FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Beyond Uniform Equivalence between Answer-Set Programs

Relativisation and Projection

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computational Intelligence

eingereicht von

Johannes Oetsch
Matrikelnummer 0025631

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: A.o. Univ. Prof. Dr. Hans Tompits

Wien, 31.09.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Johannes Oetsch
Telekygasse 24, 1190 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First, I would like to thank Hans Tompits for supervising this work and for his excellent introduction to scientific research. Furthermore, I want to thank Martina Seidl and Stefan Woltran for their collaboration on many paper projects related to this thesis. Also, I am graceful to Elfriede Nedoma, Eva Nedoma, and Matthias Schlögel for their important administrative and technical support. Finally, I would like to thank my parents, Elke Oetsch and Hans Oetsch, for all the support and for not giving up hope that I would eventually finish my master thesis.

This work has been supported by the FWF project "Formal Methods for Comparing and Optimizing Nonmonotonic Logic Programs" under grant P18019 and by the FWF project "Methods and Methodologies for Developing Answer-Set Programs" under grant P21698.

Deutsche Zusammenfassung

Diese Arbeit beschäftigt sich mit erweiterten Äquivalenzbegriffen für nichtmonotone logische Programme unter der Answer-Set Semantik. Dieses Thema erfreut sich zunehmendem Interesse da solche Äquivalenzbegriffe die theoretische Basis für viele Entwicklungsaufgaben wie Optimierung von Programmen, Debugging, modulares Programmieren und Verifikation bilden.

Verschiedene Äquivalenzbegriffe für Logikprogramme sind ein relevanter Forschungsschwerpunkt auf dem Gebiet der *Answer-Set Programmierung* (ASP). Ein prominenter Vertreter eines solchen Äquivalenzbegriffes ist *uniforme Äquivalenz*, welche überprüft ob zwei Programme, vereint mit einer beliebigen Menge von Fakten, die gleiche Semantik besitzen. In dieser Diplomarbeit studieren wir eine Familie von verfeinerten Versionen der uniformen Äquivalenz, nämlich *relativierte uniforme Äquivalenz mit Projektion*, welche gewöhnliche uniforme Äquivalenz durch zwei zusätzliche Parameter ergänzt: einen, um das Eingabealphabet der Programme zu spezifizieren, und einen, um das Ausgabealphabet festzulegen. Der zweite Parameter wird benötigt um Answer Sets auf bestimmte Ausgabeatome zu projizieren. Diese Answer-Set Projektion erlaubt es, im speziellen, Programme, die auf Hilfsatome zurückgreifen, miteinander zu vergleichen. Dieser Aspekt ist aufgrund praktischer Erfordernisse unerlässlich.

Wir führen neue semantische Charakterisierungen für die betrachtete Formen der Programmäquivalenz ein, und wir analysieren die strukturelle Berechnungskomplexität entsprechender Entscheidungsprobleme. Im allgemeinen Fall befinden sich diese Probleme auf der dritten Ebene der polynomiellen Hierarchie und können daher, unter den üblichen koplexitätstheoretischen Annahmen, nicht auf (propositionale) Answer-Set Programme selbst reduziert werden. Dennoch sind Problemreduktionen auf quantifizierte Boole'sche Formeln (QBFs) möglich. Wir beschreiben solche Reduktionen auf QBFs und diskutieren Vereinfachungen für spezielle Sonderfälle von Äquivalenzproblemen.

Die betrachteten Übersetzungen haben die Eigenschaft, dass sie immer effizient, sogar in linearer Zeit, konstruierbar sind. Diese QBF Reduktionen liefern die Basis für eine Prototyp-Implementierung in Form des Systems `cc⊤`. Der in diesem System realisierte Ansatz basiert auf der Übersetzung eines Programmkorrespondenzproblems in QBFs sowie der Verwendung externer QBF Beweiser zur Lösung der resultierenden Formeln. Wir beschreiben eine Anwendung von `cc⊤` um die Korrektheit von Programmen zu verifizieren, die als Lösungen von Studenten im Rahmen von Lehrveranstaltungen über logische Programmierung und Wissensrepräsentation an der TU Wien eingereicht wurden. Der zugrundeliegende Äquivalenzbegriff um Korrektheit zu entscheiden ist relativierte uniforme Äquivalenz mit Projektion. Schließlich komplementieren wir unsere Untersuchungen mit einer Leistungsevaluierung von `cc⊤`, welche zeigt, dass es wesent-

lich ist zwischen verschiedenen QBF Beweisern zu unterscheiden um eine optimale Performanz zu erreichen.

Abstract

This thesis deals with advanced notions of equivalence between nonmonotonic logic programs under the answer-set semantics, a topic of increasing interest because such notions form the basis for program optimisation, debugging, modular programming, and program verification.

