An Extension of the System $\mathsf{cc}\top$ for Testing Relativised Uniform Equivalence under Answer-Set Projection[*]

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

Stefan Woltran
Institut für Informationssysteme 184/2,
Technische Universität Wien,
Favoritenstraße 9-11,
A-1040 Vienna, Austria
woltran@dbai.tuwien.ac.at

Abstract

The system $cc\top$ is a tool for testing correspondence between nonmonotonic logic programs under the answer-set semantics with respect to different refined notions of program correspondence. The basic architecture of $cc\top$ is to reduce a given correspondence problem into the satisfiability problem for quantified propositional logic and to employ off-the-shelf solvers for the latter language as back-end inference engines. In a previous incarnation of $cc\top$, the system was designed to test correspondence between logic programs based on relativised strong equivalence under answer-set projection. *Such a setting generalises the usual notion of strong equivalence by taking the alphabet of the context programs as well as the projection of the compared answer sets to a set of designated output atoms into account. In this paper, we describe an extension of $cc\top$ for testing similarly parameterised correspondence problems but generalising* uniform equivalence*, which have recently been introduced in previous work. Besides reviewing the formal underpinnings of the new component of $cc\top$, we discuss an alternative encoding as well as optimisations for special problem classes. Furthermore, we give a preliminary performance evaluation of the new component.*

1. Introduction

In this paper, we deal with a system for testing various refined notions of program correspondence for nonmonotonic logic programs under the answer-set semantics. The latter formalism has been proven useful in a variety of domains including planning, diagnosis, information integration, and Semantic-Web reasoning, and represents the canonical instance of the general *answer-set programming* (ASP) *paradigm*, an important approach for declarative problem solving.

The system discussed here, called $cc\top$ (standing for "correspondence-checking tool") [12], belongs to a current line of research in ASP about questions of program equivalence relevant for different software engineering tasks like modular programming and debugging. This research was for the most part initiated by the seminal work of Lifschitz, Pearce, and Valverde [11] about *strong equivalence*. Albeit the latter notion circumvents a shortcoming of ordinary equivalence between logic programs (viz., that ordinary equivalence does not yield a replacement property similar to the one of classical logic), it is however too restrictive for certain applications. This led to the investigation of more liberal notions, chiefly among them *uniform equivalence* [6]. In any case, both strong and uniform equivalence do not take standard programming techniques like the use

[*]The authors of this work were 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.

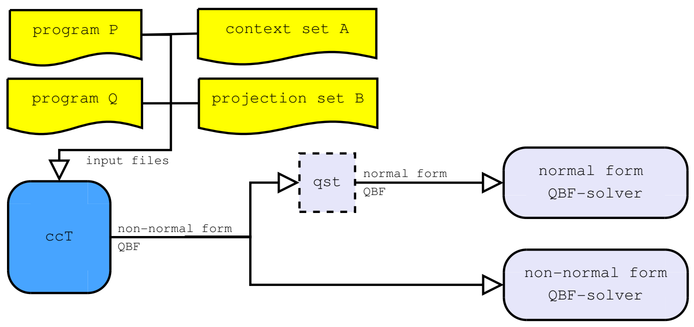

Figure 1. Overall architecture of $\mathtt{cc}\top$**.**

of local (auxiliary) variables into account, which may occur in some subprograms but which are ignored in the final computation. In other words, these notions do not admit the *projection* of answer sets to a set of dedicated output atoms.

To accommodate issues like the above, Eiter *et al.* [7] introduced a general framework for specifying parameterised notions of program correspondence, allowing both answer-set projection as well as the specification which kind of context class should be used for the program comparison. This framework thus generalises not only strong and uniform equivalence but also *relativised* versions thereof [18] (where "relativised" means that the alphabet of the context class is an additional parameter).

The system $\mathtt{cc}\top$ was developed as a checker for the type of correspondence problems which were the main focus of the analysis of Eiter *et al.* [7], viz. for correspondence problems generalising strong equivalence. The main approach of $\mathtt{cc}\top$ to verify these kinds of problems is to reduce them to the satisfiability problem of quantified propositional logic.[1] Such a reduction approach is motivated by two aspects: (i) the high complexity of the considered problems—lying on the fourth level of the polynomial hierarchy—makes it in general presumably infeasible to compute them by means of answer-set solvers, yet efficient encodings to quantified propositional logic are possible, and (ii) the existence of sophisticated solvers for quantified propositional logic.

In this paper, we discuss an extension of $\mathtt{cc}\top$ for checking further classes of correspondence problems from the framework of Eiter *et al.* [7], viz. problems generalising *uniform equivalence*. These kinds of problems were recently analysed in previous work [14] and are called *propositional query equivalence problems* (PQEPs) and *propositional query inclusion problems* (PQIPs) (the names stem from taking a database point of view in which programs are considered as queries over databases). Like for their strong pendants, checking PQEPs and PQIPs is computationally hard, lying on the third level of the polynomial hierarchy, therefore a similar reduction approach to QBFs is viable. Indeed, such reductions are described in previous work [14], and the new component of $\mathtt{cc}\top$ is based on these reductions, which we review in this paper.

The overall architecture of $\mathtt{cc}\top$ is depicted in Figure 1. It takes as input two programs, P and Q, as well as the context set A and the projection set B. This input is then transformed into a QBF which can then handed to QBF solvers. Validity of the resulting QBF reflects the outcome of the original problem, which holds when, for any set $R \subseteq A$ of facts, the answer sets of $P \cup R$ and $Q \cup R$ projected to B coincide (in case of a PQEP) or when the answer sets of $P \cup R$ projected to B are included in the answer sets of $Q \cup R$ projected to B (in case of a PQIP).[2] Since the QBFs generated by $\mathtt{cc}\top$ are not in a particular normal form, for solvers requiring normal-form QBFs, a corresponding normaliser, $\mathtt{qst}$ [19], is needed.

