck/bdd.html`.

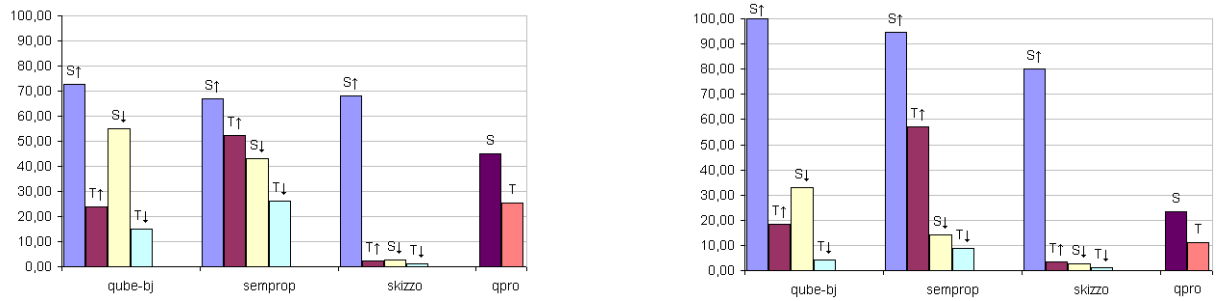


Figure 1: Results for true (left chart) and false (right chart) problem instances subdivided by solvers and encodings.

$$
\begin{aligned}
Q = \{&v \vee \bar{v} \leftarrow \mid v \in X \cup Y\} \cup \\
&\{v \leftarrow u, \bar{u};\ \bar{v} \leftarrow u, \bar{u} \mid v, u \in X \cup Y\} \cup \\
&\{\leftarrow x', \bar{x}';\ \leftarrow not\, x', not\, \bar{x}' \mid x \in X\} \cup \\
&\{v \leftarrow x';\ \bar{v} \leftarrow x';\\
&v \leftarrow \bar{x}';\ \bar{v} \leftarrow \bar{x}' \mid v \in X \cup Y, x \in X\} \cup \\
&\{x' \leftarrow \bar{x}, not\, \bar{x}';\ \bar{x}' \leftarrow x, not\, x' \mid x \in X\}.
\end{aligned}
$$

Finally, sets A and B are defined as:

$$A = B = \{X \cup \bar{X} \cup Y \cup \bar{Y}\}.$$

It can be shown that Φ is valid iff $(P, Q, \mathcal{P}_A, \subseteq_B)$ holds.

We have set up a test series comprising 1000 instances of inclusion problems generated that way (465 of them evaluating to true), where the first program P has 620 rules, the second program Q has 280 rules, using a total of 40 atoms, and the sets A and B of atoms are chosen to contain 16 atoms. After employing cc$\top$, the resulting QBFs possess, in case of translation $\mathsf{S}[\cdot]$, 200 atoms and, in case of translation $\mathsf{T}[\cdot]$, 152 atoms. The additional prenexing step (together with the translation of the propositional part into CNF) yields, in case of $\mathsf{S}[\cdot]$, QBFs with 6575 clauses over 2851 atoms and, in case of $\mathsf{T}[\cdot]$, QBFs with 6216 clauses over 2555 atoms.

We compared four different state-of-the-art QBF solvers, namely `qube-bj` (Giunchiglia, Narizzano, & Tacchella 2003), `semprop` (Letz 2002), `skizzo` (Benedetti 2005), and `qpro` (Egly, Seidl, & Woltran 2006). The former three require QBFs in PCNF as input (thus, we tested them using encodings $\mathsf{S}_{\uparrow}[\cdot]$, $\mathsf{S}_{\downarrow}[\cdot]$, $\mathsf{T}_{\uparrow}[\cdot]$, and $\mathsf{T}_{\downarrow}[\cdot]$), while `qpro` admits arbitrary QBFs as input (we tested it with the non-prenex encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$). Our results are depicted in Figure 1. The y-axis shows the (arithmetically) average running time in seconds (time-out was 100 seconds) for each solver (with respect to the chosen translation and prenexing strategy).