In fact, there is extensive research in *answer-set programming* (ASP) dealing with different notions of equivalence between programs. Prominent among these notions is *uniform equivalence* which checks whether two programs have the same semantics when joined with an arbitrary set of facts. In this thesis, we study a family of more fine-grained versions of uniform equivalence, viz. *relativised uniform equivalence with projection*, which extends standard uniform equivalence in terms of two additional parameters: one for specifying the input alphabet and one for specifying the output alphabet for programs. In particular, the second parameter is used for *projecting* answer sets to a set of designated output atoms. Answer-set projection, in particular, allows to compare programs that make use of different auxiliary atoms which is important for practical programming aspects.

We introduce novel semantic characterisations for the program correspondence problems under consideration and analyse the computational complexity of deciding these problems. In the general case, deciding these problems lies on the third level of the polynomial hierarchy. Therefore, this task cannot be efficiently reduced to propositional answer-set programs itself (under the usual complexity-theoretic assumptions). However, reductions to *quantified Boolean formulas* (QBFs) are feasible. Indeed, we provide efficient, in fact, linear-time constructible, reductions to QBFs and discuss simplifications for certain special cases. These QBF reductions yield the basis for a prototype implementation, the system ccT, for deciding correspondence problems by using external QBF solvers. We discuss an application of ccT for verifying the correctness of students' solutions drawn from a laboratory course on logic programming and knowledge representation at the Vienna University of Technology, employing relativised uniform equivalence with projection as the underlying program correspondence notion. We complement our investigation by discussing a performance evaluation of ccT, showing that discriminating among different back-end solvers for quantified propositional logic is a crucial issue towards optimal performance.

Contents

CHAPTER 1

Introduction

An important issue in software development is to determine whether two encodings of a given problem are equivalent, i.e., whether they compute the same result on a given problem instance. Although the question is well known to be undecidable for Turing-complete programming languages, there are important programming languages for knowledge representation (KR) where it is decidable. Our object of investigation is one such language, viz. the class of *disjunctive logic programs* (DLPs) *under the answer-set semantics* [34].

Logic programs under the answer-set semantics represent the canonical instance of the general *answer-set programming* (ASP) *paradigm*, an important approach for declarative problem solving. The characteristic feature of ASP is that solutions to problems are given by the models (the "answer sets") of their encodings and not by proofs as in traditional logic-based formalisms. ASP has been proven useful in a variety of domains, such as semantic-web reasoning [71], systems biology [36], planning [23], diagnosis [22,53], configuration [79], multi-agent systems [4], cladistics [11,31], super optimisation [10], etc. The success of ASP is mainly due to two reasons: First, the performance of answer-set solvers, like `clasp` [33, 72] or `DLV` [17, 44], is improving continuously [12, 16]. Second, ASP offers an expressive high-level specification language and thus allows to model problems with ease. For a comprehensive introduction to ASP, we refer to the well-known textbook by Baral [3].

Given the nonmonotonic nature of DLPs under the answer-set semantics, a standard equivalence notion in the sense that two programs are viewed as being equivalent if they have the same answer sets is too weak to yield a replacement property like in classical logic. That is to say, given a program M along with some subprogram $P \subseteq M$, when replacing P with an equivalent program Q in M, it is not guaranteed that $Q \cup (M \setminus P)$ is equivalent to M. This led to the introduction of stricter notions of equivalence, viz. *strong equivalence* [46] and *uniform equivalence* [25].

In fact, this research was for the most part initiated by the seminal work of Lifschitz, Pearce, and Valverde [46] about *strong equivalence*. Albeit the latter notion in effect amounts to a replacement property by definition, and thus circumvents the failure of ordinary equivalence in this regard, it is however too restrictive for certain applications. This led to the investigation of more liberal notions, chiefly among them *uniform equivalence* [25]: uniform equivalence checks whether two

programs have the same answer sets for any arbitrary *input*, i.e., when joined with any set of facts.

In more formal terms, two programs P and Q are strongly equivalent iff, for any program R (the "context program"), $P \cup R$ and $Q \cup R$ have the same answer sets, and P and Q are uniformly equivalent iff the former condition holds for any set R of facts. For illustration, consider the programs

$$P = \{a \vee b \leftarrow\} \quad \text{and} \quad Q = \left\{ \begin{array}{l} a \leftarrow \text{not } b\,, \\ b \leftarrow \text{not } a \end{array} \right\},$$

which express the nondeterministic selection of one of the atoms a or b. Both programs yield the same answer sets, viz. $\{a\}$ and $\{b\}$. Nevertheless, consider the context program

$$R = \left\{ \begin{array}{l} a \leftarrow b\,, \\ b \leftarrow a \end{array} \right\}.$$

The unique answer set of $P \cup R$ is $\{a, b\}$ while $Q \cup R$ has no answer set at all. Thus, P and Q are not strongly equivalent. On the other hand, P and Q are uniformly equivalent since any addition of facts does not close the cycle between a and b, and thus the difference between disjunction and guessing via default negation does not come into effect in this case.

While strong equivalence is relevant for program optimisation and modular programming in general [26, 49, 67], uniform equivalence is useful in the context of hierarchically structured program components, where lower-layered components provide input for higher-layered ones. We note that uniform equivalence was first studied in the context of datalog programs as a decidable approximation of datalog equivalence [78].