In complementing the reductions given by Oetsch *et al.* [14], we provide refined reductions for PQEPs and PQIPs which use less variables than the original ones. Furthermore, we discuss simplified transformations for special problem classes. All these transformations have been implemented in an extension of $\mathtt{cc}\top$. We round off our discussion by reporting about a preliminary experimental evaluation of the extension of $\mathtt{cc}\top$ using different state-of-the-art QBF solvers.

2. Preliminaries

Answer-set semantics. We are concerned with *propositional disjunctive logic programs* (DLPs) which are finite sets of rules of form

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

[1]Recall that quantified propositional logic is an extension of ordinary propositional logic allowing quantifications over atomic formulas. Following custom, we refer to formulas of quantified propositional logic as *quantified Boolean formulas* (QBFs).

[2]In the notions generalising strong equivalence, R would be a program over A.

where $n \geq m \geq l \geq 0$, all a_i are propositional atoms from some fixed universe $\mathcal{U}$, and "not" denotes *default negation*. Rules of form $a \leftarrow$ are *facts* and are usually written without the symbol "$\leftarrow$". We denote by $At(P)$ the set of all atoms occurring in a program P, and say that P is *over* A if $At(P) \subseteq A$. $\mathcal{P}_A$ refers to the set of all programs over A, and 2^A to the set of all facts over A. By an *interpretation* we understand a set of atoms, and, as usual, an interpretation is a *model* of a rule r iff it satisfies the head of r whenever it satisfies the body of r. The notion of a model extends to programs in the usual way and is denoted by $I \models P$. Following Gelfond and Lifschitz [8], 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 (i) deleting all rules containing a default negated atom $\mathrm{not}\ a$ such that $a \in I$, and (ii) deleting all default negated atoms in the remaining rules. The collection of all answer sets of a program P is denoted by $\mathcal{AS}(P)$.

Program correspondence. We use the following notations in the sequel: For an interpretation I and a set $\mathcal{S}$ of interpretations, $\mathcal{S}|_I$ is defined as $\{Y \cap I \mid Y \in \mathcal{S}\}$. For a singleton set $\mathcal{S} = \{Y\}$, we also write $Y|_I$ instead of $\mathcal{S}|_I$. Furthermore, for sets $\mathcal{S}, \mathcal{S}'$ of interpretations, an interpretation B, and $\odot \in \{\subseteq, =\}$, we define $\mathcal{S} \odot_B \mathcal{S}'$ as $\mathcal{S}|_B \odot \mathcal{S}'|_B$.

Some basic equivalence notions are defined as follows: Two programs P and Q are (i) *ordinarily equivalent* iff $AS(P) = AS(Q)$; (ii) *uniformly equivalent* [6] iff, for each set F of facts, $AS(P \cup F) = AS(Q \cup F)$; and (iii) *strongly equivalent* [11] iff, for each program R, $AS(P \cup R) = AS(Q \cup R)$.

In abstracting from these notions, Eiter *et al.* [7] introduced the notion of a *correspondence problem* which allows to specify (i) a *context*, i.e., a class of programs used to be added to the programs under consideration, and (ii) the relation that has to hold between the answer sets of the extended programs. The concrete formal realisation is as follows. A correspondence problem (*over* $\mathcal{U}$) is a quadruple $\Pi = (P, Q, \mathcal{C}, \rho)$, where $P, Q \in \mathcal{P}_{\mathcal{U}}$ are programs, $\mathcal{C} \subseteq \mathcal{P}_{\mathcal{U}}$ is a class of programs (the *context class* of Π), and $\rho \subseteq 2^{2^{\mathcal{U}}} \times 2^{2^{\mathcal{U}}}$ is a binary relation over sets of interpretations. Π is said to *hold* iff, for each program $R \in \mathcal{C}$, $(AS(P \cup R), AS(Q \cup R)) \in \rho$. By instantiating $\mathcal{C}$ and ρ, different equivalence notions from the literature can be expressed. In particular, P and Q are (i) strongly equivalent iff $(P, Q, \mathcal{P}_{\mathcal{U}}, =_{\mathcal{U}})$ holds, (ii) uniformly equivalent iff $(P, Q, 2^{\mathcal{U}}, =_{\mathcal{U}})$ holds, (iii) strongly equivalent relative to A [18], for $A \subseteq \mathcal{U}$, iff $(P, Q, \mathcal{P}_A, =_{\mathcal{U}})$ holds, and (iv) uniformly equivalent relative to A [18], for $A \subseteq \mathcal{U}$, iff $(P, Q, 2^A, =_{\mathcal{U}})$ holds.

Some important further kinds of correspondence problems that generalise the above ones are those of form $(P, Q, \mathcal{P}_A, \odot_B)$ and of form $(P, Q, 2^A, \odot_B)$, respectively, for $\odot \in \{\subseteq, =\}$, taking *projection* to a dedicated set B of output atoms into account. The former kinds of problems were analysed by Eiter *et al.* [7] while the latter ones by Oetsch *et al.* [14]. Here, we are interested in those latter kinds of problems and, like in previous work [14], we call problems of form $(P, Q, 2^A, \subseteq_B)$ *propositional query inclusion problems*, or *PQIPs* for short, and problems of form $(P, Q, 2^A, =_B)$ *propositional query equivalence problems* or *PQEPs*.

A pair (X, Y) with $X \subseteq A$ and $Y \subseteq \mathcal{U}$ is called a *counterexample*[3] *for a PQIP* $(P, Q, 2^A, \subseteq_B)$ iff $Y \in AS(P \cup X)$ and no Y' with $Y'|_B = Y|_B$ is contained in $AS(Q \cup X)$. Hence, a PQIP Π has a counterexample iff Π does not hold [14].