As expected, for all solvers, the more compact encodings of form $\mathsf{T}[\cdot]$ were evaluated faster than the QBFs stemming from encodings of form $\mathsf{S}[\cdot]$. The performance of the prenex-form solvers `qube-bj`, `semprop`, and `skizzo` is highly dependent on the prenexing strategy, and $\downarrow$ dominates $\uparrow$.

For the special case of ordinary equivalence, we compared our approach against the system DLPEQ (Oikarinen & Janhunen 2004) which is based on a reduction to disjunctive logic programs, using `gnt` (Janhunen *et al.* 2006) as answer-set solver. The benchmarks rely on randomly generated $(2, \exists)$-QBFs using Model A (Gent & Walsh 1999). Each QBF is reduced to a program following Eiter & Gottlob (1995), such that the latter possesses an answer set iff the original QBF is valid. The idea of the benchmarks is to compare each such program with one in which one randomly selected rule is dropped, simulating a "sloppy" programmer, in terms of ordinary equivalence.

Average running times are shown in Table 1. The number n of variables in the original QBF varies from 10 to 24, and, for each n, 100 such program comparisons are generated for which the portion of cases where equivalence holds is between 40% and 50% (for details about the benchmarks, cf. Oikarinen & Janhunen (2004)). We set a time-out of 120 seconds, and both the one-phased mode (DLPEQ1) as well as the two-phased mode (DLPEQ2) of DLPEQ were tested. For cc$\top$, we compared the same back-end solvers as above, using encoding $\mathsf{T}[\cdot]$. Recall that for ordinary equivalence cc$\top$ provides $(2, \forall)$-QBFs, thus we can resign on the distinction between prenexing strategies. The dedicated DLPEQ approach turns out to be faster, but, interestingly, among the tested QBF solvers, `qpro` is the most competitive one, while the PCNF-QBF solvers perform bad even for small instances. This result is encouraging as regards further development of the non-normal form approach of QBF solvers.

Conclusion

In this paper, we discussed an implementation for advanced program comparison in answer-set programming via encodings into quantified propositional logic. This approach was motivated by the high computational complexity we have to face for correspondence checking, making a direct realisation via ASP hard to accomplish. Since currently practicably efficient solvers for quantified propositional logic are available, they can be employed as back-end inference engines to verify the correspondence problems under consideration using the proposed encodings. Moreover, since such problems are one of the few natural ones 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 (cf., Le Berre *et al.* (2005)).

References

Arieli, O., and Denecker, M. 2003. Reducing Preferential Paraconsistent Reasoning to Classical Entailment. *Journal*

	qube-bj	semprop	skizzo	qpro	DLPEQ1	DLPEQ2
10	120.00	120.00	14.71	0.05	0.05	0.04
12	120.00	120.00	18.45	0.17	0.06	0.06
14	120.00	120.00	48.70	0.51	0.09	0.08
16	120.00	120.00	120.00	1.54	0.13	0.11
18	120.00	120.00	120.00	4.85	0.19	0.15
20	120.00	120.00	120.00	15.07	0.31	0.25
22	120.00	120.00	120.00	46.23	0.50	0.39
24	120.00	120.00	120.00	120.00	0.84	0.64

	qube-bj	semprop	skizzo	qpro	DLPEQ1	DLPEQ2
10	0.29	56.00	12.27	0.01	0.03	0.03
12	1.49	65.06	18.24	0.02	0.05	0.03
14	5.35	69.35	33.17	0.07	0.05	0.04
16	25.48	86.53	120.00	0.23	0.07	0.06
18	46.10	65.74	120.00	0.50	0.09	0.07
20	82.06	90.34	120.00	1.95	0.20	0.15
22	76.77	86.95	120.00	6.11	0.20	0.15
24	83.68	92.43	120.00	14.81	0.40	0.34

Table 1: Comparing cc$\top$ against DLPEQ on true (left table) and false (right table) problem instances subdivided by solvers.

of Logic and Computation 13(4):557–580.