Strong and uniform equivalence are, however, too restrictive in the sense that standard programming techniques like the use of local (auxiliary) variables, which may occur in some subprograms but which are ignored in the final computation, is not taken into account. In other words, these notions do not admit the *projection* of answer sets to a set of designated output letters. To illustrate this, consider programs

$$P' = P \cup \{a \leftarrow c\} \quad \text{and} \quad Q' = Q \cup \left\{ \begin{array}{l} a \leftarrow e, c\,, \\ e \leftarrow \end{array} \right\},$$

where P and Q are the programs from the above. While the rule $a \leftarrow c$ in P' expresses that a is selected if c is known, the same condition is formulated in Q' using an additional fact e. The augmented programs P' and Q' are not uniformly equivalent as each answer set of Q' contains e which is not in any answer set of P'. However, if we take a and b as designated output letters and determine uniform equivalence on the basis of answer sets projected to $\{a, b\}$, equivalence does hold.

In previous work, Eiter, Tompits, and Woltran [30] introduced a general framework for defining parameterised notions of program correspondence, allowing for both answer-set projection as well as the specification of the context class of programs that should be used for comparisons. This framework thus generalises not only strong and uniform equivalence but also *relativised* versions thereof [87] (where "relativised" means that the alphabet of the context class is an additional parameter). In their analysis, Eiter, Tompits, and Woltran [30] focused on correspondence problems for propositional DLPs effectively generalising strong equivalence—in other words, they considered correspondence problems amounting to *relativised strong equivalence with projection*.

The system cc$\top$ [55] was developed as a checker for the type of correspondence problems which were the main focus of the analysis of Eiter et al. [30], i.e., for correspondence problems generalising strong equivalence. The main approach of cc$\top$ to verify these kinds of problems is to reduce them to the satisfiability problem of quantified propositional logic. 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). 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 presumably infeasible to compute them by means of (propositional) 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 thesis, we complement these investigations by considering correspondence problems that amount to *relativised uniform equivalence with projection*. More formally, we assume two fixed alphabets A and B (i.e., sets of atoms) which serve as parameters of our equivalence problems. Then, given two programs P and Q, we check, for any set $R \subseteq A$ of facts, whether the answer sets of $P \cup R$ and $Q \cup R$ projected to B coincide. (In a relativised strong equivalence problem with projection, R would be a program over A.) Like Eiter, Tompits, and Woltran [30], we also consider *implication problems*, i.e., checking set inclusion of the projected answer sets rather than equality. In such a setting, Q can be viewed as an approximation of P which is sound with respect to cautious reasoning from P. Note that since relativised strong equivalence (resp., implication) with projection implies relativised uniform equivalence (resp., implication) with projection (with respect to the same alphabets) but not vice versa, characterisations of the former kinds of problems in general do not capture the latter kinds of problems and so new methods are needed. Developing such characterisations is actually one of the main achievements of this thesis.

Taking a database point of view in which logic programs are seen as queries over databases, we refer to the equivalence problems studied here as *propositional query equivalence problems* (PQEPs) and to the considered implication problems as *propositional query implication problems* (PQIPs).

Contribution of the Thesis

The main contributions of this thesis can be summarised as follows:

- We introduce semantic characterisations for PQEPs and PQIPs in terms of novel structures associated with each program. We have that a PQEP holds iff the associated structures coincide, and a PQIP holds iff the structures meet set inclusion. Interestingly, our characterisation differs from the well-known characterisation of (relativised) uniform equivalence in terms of (relativised) UE-models [25, 87] in case the projection set is unrestricted. Thus, as a by-product, we obtain a new characterisation of these special forms of equivalence.

- We analyse the computational complexity of checking PQEPs and PQIPs. While checking the kinds of correspondence problems analysed by Eiter, Tompits, and Woltran [30] is

Π_4^P-complete in general, checking PQEPs or PQIPs is "only" Π_3^P-complete. As checking relativised strong or uniform equivalence is Π_2^P-complete [87], projection thus adds a source of complexity, providing the polynomial hierarchy does not collapse.

- Since checking PQEPs and PQIPs is computationally hard—lying on the third level of the polynomial hierarchy—a similar reduction approach to QBFs as for the their strong counterparts is viable. In fact, we provide efficient reductions of PQEPs and PQIPs into quantified propositional logic and discuss simplifications for certain special cases. We also discuss an extension of `cc⊤` [55] for checking PQIPs and PQEPs. The new component of `cc⊤` is based on these QBF reductions. Given the availability of efficient off-the-shelf solvers for the latter language, we thus can employ them for deciding the correspondence problems under consideration.