Example 1 *Consider* $P = \{a \vee b \leftarrow; a \leftarrow c\}$, $Q = \{a \leftarrow \mathrm{not}\ b; b \leftarrow \mathrm{not}\ a; c \leftarrow a\}$*, and* $B = \{a, b\}$*. Since* $AS(P) = \{\{a\}, \{b\}\}$ *and* $AS(Q) = \{\{a, c\}, \{b\}\}$*, we get* $AS(P)|_B = AS(Q)|_B = \{\{a\}, \{b\}\}$*. Now, for* $A = B$*,* $(P, Q, 2^A, \subseteq_B)$ *holds, while for* $A' = \{a, b, c\}$ *the PQIP* $\Pi = (P, Q, 2^{A'}, \subseteq_B)$ *does not hold. This is witnessed by* $(\{b, c\}, \{a, b, c\})$ *which is the unique counterexample* (*over* $\{a, b, c\}$) *for* Π. $\Diamond$

For a PQEP $\Pi = (P, Q, 2^A, =_B)$, the PQIPs $\Pi^{\rightarrow} = (P, Q, 2^A, \subseteq_B)$ and $\Pi^{\leftarrow} = (Q, P, 2^A, \subseteq_B)$ are called *associated* with Π. Obviously, a PQEP Π holds iff both $\Pi^{\rightarrow}$ and $\Pi^{\leftarrow}$ hold. We extend the definition of a counterexample to PQEPs and call a pair (X, Y) a counterexample for a PQEP Π if (X, Y) is a counterexample for $\Pi^{\rightarrow}$ or $\Pi^{\leftarrow}$.

Concerning complexity, as shown previously [14], given programs $P, Q \in \mathcal{P}_{\mathcal{U}}$, sets $A, B \subseteq \mathcal{U}$ of atoms, and $\odot \in \{\subseteq, =\}$, deciding whether $(P, Q, 2^A, \odot_B)$ holds is Π_3^P-complete. Moreover, the problem is Π_2^P-complete in case $B = \mathcal{U}$. Both hardness results hold even for arbitrary but fixed A.

Quantified propositional logic. The complexity results above show that PQIPs and PQEPs can be efficiently reduced to *quantified propositional logic*, an extension of classical propositional logic in which formulas are permitted to contain quantifications over propositional variables. Similar to predicate logic, $\exists$ and $\forall$ are used as symbols for existential and universal quantification, respectively. Such formulas are also called *quantified Boolean formulas* (QBFs); we denote them by upper-case Greek letters.

For an interpretation I and a QBF Φ, the relation $I \models \Phi$ is defined analogously as in classical propositional logic, with the additional conditions that $I \models \exists p\, \Psi$ iff $I \models \Psi[p/\top]$ or $I \models \Psi[p/\bot]$, and $I \models \forall p\, \Psi$ iff $I \models \Psi[p/\top]$ and $I \models \Psi[p/\bot]$, for $\Phi = \mathsf{Q} p\, \Psi$ with $\mathsf{Q} \in \{\exists, \forall\}$, where

[3]Note that in our previous work [14] we used "explicit counterexample" instead of "counterexample".

$\Psi[p/\phi]$ denotes the QBF resulting from Ψ by replacing each free occurrence of p in Ψ by ϕ (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). 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.

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 $P = \{p_1, \ldots, p_n\}$. A QBF Φ is said to be in *prenex normal form* (PNF) iff it is closed and of the form $\mathsf{Q}_nP_n \ldots \mathsf{Q}_1P_1\,\phi$, where $n \geq 0$, ϕ is a propositional formula, and $\mathsf{Q}_i \in \{\exists, \forall\}$ such that $\mathsf{Q}_i \neq \mathsf{Q}_{i+1}$ for $1 \leq i \leq n-1$. Moreover, if ϕ is in conjunctive normal form, then Φ is in *prenex conjunctive normal form* (PCNF), and if ϕ is in disjunctive normal form, then Φ is in *prenex disjunctive normal form* (PDNF). A QBF $\Phi = \mathsf{Q}_nP_n \ldots \mathsf{Q}_1P_1\,\phi$ 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 Φ corresponds to a quantifier occurrence in the prenex normal form. In general, there are different ways to obtain an equivalent prenex QBF (cf. Egly *et al.* [4] for more details on this issue). Well-known complexity results for the evaluation problem of QBFs [16] indicate that PQIPs and PQEPs can be efficiently reduced to $(3,\forall)$-QBFs. These reductions are the central theoretical basis for our system and are discussed in detail in the next section.

3. System specifics

We now discuss details of the new extension of $\mathtt{cc}\top$ for verifying PQIPs and PQEPs. The overall architecture of $\mathtt{cc}\top$ was already outlined in the introduction and depicted in Figure 1. Originally, $\mathtt{cc}\top$ was developed as an implementation for verifying correspondence problems of form $(P, Q, \mathcal{P}_A, \odot_B)$, for $\odot \in \{\subseteq, =\}$ [12]. The syntax to specify programs in $\mathtt{cc}\top$ corresponds to the basic `DLV` syntax.[4] Furthermore, the tool is entirely developed in *ANSI C*, using *LEX* and *YACC* for the parser, and publicly available (including the source code); it can be downloaded from the Web at

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

The section is organised as follows. First, we review the basic encodings for mapping PQIPs and PQEPs into QBFs, as developed in previous work [14]. Afterwards, we provide an alternative reduction and discuss its outcome on special instances of correspondence problems. Finally, we give some details on how to apply the system.

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

3.1. Translating query problems

In what follows, we make use of sets of globally new atoms in order to refer to different assignments of the same atoms within a single formula. More formally, given a 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 $\bigwedge_{v \in V}(v^i \leftrightarrow v^j)$.

These operators allow to compare different subsets of atoms from a common set V under subset inclusion, proper-subset inclusion, and equality, respectively. Formally, we have that, given $X, Y \subseteq V$, an interpretation I with $I|_{V^i} = X^i$ and $I|_{V^j} = Y^j$ is (i) a model of $V^i \leq V^j$ iff $X \subseteq Y$, (ii) a model of $V^i < V^j$ iff $X \subset Y$, and (iii) a model of $V^i = V^j$ iff $X = Y$.

We use superscripts 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 v in α by v^i, where α is any formula or rule. 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 $\top$.