Benedetti, M. 2005. sKizzo: A Suite to Evaluate and Certify QBFs. In *Proc. CADE'05*, volume 3632 of *LNCS*, 369–376. Springer.

Besnard, P.; Schaub, T.; Tompits, H.; and Woltran, S. 2005. Representing Paraconsistent Reasoning via Quantified Propositional Logic. In *Inconsistency Tolerance*, volume 3300 of *LNCS*, 84–118. Springer.

Chen, Y.; Lin F.; and Li, L. 2005. SELP - A System for Studying Strong Equivalence Between Logic Programs. In *Proc. LPNMR'05*, volume 3552 of *LNAI*, 442–446. Springer.

Delgrande, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2004. On Computing Solutions to Belief Change Scenarios. *Journal of Logic and Computation* 14(6):801–826.

Egly, U.; Eiter, T.; Tompits, H.; and Woltran, S. 2000. Solving Advanced Reasoning Tasks using Quantified Boolean Formulas. In *Proc. AAAI'00*, 417–422. AAAI Press.

Egly, U.; Seidl, M.; Tompits, H.; Woltran, S.; and Zolda, M. 2004. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proc. SAT'03. Selected Revised Papers*, volume 2919 of *LNCS*, 214–228. Springer.

Egly, U.; Seidl, M.; and Woltran, S. 2006. A Solver for QBFs in Nonprenex Form. In *Proc. ECAI'06*.

Eiter, T., and Fink, M. 2003. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proc. ICLP'03*, volume 2916 of *LNCS*, 224–238. Springer.

Eiter, T., and Gottlob, G. 1995. On the Computational Cost of Disjunctive Logic Programming: Propositional Case. *Annals of Mathematics and Artificial Intelligence* 15(3/4):289–323.

Eiter, T.; Tompits, H.; and Woltran, S. 2005. On Solution Correspondences in Answer Set Programming. In *Proc. IJCAI'05*, 97–102.

Gelfond, M., and Leone, N. 2002. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence* 138(1-2):3–38.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing* 9:365–385.

Gent, I., and Walsh, T. 1999. Beyond NP: The QSAT Phase Transition. In *Proc. AAAI'99*, 648–653. AAAI Press.

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2003. Backjumping for Quantified Boolean Logic Satisfiability. *Artificial Intelligence* 145:99–120.

Janhunen, T.; Niemelä, I.; Seipel, D.; and Simons, P. 2006. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM Transactions on Computational Logic* 7(1):1–37.

Le Berre, D.; Narizzano, M.; Simon, L.; and Tacchella, A. 2005. The Second QBF Solvers Comparative Evaluation. In *Proc. SAT'04. Revised Selected Papers*, volume 3542 of *LNCS*, 376–392. Springer.

Letz, R. 2002. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proc. TABLEAUX'02*, volume 2381 of *LNCS*, 160–175. Springer.

Lifschitz, V.; Pearce, D.; and Valverde, A. 2001. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic* 2(4):526–541.

Lin, F. 2002. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic. In *Proc. KR'02*, 170–176. Morgan Kaufmann.

Oikarinen, E.; and Janhunen, T. 2004. Verifying the Equivalence of Logic Programs in the Disjunctive Case. *Proc. LPNMR'04*, volume 2923 of *LNCS*, 180–193. Springer.

Pearce, D.; Tompits, H.; and Woltran, S. 2001. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions. In *Proc. EPIA'01*, volume 2258 of *LNCS*, 306–320. Springer.

Rintanen, J. 1999. Constructing Conditional Plans by a Theorem Prover. *JAIR* 10:323–352.

Tompits, H., and Woltran, S. 2005. Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In *Proc. ICLP'05*, volume 3668 of *LNCS*, 189–203. Springer.

Tseitin, G. S. 1968. On the Complexity of Derivation in Propositional Calculus. *Studies in Constructive Mathematics and Mathematical Logic, Part II*. 234–259.

Woltran, S. 2004. Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In *Proc. JELIA'04*, volume 3229 of *LNCS*, 161–173. Springer.

Zolda, M. 2004. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. Master's Thesis, Vienna University of Technology.