- As `cc⊤` admits the use of different QBF solvers as back-end engines, we report about an experimental evaluation of the tool using a set of benchmark problems. This evaluation shows the runtime behaviour of the system depending on a chosen solver. The experiments were based on a set of parameterisable benchmarks stemming from the hardness proof of the complexity analysis for the corresponding equivalence problems. These benchmarks have the particular advantage that they can be used to easily verify the correctness not only of `cc⊤` but also of the employed QBF solvers. This proved to be very helpful during the development of the system. The experiments show that discriminating among different back-end QBF solvers is crucial towards optimal performance.

- We discuss how PQEPs can be used to verify the correctness of solutions provided by students as part of their assignments for a laboratory course on logic programming and knowledge-based systems at our university, relative to a reference solution. The assignments are taken from the domain of model-based diagnosis and use the diagnosis front-end of the well-known answer-set solver `DLV` [17, 44] as underlying reasoning engine. The main difficulty for verifying the students' solutions is that PQEPs deal with propositional programs only, whilst the programs in our application scenario are non-ground. A naive grounding would not be feasible, so we resorted to a special technique restricting the domain to admissible inputs as well as employing the intelligent grounder of `DLV`. It turned out that verifying the solutions in this way yielded less false positives than the testing approach employed in the courses, which is based on a collection of sample test cases.

Organisation of the Thesis

The thesis is organised as follows. In Chapter 2, we give a formal background on logic programs under the answer-set semantics, notions of program correspondence, quantified propositional logic, and complexity theory. Then, in Chapter 3, we introduce the central objects of our investigations: PQIPs and PQEPs. We provide basic definitions and study properties as well as relations between such correspondence problems. After a model-theoretic characterisation of PQIPs and PQEPs, we provide a basic complexity analysis regarding the important reasoning tasks. In Chapter 4, we describe how PQIPs and PQEPs can be translated to QBFs. In particular, we outline the single modules used to realise such translations in detail, we discuss how the transformation

can be simplified for many special cases of correspondence problems known from the literature, and we review our method to transform resulting QBFs into certain normal forms required by most off-the-shelf QBF solvers. The tool ccT that implements respective QBF translations is subject of Chapter 5. We explain the overall methodology of ccT and illustrate how to use it with a simple example. In Chapter 6, we report on an empirical analysis of ccT and different back-end QBF solvers on different classes of benchmark problems. We outline the experimental setup and discuss the obtained results. Subsequently, in Chapter 7, we describe a case study where we used ccT in a real-world scenario. In particular, we outline an application scenario of ccT for verifying students' solutions stemming from a course on logic programming. Then, we review related work in Chapter 8. Finally, we conclude and give pointers to future work in Chapter 9.

Bibliographic Notes

The results of this thesis are published in different workshop and conference proceedings. In particular, the system ccT was first presented at the 20th Workshop on Logic Programming (WLP 2006) [58]. More thorough descriptions of the system and the underlying reduction approach for problems generalising strong equivalence appeared in different workshop proceedings [56, 57]. A system description of ccT was also accepted for the 10th European Conference on Logics in Artificial Intelligence (JELIA 2006) [54]. A detailed performance evaluation of ccT for problems generalising strong equivalence appeared in the proceedings of the 15th International Conference on Computing (CIC 2006) [55].

PQIPs and PQEPs were first introduced at the Workshop on Correspondence and Equivalence for Nonmonotonic Theories (CENT 2007) [62]. Later, a respective conference paper was accepted for the 22nd National Conference on Artificial Intelligence (AAAI 2007) both as a poster and as a regular research paper [63]. The extended component of ccT for deciding PQIPs and PQEPs was first presented at the 20th Workshop on Logic Programming (WLP 2007) [59]. A performance analysis of this component as well as a case study describing an verification application appeared in the proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (ICLP 2009) [60].

CHAPTER 2

Preliminaries

In the following, we present formal preliminaries about logic programs under the answer-set semantics and we review notions of program correspondence. Furthermore, we review quantified propositional logic and basics on complexity theory.

2.1 Logic Programs

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

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n, \tag{2.1}$$

where $n \geq m \geq l \geq 0$, all a_i, $1 \leq i \leq n$, 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 "$\leftarrow$". For a rule r of form (2.1), We refer to $a_1, \ldots, a_l$ as the *head* of r, to $a_{l+1}, \ldots, a_m$ as the *positive body* of r, and to $a_{m+1}, \ldots, a_n$ as the *negative body* of r. If the head of r is empty, we say that r is as a *constraint*.

For a program P, an atom a is *extensional* in P if it does not occur in the head of any rule in P, and a is *intensional* otherwise.

We denote by $At(P)$ the set of all atoms occurring in a program P, and we say that a program is *over* A if $At(P) \subseteq A$. We use $\mathcal{P}_A$ to refer to the set of all programs over A, and we use $\mathcal{F}_A$ to refer to the set of all programs over A that consist of facts only.

By an interpretation, we understand a set of atoms. A rule r of form (2.1) is *true* under an interpretation I, symbolically $I \models r$, iff $\{a_1, \ldots, a_l\} \cap I \neq \emptyset$ whenever $\{a_{l+1}, \ldots, a_m\} \subseteq I$ and $\{a_{m+1}, \ldots, a_n\} \cap I = \emptyset$. If $I \models r$ holds, then I is also said to be a *model* of r. As well, I is a model of a program P, symbolically $I \models P$, iff $I \models r$, for all $r \in P$.