Proposition 1 ([17]) *Let P be a program with $At(P) = V$, I an interpretation, and $X, Y \subseteq V$ such that, for some $i, j \geq 0$, $I|_{V^i} = X^i$ and $I|_{V^j} = Y^j$. Then, $X \models P^Y$ iff $I \models P^{\langle i,j \rangle}$, where*

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

Example 2 *Consider the program $Q = \{a \leftarrow \text{not } b;\ b \leftarrow \text{not } a\}$. Then, for instance, $Q^{\langle 1,2 \rangle}$ is given by $(\neg b^2 \rightarrow a^1) \wedge (\neg a^2 \rightarrow b^1)$, and we have that $\{a^2, b^2\} \cup X^1$ is a model of $Q^{\langle 1,2 \rangle}$, for each $X^1 \subseteq \{a^1, b^1\}$, reflecting the fact that any interpretation X is a model of the reduct $Q^{\{a,b\}}$.* $\Diamond$

With these building blocks at hand, we can state the following encoding, as introduced by Oetsch *et al.* [14].

Definition 1 *Let $\Pi = (P, Q, 2^A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, and $A, B \subseteq V$. Then,*

$$\mathcal{S}[\Pi] = \Phi_\Pi \wedge \forall V^4\big((B^4 = B^1) \rightarrow \Psi_\Pi\big), \text{ where}$$

$$\Phi_\Pi = P^{\langle 1,1 \rangle} \wedge (A^2 \leq A^1) \wedge \forall V^3\Big(\big((A^2 \leq A^3) \wedge (V^3 < V^1)\big) \rightarrow \neg P^{\langle 3,1 \rangle}\Big) \text{ and}$$

$$\Psi_\Pi = \Big(\big(Q^{\langle 4,4 \rangle} \wedge (A^2 \leq A^4)\big) \rightarrow \exists V^5\big(\big((A^2 \leq A^5) \wedge (V^5 < V^4)\big) \wedge Q^{\langle 5,4 \rangle}\big)\Big).$$

Table 1. Outcome of the different encodings of $\Pi = (P, Q, 2^A, \subseteq_B)$ from Example 1.

$\mathcal{S}[\Pi]$	$\mathcal{T}[\Pi]$
$\Phi_\Pi \wedge \forall a^4 b^4 c^4 \big((a^4 \leftrightarrow a^1) \wedge (b^4 \leftrightarrow b^1) \rightarrow$	$\Phi_\Pi \wedge \forall c^4$
$(((\neg b^4 \rightarrow a^4) \wedge (\neg a^4 \rightarrow b^4) \wedge (a^4 \rightarrow c^4) \wedge$	$(((\neg b^1 \rightarrow a^1) \wedge (\neg a^1 \rightarrow b^1) \wedge (a^1 \rightarrow c^4) \wedge$
$(a^2 \rightarrow a^4) \wedge (b^2 \rightarrow b^4)) \rightarrow \exists a^5 b^5 c^5$	$(a^2 \rightarrow a^1) \wedge (b^2 \rightarrow b^1)) \rightarrow \exists a^5 b^5 c^5$
$((a^2 \rightarrow a^5) \wedge (b^2 \rightarrow b^5) \wedge$	$((a^2 \rightarrow a^5) \wedge (b^2 \rightarrow b^5) \wedge$
$(a^5 \rightarrow a^4) \wedge (b^5 \rightarrow b^4) \wedge (c^5 \rightarrow c^4) \wedge$	$(a^5 \rightarrow a^1) \wedge (b^5 \rightarrow b^1) \wedge (c^5 \rightarrow c^4) \wedge$
$\neg((a^4 \rightarrow a^5) \wedge (b^4 \rightarrow b^5) \wedge (c^4 \rightarrow c^5)) \wedge$	$\neg((a^1 \rightarrow a^5) \wedge (b^1 \rightarrow b^5) \wedge (c^4 \rightarrow c^5)) \wedge$
$(\neg b^4 \rightarrow a^5) \wedge (\neg a^4 \rightarrow b^5) \wedge (a^5 \rightarrow c^5))))\big)$	$(\neg b^1 \rightarrow a^5) \wedge (\neg a^1 \rightarrow b^5) \wedge (a^5 \rightarrow c^5)))$

Observe that the free variables of $\mathcal{S}[\Pi]$ are given by $V^1 \cup A^2$. Assignments to $V^1 \cup A^2$ yield the counterexamples for Π, in case $\mathcal{S}[\Pi]$ is satisfied by those assignments.

Proposition 2 ([14]) *Let $\Pi = (P, Q, 2^A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, $A, B \subseteq V$, $X \subseteq A$, and $Y \subseteq V$. Then, (X, Y) is a counterexample for Π iff $Y^1 \cup X^2 \models \mathcal{S}[\Pi]$. Moreover, Π holds iff the closed QBF $\mathsf{S}[\Pi] = \forall V^1 \forall A^2 \neg \mathcal{S}[\Pi]$ is valid.*

The extension of the encodings to PQEPs is done by means of the associated PQIPs.

Proposition 3 ([14]) *Let $\Pi = (P, Q, 2^A, =_B)$ be a PQEP, $At(P \cup Q) = V$, $A, B \subseteq V$, $X \subseteq A$, and $Y \subseteq V$. Then, (X, Y) is a counterexample for Π iff $Y^1 \cup X^2 \models \mathcal{S}[\Pi^{\rightarrow}] \vee \mathcal{S}[\Pi^{\leftarrow}]$. Moreover, Π holds iff $\mathsf{S}[\Pi] = \forall V^1 \forall A^2 (\neg \mathcal{S}[\Pi^{\rightarrow}] \wedge \neg \mathcal{S}[\Pi^{\leftarrow}])$ is valid.*

3.2. An alternative encoding and special cases

We now introduce an adaption of the above encodings. The benefit of the refined encodings is that the number of universally quantified variables is reduced—in fact, in some specific cases, one quantifier block even vanishes. This guarantees adequacy (in the sense of Besnard *et al.* [1]) also for special cases of query problems without projection.