Following Gelfond and Lifschitz [34], an interpretation I is an *answer set* of a program P iff it is a subset-minimal model of the *reduct* P^I, resulting from P by

(i) deleting all rules containing a default negated atom $\text{not } a$ 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 $AS(P)$. Note that each answer set of P is a subset of $At(P)$. As customary, we will identify sets of facts with interpretations and vice versa.

For instance, consider the following program:

$$P = \left\{ \begin{array}{rcl} male \vee female & \leftarrow & person\,, \\ adult & \leftarrow & person,\ \text{not}\ child\,, \\ person & \leftarrow & \end{array} \right\}.$$

The intended meaning is that a person is either male or female. Furthermore, a person is an adult by default. More precisely, we assume that a person is an adult if there is no evidence that the person is a child. One answer set M of P is $\{person, adult, female\}$. In order to verify this, we have to inspect the reduct of P with respect to M:

$$P^M = \left\{ \begin{array}{rcl} male \vee female & \leftarrow & person\,, \\ adult & \leftarrow & person\,, \\ person & \leftarrow & \end{array} \right\}$$

and test whether M is minimal model of P^M. This is indeed the case, thus M is an answer set of P. One can verify that $N = \{person, adult, female\}$ is the only answer set of P besides M by investigating all other subsets of $At(P)$.

We will make use of the following properties of the program reducts and the relation $\models$ in the sequel:

- For every interpretation Y and for every program P, $Y \models P^Y$ iff $Y \models P$.
- For every interpretation I and J, and for every program P and Q, the following statements are equivalent:
  (i) $I \models (P \cup Q)^J$,
  (ii) $I \models P^J \cup Q^J$, and
  (iii) both $I \models P^J$ and $I \models Q^J$.
- Let X be a set of facts and let Y be an arbitrary interpretation. Then, $X^Y = X$ and, for any interpretation Z, $Z \models X^Y$ iff $X \subseteq Z$.

Proposition 1 *Let P be a program over universe $\mathcal{U}$, X a set of facts over $\mathcal{U}$, and let Y be an interpretation. Then, $Y \in AS(P \cup X)$ iff*

(i) $Y \models P$,

(ii) $X \subseteq Y$, *and*

(iii) *for each Y', if $X \subseteq Y' \subset Y$, then $Y' \not\models P^Y$.*

Proof. By the definition of the an answer set, $Y \in AS(P \cup X)$ iff Y is a minimal model of $(P \cup X)^Y$, which holds iff Y is a minimal model of $P^Y \cup X^Y$. Since X is a set of facts, $X^Y = X$, and we obtain that $Y \in AS(P \cup X)$ iff Y is a minimal model of $P^Y \cup X$. The latter holds iff $Y \models P^Y \cup X$ and for each Y', if $Y' \subset Y$ then $Y' \not\models P^Y \cup X$. Now, $Y \models P^Y \cup X$ iff $Y \models P^Y$ and $Y \models X$. As well, $Y' \not\models P^Y \cup X$ iff $Y' \not\models P^Y$ or $Y' \not\models X$. We obtain that $Y \in AS(P \cup X)$ iff $Y \models P^Y$, $Y \models X$, and for each Y', if $Y' \subset Y$ and $Y' \models X$, then $Y' \not\models P^Y$. Finally, we use the properties that

- $Y \models P^Y$ iff $Y \models P$,
- $Y \models X$ iff $X \subseteq Y$, and
- $Y' \models X$ iff $X \subseteq Y'$,

and obtain that $Y \in AS(P \cup X)$ iff $Y \models P$, $X \subseteq Y$, and for each Y', if $X \subseteq Y' \subset Y$, then $Y' \not\models P^Y$, which concludes the proof. □

We will also make use of a splitting property introduced by Lifschitz and Turner [47]. Let P be a logic program. Then, a *splitting set* for P is any set U such that, for every rule $r \in P$ of form (2.1), if $\{a_1, \ldots, a_l\} \cap U \neq \emptyset$, then $\{a_1, \ldots, a_n\} \subseteq U$. The set of rules $r \in P$ of form (2.1) with $\{a_1, \ldots, a_n\} \subseteq U$ is the *bottom* of P relative to U.

Proposition 2 (Lifschitz [47]) *Let U be a splitting set for a program P and let B be the bottom of P with respect to U. Then, an interpretation A is an answer set of P iff there exists some $A' \in AS(B)$ such that $A \in AS((P \setminus B) \cup A')$.*

We use the following notation in the sequel: For an interpretation I and a collection $\mathcal{S}$ 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 $\mathcal{S}$ and $\mathcal{S}'$ of interpretations, an interpretation B, and $\odot \in \{\subseteq, =\}$, we define $\mathcal{S} \odot_B \mathcal{S}'$ iff $\mathcal{S}|_B \odot \mathcal{S}'|_B$.

2.2 Program Correspondence

Next, we recapitulate characterisations of some important notions of program correspondence from the literature.

One rather basic concept is the notion of *ordinary equivalence*: Two programs P and Q are ordinarily equivalent iff $AS(P) = AS(Q)$.

As noted already in the introduction, Lifschitz, Pearce, and Valverde [46] introduced the notion of *strong equivalence* where two programs P and Q are strongly equivalent iff $AS(P \cup R) = AS(Q \cup R)$, for any program R. Strong equivalence can be characterised by *SE-models* [85]: A pair (X, Y) of interpretations with $X \subseteq Y$ is an SE-model for a program P iff $Y \models P$ and $X \models P^Y$. Let $SE(P)$ denote the set of SE-models of a program P. Then, two programs P and Q are strongly equivalent iff $SE(P) = SE(Q)$ [85].

Following Eiter and Fink [25], two programs P and Q are *uniformly equivalent* iff $AS(P \cup F) = AS(Q \cup F)$, for any set F of facts. They gave a characterisation in terms of *UE-models* which are a selection of special SE-models: A pair (X, Y) of interpretations is a UE-model for

a program P iff, for every SE-model (X', Y) of P, it holds that $X \subset X'$ implies $X' = Y$. We denote the set of all UE-models of P by $UE(P)$. Two programs P and Q are uniformly equivalent iff $UE(P) = UE(Q)$.

The notion of strong equivalence has been generalised as follows. Two programs P and Q are strongly equivalent relative to a set A of atoms iff $AS(P \cup R) = AS(Q \cup R)$, for any program R over A [87]. This notion can be characterised by means of *A-SE-models*: A pair (X, Y) with $X = Y$ or $X \subset Y|_A$ is an A-SE-model for a program P iff $Y \models P$, for all $Y' \subseteq Y$ with $Y'|_A = Y|_A$, $Y' \not\models P^Y$, and $X \subset Y$ implies the existence of an $X' \subseteq Y$ with $X'|_A = X$ such that $X' \models P^Y$ [87]. Let $SE^A(P)$ denote the set of all A-SE-models of a program P. Two programs P and Q are strongly equivalent relative to a set A of atoms iff $SE^A(P) = SE^A(Q)$.

Woltran [87] also relativised the notion of uniform equivalence in the following sense: Let P and Q be two programs, P and Q are *uniformly equivalent relative to* A iff $AS(P \cup F) = AS(Q \cup F)$, for any set $F \subseteq A$. A pair (X, Y) is an *A-UE-model* of a program P iff (X, Y) is an A-SE-model for P and, for every A-SE-model (X', Y) of P, $X \subset X'$ implies $X' = Y$. We denote the set of all A-UE-models of P by $UE^A(P)$. Then, two programs P and Q are uniformly equivalent relative to A iff $UE^A(P) = UE^A(Q)$.

Following Eiter, Tompits, and Woltran [30], 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 over $\mathcal{U}$, $\mathcal{C} \subseteq \mathcal{P}_\mathcal{U}$ 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, the following relations hold:

- P and Q are ordinarily equivalent iff $(P, Q, \{\emptyset\}, =_\mathcal{U})$ holds;
- P and Q are strongly equivalent iff $(P, Q, \mathcal{P}_\mathcal{U}, =_\mathcal{U})$ holds;
- P and Q are uniformly equivalent iff $(P, Q, \mathcal{F}_\mathcal{U}, =_\mathcal{U})$ holds;
- P and Q are strongly equivalent relative to A, for $A \subseteq \mathcal{U}$, iff $(P, Q, \mathcal{P}_A, =_\mathcal{U})$ holds; and
- P and Q are uniformly equivalent relative to A, for $A \subseteq \mathcal{U}$, iff $(P, Q, \mathcal{F}_A, =_\mathcal{U})$ holds.

Eiter, Tompits, and Woltran [30] considered even more fine-grained correspondence problems, viz. problems of form $(P, Q, \mathcal{P}_A, =_B)$ and $(P, Q, \mathcal{P}_A, \subseteq_B)$ with $A, B \subseteq \mathcal{U}$. For verifying such correspondence problems, Eiter, Tompits, and Woltran [30] introduced the notion of an *A-B-certificate*: without going into details, $(P, Q, \mathcal{P}_A, =_B)$ holds iff P and Q have the same A-B-certificates. In this work, the focus is on problems of form $(P, Q, \mathcal{F}_A, =_B)$ and $(P, Q, \mathcal{F}_A, \subseteq_B)$, for $A, B \subseteq \mathcal{U}$. Inter alia, we will introduce model-theoretic characterisations for such problems similar to A-B-certificates in the sense that $(P, Q, \mathcal{F}_A, =_B)$ holds iff the associated structures coincide, and $(P, Q, \mathcal{F}_A, \subseteq_B)$ holds if the associated structures meet set inclusion.