The key observation for the subsequent adaption is that we use a *fixed assignment* for atoms, in view of the subformula $B^4 = B^1$ of Definition 1. Hence, for the quantifier block $\forall V^4$, it is sufficient to take only atoms from $V^4 \setminus B^4$ into account and replace all occurrences of atoms $v^4 \in B^4$ by v^1 within the remaining part of the formula. The modified translation is given as follows.

Definition 2 *Let $\Pi = (P, Q, 2^A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, and $A, B \subseteq V$. Then,*

$$\mathcal{T}[\Pi] = \Phi_\Pi \wedge \forall (V^4 \setminus B^4)\, \Psi_\Pi[B^4/B^1],$$

where Φ_Π and Ψ_Π are defined as in Definition 1 and $\Psi_\Pi[B^4/B^1]$ denotes the QBF resulting from Ψ_Π by replacing all occurrences of atoms $v^4 \in B^4$ by v^1.

For illustration, Table 1 depicts the different outcomes of the two encodings for the PQIP $\Pi = (P, Q, 2^A, \subseteq_B)$ from Example 1 with $A = B = \{a, b\}$.

Lemma 1 *For any PQIP Π, the QBFs $\mathcal{S}[\Pi]$ and $\mathcal{T}[\Pi]$ are logically equivalent.*

As an immediate consequence, we thus obtain the following results.

Theorem 1 *Let $\Pi = (P, Q, 2^A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, $A, B \subseteq V$, $X \subseteq A$, and $Y \subseteq V$. Then, (X, Y) is a counterexample for Π iff $Y^1 \cup X^2 \models \mathcal{T}[\Pi]$. Moreover, Π holds iff the closed QBF $\mathsf{T}[\Pi] = \forall V^1 \forall A^2 \neg \mathcal{T}[\Pi]$ is valid.*

Theorem 2 *Let $\Pi = (P, Q, 2^A, =_B)$ be a PQEP, $At(P \cup Q) = V$, $A, B \subseteq V$, $X \subseteq A$, and $Y \subseteq V$. Then, (X, Y) is a counterexample for Π iff $Y^1 \cup X^2 \models \mathcal{T}[\Pi^{\rightarrow}] \vee \mathcal{T}[\Pi^{\leftarrow}]$. Moreover, Π holds iff $\mathsf{T}[\Pi] = \forall V^1 \forall A^2 (\neg \mathcal{T}[\Pi^{\rightarrow}] \wedge \neg \mathcal{T}[\Pi^{\leftarrow}])$ is valid.*

Obviously, these encodings, as well as the ones from the previous section, are (i) always linear in the size of P, Q, A, and B, and (ii) possess at most two quantifier alternations in any branch of the formula tree. The latter shows that any such encoding is easily translated into a $(3, \forall)$-QBF. Thus, the complexity of evaluating these QBFs is not harder than the complexity of the encoded decision problems, which shows adequacy in the sense of Besnard *et al.* [1].

We proceed with a discussion how our new reduction can be simplified for special cases. Recall that by a proper parameterisation of a PQIP (resp., PQEP) also some important special cases of correspondence checking can be realised. All simplifications outlined below have been implemented in our extension of $\mathsf{cc}\top$.

Ordinary inclusion with projection. For problems of form $(P, Q, 2^A, \subseteq_B)$ with $A = \emptyset$ we get that all terms $(A^i \leq A^j)$ are trivially true and can therefore be eliminated. Also, the free variables of $\mathcal{T}[\Pi]$ reduce to V^1. We obtain that $\mathcal{T}[\Pi]$ is equivalent to

$$\Big(P^{\langle 1,1\rangle} \wedge \forall V^3\big((V^3 < V^1) \to \neg P^{\langle 3,1\rangle}\big)\Big)\wedge \forall(V^4 \setminus B^4)\,\Big(Q^{\langle 4,4\rangle} \to \exists V^5((V^5 < V^4) \wedge Q^{\langle 5,4\rangle})\Big)[B^4/B^1].$$

Still, on each branch of the formula tree there are at most two quantifier alternations witnessing the Π_3^P-complexity of this special case.

Relativised uniform inclusion. Next, we analyse special settings without projection, i.e., problems of form $(P, Q, 2^A, \subseteq_B)$ with $B = \mathcal{U}$. Further special cases are then obtained by setting $A = \emptyset$ and $A = \mathcal{U}$, respectively. In view of of the Π_2^P-complexity result for problems without projection, we expect that the number of quantifier alternations in the resulting QBFs decreases by one. In fact, $\mathcal{T}[\Pi]$ simplifies to

$$\Phi_\Pi \wedge \Big(Q^{\langle 1,1\rangle} \to \exists V^5\big((A^2 \le A^5)\wedge (V^5 < V^1) \wedge Q^{\langle 5,1\rangle}\big)\Big). \quad (2)$$

Observe that the quantifier block $\forall(V^4 \setminus B^4)$ vanishes since $V \setminus B = \emptyset$. Thus, all atoms v^4 in the encoding are replaced by v^1. The structure of the formula now matches the Π_2^P-complexity result for relativised uniform inclusion. Interestingly, QBF (2) is *satisfiability equivalent* to the even simpler formula

$$\mathcal{T}^\circ[\Pi] = \Phi_\Pi \wedge (Q^{\langle 1,1\rangle} \to ((V^2 < V^1) \wedge Q^{\langle 2,1\rangle}),$$

where the quantifier block $\exists V^5$ is removed as well. Observe that satisfiability equivalence of the two formulas entails that $\mathcal{T}^\circ[\Pi]$ does no longer encode *all* counterexamples. However, the simplification in $\mathcal{T}^\circ[\Pi]$ does not influence the number of quantifier alternations.

Uniform inclusion. For the case of (plain) uniform inclusion, i.e., problems of the form $(P, Q, 2^A, \subseteq_B)$ with $A = B = \mathcal{U}$, no further simplification is obtained compared to (2), except that each occurrence of A^i is now given by V^i. As uniform inclusion is a special case of relativised uniform inclusion, also this QBF is satisfiability equivalent to $\mathcal{T}^\circ[\Pi]$.

Ordinary inclusion. Finally, concerning ordinary inclusion, i.e., problems of the form $(P, Q, 2^A, \subseteq_B)$ with $A = \emptyset$ and $B = \mathcal{U}$, we observe similar effects as in the encoding for ordinary inclusion with projection. In particular, all terms $(A^i \le A^j)$ can be eliminated because $A = \emptyset$. Also, the free variables of $\mathcal{T}[\Pi]$ reduce to V^1. Hence, $\mathcal{T}[\Pi]$ is equivalent to

$$\Big(P^{\langle 1,1\rangle} \wedge \forall V^3\big((V^3 < V^1) \to \neg P^{\langle 3,1\rangle}\big)\Big)\wedge \Big(Q^{\langle 1,1\rangle} \to \exists V^5((V^5 < V^1) \wedge Q^{\langle 5,1\rangle})\Big).$$

The QBF is true under interpretation Y^1 if $Y \in AS(P)$ but $Y \notin AS(Q)$. Note that the structure of the closed QBF $\mathsf{T}[\Pi]$, given by $\forall V^1(\neg\mathcal{T}[\Pi^{\to}] \wedge \neg\mathcal{T}[\Pi^{\leftarrow}])$, then witnesses the Π_2^P-membership of ordinary equivalence.

As ordinary equivalence is a special case of relativised uniform equivalence, we can obtain a further simplification in terms of $\mathcal{T}^\circ[\Pi]$. Indeed, $\mathcal{T}^\circ[\Pi]$ reduces here to

$$\Big(P^{\langle 1,1\rangle} \wedge \forall V^3\big((V^3 < V^1) \to \neg P^{\langle 3,1\rangle}\big)\Big)\wedge \Big(Q^{\langle 1,1\rangle} \to ((V^2 < V^1) \wedge Q^{\langle 2,1\rangle})\Big).$$

Hence, we have shown that all special cases with $B = \mathcal{U}$ have in common that the encodings $\mathsf{T}[\cdot]$ simplify to QBFs with at most one quantifier alternation in each branch of the formula, witnessing the Π_2^P-membership of those problems.

4. Experiments

In this section, we present a preliminary experimental evaluation of our implementation. The goal of the experiments is to clarify the interplay of different QBF solvers, different encodings, and different problem settings in terms of run-time performance. In the spirit of previous experiments with ccT [12], we use the reduction from QBFs to PQIPs given by the Π_3^P-hardness proof for deciding PQIPs [14]. This provides us with a class of random benchmark problems for ccT which captures the inherent hardness of the problem. More precisely, the method is as follows:

1. generate a random $(3, \forall)$-QBF Φ in PDNF;
2. reduce Φ to a PQIP $\Pi = (P, Q, 2^A, \subseteq_B)$ such that Π holds iff Φ is valid [14];
3. apply ccT to derive the corresponding encoding Ψ for Π.

Our benchmark set consists of 1000 instances. The randomly generated QBFs of Step 1 contain 24 different atoms each. From those 24 atoms, each quantifier block bounds 8 of them. Each term in the PDNF contains 4 atoms which are selected by random from the 24 atoms and are negated with probability 0.5. The whole formula consists of 38 terms. From the 1000 instances, 506 evaluate to true and 494 evaluate to false. Thus, the ratio between true and false instances is close to 1. Therefore, having easy-hard-easy

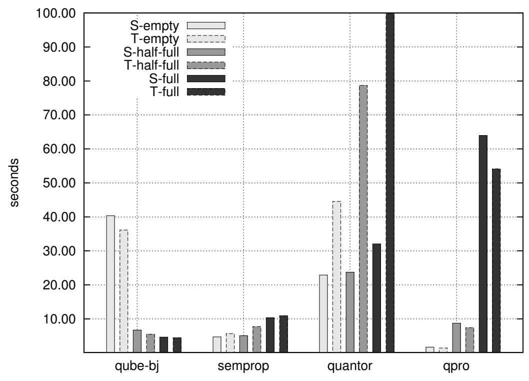

Figure 2. Median running times for different solvers, encodings, and problem settings.

patterns in mind, we suppose the benchmark set to be located in a rather hard region. From each Φ, we construct a PQIP $\Pi = (P, Q, 2^A, \subseteq_B)$ such that Φ is true iff Π holds. It is important to notice that P, Q, and B are determined by the reduction but the context A can be chosen arbitrarily.

For our experiments, we use three different settings, namely the empty context $A = \emptyset$, the full context $A = \mathcal{U}$, and an in-between setting $\emptyset \subseteq A \subseteq \mathcal{U}$. For the last setting, each atom occurring in one of the two programs P and Q is in A with probability 0.5. We consider both encodings from PQIPs to QBFs, $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, together with the three settings for the context. The QBFs stemming from $\mathsf{S}[\cdot]$ possess 197 atoms each for the empty context; 221 atoms (on average) for the half-full context; and 246 atoms for the full context. For QBFs from $\mathsf{T}[\cdot]$, the respective numbers are 189, 213, and 238.

We compare the QBF solvers `semprop` [10] (release 24/02/02), `qube-bj` [9] (v1.2), `quantor` [2] (release 25/01/04), and `qpro` [5]. We selected these solvers because they proved to be competitive in previous QBF evaluations and yielded only correct results on our benchmarks. The solvers `qpro`, `qube-bj`, and `semprop` are based on the standard DPLL decision procedure extended by special learning techniques whereas `quantor` implements a combination of resolution and variable expansion. All solvers except `qpro` require the input to be in prenex conjunctive normal form. Thus, for those solvers, an intermediate prenexing step is necessary. In general, this prenexing step is not deterministic and different *prenexing strategies* [4] are possible. However, for our instances, the structure of the prenex is fixed in such a way that avoiding an increase of the number of quantifier alternations during the transformation to PNF can only be accomplished by placing each quantifier into a uniquely determined quantifier block of the target $(3, \forall)$-QBF. It is worth mentioning that for both translations `ccT` encodes the *complementary problem*, i.e., generates QBFs of form $\neg\mathsf{S}[\cdot]$ or $\neg\mathsf{T}[\cdot]$ if projection is used. The reason is to avoid an additional quantifier alternation after the transformation to PCNF—details are discussed in previous work [13].