Example 1 Recall program P from the above. Define $P' = P \setminus \{person \leftarrow\}$ and consider the following program:

$$Q = \left\{ \begin{array}{rcl} male & \leftarrow & person, \text{not } female\,, \\ female & \leftarrow & person, \text{not } male \end{array} \right\}.$$

Since $AS(P') = AS(Q) = \{\emptyset\}$, P' and Q are equivalent in the ordinary sense. Nevertheless, P' and Q are not strongly equivalent which is witnessed, e.g., by $R = \{person\}$:

$$AS(P' \cup R) = \{\{male, adult\}, \{female, adult\}\}$$

but

$$AS(Q \cup R) = \{\{male\}, \{female\}\}\,.$$

Since R contains only a fact, it follows that P and Q are not uniformly equivalent as well. However, consider problems $(P', Q, \mathcal{F}_A, =_B)$ over $\mathcal{U}$ with $A \subseteq \mathcal{U}$ and $B = \{male, female\}$. It is straightforward to verify that all such problems hold. $\Diamond$

2.3 Quantified Propositional Logic

We also make use of *quantified propositional logic*, an extension of classical propositional logic where formulas are permitted to contain quantifications over propositional atoms. In fact, quantified propositional logic constitutes the target language for our reduction approach for deciding correspondence problems. We assume that the reader is familiar with classical propositional logic and recapitulate the basic facts about quantified propositional logic in this section.

Similar to predicate logic, $\exists$ and $\forall$ are used as symbols for existential and universal quantification, respectively. It is customary to refer to formulas of quantified propositional logic as *quantified Boolean formulas* (QBFs).

The language of QBFs is inductively defined as follows:

(i) each propositional atom, including the truth constant $\top$, is a QBF;

(ii) if Φ is a QBF, then $\neg(\Phi)$ is a QBF;

(iii) if Φ and Ψ are QBFs, then $(\Phi \circ \Psi)$, for $\circ \in \{\wedge, \vee\}$, is a QBF; and

(iv) if Φ is a QBF, then $\mathsf{Q}p\,(\Phi)$, where $\mathsf{Q} \in \{\exists, \forall\}$ and p is a propositional atom, is a QBF,

(v) a QBF is constructed only according to (i)–(iv).

For two QBFs Φ and Ψ, $(\Phi \rightarrow \Psi)$ is an abbreviation for $(\neg(\Phi) \vee \Psi)$, and $(\Phi \leftrightarrow \Psi)$ is an abbreviation for $((\Phi \rightarrow \Psi) \wedge (\Psi \rightarrow \Phi))$. Also, $\bot$ abbreviates $\neg(\top)$. As usual, parentheses may be omitted as long as no ambiguities arise. If parentheses are omitted, we assume the following precedence order concerning the binding of connectives:

$$\neg\,,\ \wedge\,,\ \vee\,,\ \rightarrow\,,\ \leftrightarrow\ .$$

Note that Items (i)–(iii) above define the language of classical propositional logic.

For a QBF of form $\mathsf{Q}p\,\Psi$, where $\mathsf{Q} \in \{\exists, \forall\}$, we call Ψ the *scope* of $\mathsf{Q}p$. An occurrence of an atom p not directly preceded by $\exists$ or $\forall$ is *free* in a QBF Φ if it does not occur in the scope of a quantifier $\mathsf{Q}p$ in Φ. A QBF without free occurrences of atoms is *closed*.

The *immediate subformula* relation is defined as follows. Given formulas Φ and Ψ, formula Ψ is an immediate subformula of Φ iff Φ is of one of the following forms: $\forall p\,\Psi$, $\exists p\,\Psi$, $\neg\Psi$, $\Psi \wedge \Psi'$,

$\Psi' \wedge \Psi$, $\Psi \vee \Psi'$, or $\Psi' \wedge \Psi$, for some formula Ψ' and some atom p. As customary, the *subformula* relation is the transitive-reflexive closure of the immediate subformula relation.

In a QBF $\neg\Phi$, Φ is the scope of $\neg$. Given a QBF Φ defined according to the formation rules (i)–(v) without the use of abbreviations, each occurrence of a subformula in Φ has an associated *polarity* with respect to Φ. Polarities are either positive or negative and can be defined as follows: An occurrence of a subformula Ψ is *positive* within a formula Φ if that occurrence of Ψ is in the scope of an even number of occurrences of the negation operator $\neg$ in Φ, otherwise the occurrence of Ψ is *negative* within Φ.

Let P be a finite set of atoms. Then, $\mathsf{Q}P\,\Psi$ stands for the QBF

$$\mathsf{Q}p_1\mathsf{Q}p_2 \cdots \mathsf{Q}p_n\,\Psi\,,$$

for any list $p_1, \ldots, p_n$ of atoms such that $\{p_1, \ldots, p_n\} = P$ and $\mathsf{Q} \in \{\forall, \exists\}$.