After that prenexing step, QBFs from $\mathsf{S}[\cdot]$ consist (on average) of 1035 clauses over 632 atoms (for the empty context), 1203 clauses over 728 atoms (for the half-full context), and 1378 clauses over 828 atoms (for the full context). For $\mathsf{T}[\cdot]$, the numbers are: 1003 clauses over 608 atoms (for the empty context), 1171 clauses over 704 atoms (for the half-full context), and 1346 clauses over 802 atoms (for the full context).

All experiments were carried out on a 3.0 GHz Dual Intel Xeon workstation, with 4 GB of RAM and Linux version 2.6.8.

Figure 2 summarises the results of the comparison. The different QBF solvers, encodings ($\mathsf{S}[\cdot]$, $\mathsf{T}[\cdot]$), and settings for the context (empty, half-full, full, respectively) are given on the abscissa, and the median running times in seconds are depicted on the ordinate.

A very interesting observation is that the alternative encoding $\mathsf{T}[\cdot]$ does not achieve faster running times for all solvers, although it uses less variables. For `qpro` and `qube-bj`, QBFs from $\mathsf{T}[\cdot]$ are solved—as one would expect—faster. This is not the case for `semprop` and `quantor`, where `semprop` solves QBFs from $\mathsf{S}[\cdot]$ slightly faster and `quantor` solves such QBFs much faster (the bar for `quantor` with full context and encoding $\mathsf{T}[\cdot]$ illustrates that the median value is above 100).

The next interesting point is the connection between running time and context parameterisation. The non-normal-form solver `qpro` achieves best results for the empty context but rather poor results for the full context. For `qube-bj` the contrary is true, however, i.e., it achieves best results for the full context but poor results for the empty context—a quite surprising observation. Finally, the most robust solver in this aspect is `semprop`. Recall that each of the derived PQIPs $(P, Q, 2^A, \subseteq_B)$ either holds for any A, or does not hold for any A. The assignments of atoms from X^1 in our encodings which "guess" context-program candidates are thus completely irrelevant for the truth value of the QBFs. Now, as `qpro` does not implement any heuristics concerning the selection of atoms, it is no longer surprising that running times scale exponentially when the context gets larger. The heuristics realised in `semprop` seem to avoid that too much time is spend on finding assignments for those "decoy" variables. On the other hand, `qube-bj` suffers from the absence of those variables.

Figures 3–6 provide some deeper insights concerning the running-time behaviour of the non-normal-form solver `qpro` and the normal-form solvers `semprop`, `qube-bj`, and `quantor`, respectively. For those figures, the abscissa

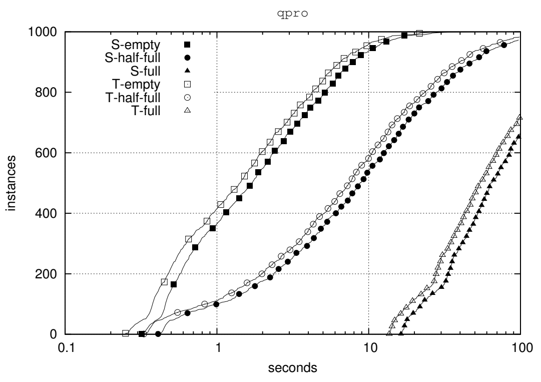

Figure 3. Run-time distribution for `qpro`.

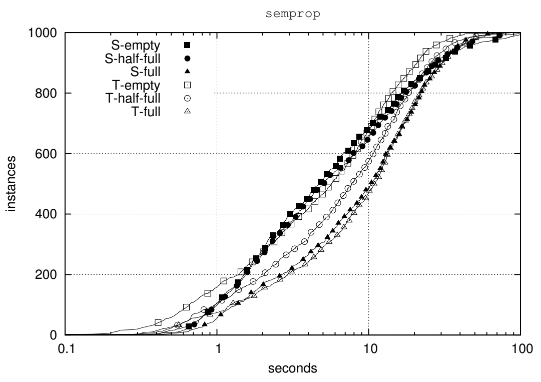

Figure 4. Run-time distribution for `semprop`.

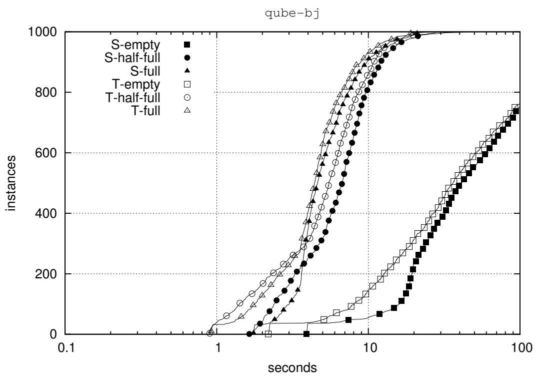

Figure 5. Run-time distribution for `qube`.

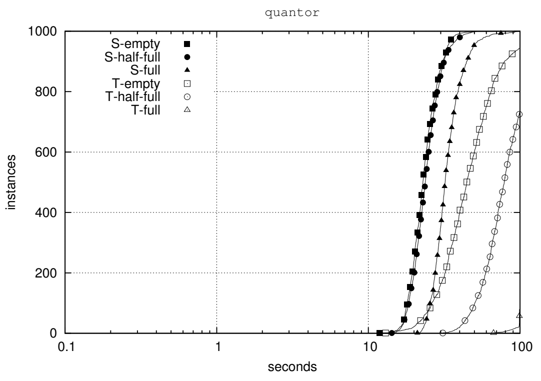

Figure 6. Run-time distribution for `quantor`.

gives the running time in seconds (scaled logarithmically) and the ordinate gives the number of solved problem instances. This means that for each running time in the data we depict how many instances were solved with running time less or equal to that time. The different curves correspond to the different combinations of the chosen encoding and context parameterisation. For better legibility, different symbols are attached to the curves. Figure 3 is a good illustration of how `qpro` benefits from the alternative encoding: the respective curves for $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$ are running in parallel. The similarity of the median running times for `semprop` in Figure 2 extends to quite similar curves in Figure 4 for the whole distribution. Note that symmetric curves (with respect to the median) on a logarithmically scaled axis imply skewed distribution of the data, i.e., low deviation for instances with running times below the median and high deviation for instances with running times above the median. Figure 3 provides some insight into the rather odd behaviour of `qube-bj` on this set of problem instances. While the curves for full and half-full context are rather similar, the curves for the empty context are standing out and illustrate the higher effort for `qube-bj` to solve them. The sharp inclination of the curves for `quantor` (Figure 6) implies that there is not much deviation in the data. Here, the running times of most instances are close to the median. Moreover, compared to the other systems, there are no instances with short running times, more precisely shorter than 11 seconds.

For space reasons, we omit a deeper analysis of the running times separated by true and false instances. The tendance is that false instances are solved faster on average. However, for empty and half-full context, `qube-bj` is faster on the true instances.

5. Conclusion

In this paper, we discussed an extension of the system `ccT` for deciding refined versions of uniform equivalence and inclusion for disjunctive logic programs under the answer-set semantics. Such correspondence problems allow to restrict the alphabet of the context class and facilitate the removal of auxiliary atoms in the comparison—two important concepts for program comparisons in practice. The tool is based on an efficient reduction to QBFs, which itself is motivated by the high complexity of the correspondence problems. While the theoretical basis was established in previous work [14], we introduced alternative encodings for PQIPs and PQEPs, and discussed simplifications realised within the new extension of ccT. We complemented our discussion with an analysis of experiments with different QBF solvers which reveal interesting differences of the solvers depending on the particular problem parameterisation and the choice of the encoding. Moreover, our encodings also provide an interesting benchmark set for QBF solvers, for which there are only a few structured problems with more than one quantifier alternation available.

As related work, we mention the system DLPEQ [15] for deciding ordinary equivalence, which is based on a reduction to logic programs, and the system SELP [3] for checking strong equivalence, which is based on a reduction to classical logic quite in the spirit of our implementation approach. An open topic for future work is, on the one hand, the extension of our work to more general classes of programs and, on the other hand, research concerning the equivalence of nonground programs. Also, we plan to conduct experiments with more real-world oriented benchmarks, like ones stemming from planning, diagnosis, and scheduling domains. In fact, we are currently running an extensive suite of experiments using different programs representing specific diagnosing problems. These programs are obtained from student data of a laboratory course on logic programming at our university.

References

[1] 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.

[2] A. Biere. Resolve and Expand. In *7th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2004*), volume 3542 of *LNCS*. Springer, 2005.

[3] Y. Chen, F. Lin, and L. Li. SELP - A System for Studying Strong Equivalence Between Logic Programs. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2005*), volume 3552 of *LNAI*, pages 442–446. Springer, 2005.

[4] U. Egly, M. Seidl, H. Tompits, S. Woltran, and M. Zolda. Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing* (*SAT 2003*). *Selected Revised Papers*, volume 2919 of *LNCS*, pages 214–228, 2004.

[5] U. Egly, M. Seidl, and S. Woltran. A Solver for QBFs in Nonprenex Form. In *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI 2006*), 2006.

[6] T. Eiter and M. Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics. In *Proceedings of the 19th International Conference on Logic Programming* (*ICLP 2003*), volume 2916 of *LNCS*, pages 224–238. Springer, 2003.

[7] T. Eiter, H. Tompits, and S. Woltran. On Solution Correspondences in Answer Set Programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI 2005*), pages 97–102, 2005.

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

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

[10] R. Letz. Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas. In *Proceedings of the 11th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods* (*TABLEAUX 2002*), volume 2381 of *LNCS*, pages 160–175, 2002.

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

[12] J. Oetsch, M. Seidl, H. Tompits, and S. Woltran. ccT: A Tool for Checking Advanced Correspondence Problems in Answer-Set Programming. In *Proceedings of the 15th International Conference on Computing* (*CIC 2006*), pages 3–10. IEEE Computer Society Press, 2006.

[13] J. Oetsch, M. Seidl, H. Tompits, and S. Woltran. A Tool for Advanced Correspondence Checking in Answer-Set Programming. In *Proceedings of the 11th International Workshop on Nonmonotonic Reasoning* (*NMR 2006*). TU Clausthal IfI Technical Report Series, 2006.

[14] J. Oetsch, H. Tompits, and S. Woltran. Facts do not Cease to Exist Because They are Ignored: Relativised Uniform Equivalence with Answer-Set Projection. In *Proceedings of the 22nd National Conference on Artificial Intelligence* (*AAAI 2007*), pages 458–464. AAAI Press, 2007.

[15] E. Oikarinen and T. Janhunen. Verifying the Equivalence of Logic Programs in the Disjunctive Case. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2004*), volume 2923 of *LNCS*, pages 180–193. Springer, 2004.

[16] L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[17] H. Tompits and S. Woltran. Towards Implementations for Advanced Equivalence Checking in Answer-Set Programming. In *Proceedings of the 21st International Conference on Logic Programming* (*ICLP 2005*), volume 3668 of *LNCS*, pages 189–203. Springer, 2005.

[18] S. Woltran. Characterizations for Relativized Notions of Equivalence in Answer Set Programming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA 2004*), volume 3229 of *LNCS*, pages 161–173. Springer, 2004.

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