We adopt the convention that an interpretation is identified with the set of atoms that are true in this interpretation. For an interpretation I and a QBF Φ, the relation $I \models \Phi$ is recursively defined as follows:

1. if $\Phi = \top$, then $I \models \Phi$;
2. if $\Phi = p$, where p is a propositional atom, then $I \models \Phi$ iff $p \in I$;
3. if $\Phi = \neg\Psi$, then $I \models \Phi$ iff $I \not\models \Psi$;
4. if $\Phi = \Psi \wedge \Omega$, then $I \models \Phi$ iff $I \models \Psi$ and $I \models \Omega$;
5. if $\Phi = \Psi \vee \Omega$, then $I \models \Phi$ iff $I \models \Psi$ or $I \models \Omega$;
6. if $\Phi = \exists p\,\Psi$, then $I \models \Phi$ iff $I \setminus \{p\} \models \Psi$ or $I \cup \{p\} \models \Psi$; and
7. if $\Phi = \forall p\,\Psi$, then $I \models \Phi$ iff $I \setminus \{p\} \models \Psi$ and $I \cup \{p\} \models \Psi$.

Let I be an interpretation. Then, I is a *model* of a QBF Φ iff $I \models \Phi$. We say that Φ is *true* under I if I is a model of Φ, otherwise Φ is *false* under I. A QBF Φ is *satisfiable* iff Φ possesses at least one model. Furthermore, Φ is *valid* iff $I \models \Phi$ for each interpretation I. Note that a closed QBF Φ is valid iff Φ is satisfiable. Two QBFs Φ and Ψ are *equivalent* iff Φ and Ψ have the same models. Moreover, Φ and Ψ are *satisfiability equivalent* iff Φ is satisfiable whenever Ψ is satisfiable and vice versa.

Let Φ be a QBF and X the set of all atoms with a free occurrence in Φ. Then, the *existential closure* of Φ is the QBF $\exists X\,\Phi$. Clearly, if a QBF Φ is satisfiability equivalent to a QBF Ψ, then the existential closure of Φ and the existential closure of Ψ are equivalent.

Similar to classical propositional logic, we can state a replacement property with respect to equivalent subformulas.

Proposition 3 *Consider two equivalent QBFs A and B. Then, any QBF Φ with A as a subformula is equivalent to Φ', where Φ' results from Φ by replacing one or more occurrences of A in Φ by B.*

Proof. The result follows by straightforward induction on the *logical complexity* of Φ, i.e., the number of connectives occurring in Φ. □

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

$$\mathsf{Q}_n P_n \cdots \mathsf{Q}_1 P_1 \, \phi, \tag{2.2}$$

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 $1 \leq i \leq n-1$, and $(P_1, \ldots, P_n)$ is a partition of the propositional variables occurring in ϕ. We say that Φ is in *prenex conjunctive normal form* (PCNF) iff Φ is of the form (2.2) and ϕ is in conjunctive normal form. Likewise, Φ is in *prenex disjunctive normal form* (PDNF) iff Φ is of the form (2.2) and ϕ is in disjunctive normal form.

A QBF of form (2.2) is also referred to as an (n, Q_n)-*QBF*. Any closed QBF Φ is easily transformed into an equivalent QBF Φ' in prenex normal form such that each quantifier occurrence from the original QBF Φ corresponds to a quantifier occurrence in Φ'. Let us call such a QBF Φ' a prenex normal form (PNF) *of* Φ. In general, there are different ways to obtain an equivalent QBF in PNF, cf. the work of Egly et al. [19] for more details on this issue.

2.4 Complexity Theory

We proceed by recapitulating some basic facts about *complexity theory* which deals with problem classes, their properties, and their relations among each other. For more information, we refer to the work of Papadimitriou [66]. Problems are examined with respect to their structural, i.e., problem inherent, complexity. A *problem class* is defined as a set of problems that can be solved using a certain *machine model* under bounded resources.

The classical machine model is the *Turing machine* [84]. Resources for computation are usually limited by running time and memory space. We distinguish between *deterministic* Turing machines (DTMs) and *non-deterministic* Turing machines (NDTMs). Both have the same computational power, but the NDTM branches non-deterministically from one state to a set of successor states and can execute computation in some sense in parallel. Whether a NDTM can be simulated by a DTM in polynomial time is one of the most prominent open question in theoretical computer science so far.

A *problem description* (or simply a *problem*) is a pair (L, Y), where L is a formal language and $Y \subseteq L$, representing the *positive instances*. The *decision problem* associated with (L, Y) is the problem of determining whether $I \in Y$, for some *problem instance* $I \in L$.

A *complexity class* is a collection of decision problems. In the context of this work, we consider the following prominent complexity classes:

- P, the class of problems that can be solved by a DTM in polynomial time with respect to the size of the problem instance;
- NP, the class of problems that can be solved by a NDTM in polynomial time with respect to the size of the problem instance; and
- PSPACE, the class of problems that can be solved by a DTM using polynomial space with respect to the size of the problem instance.

For a problem $A = (L, Y)$, the *complementary problem*, $\bar{A}$, is defined as $(L, L \setminus Y)$. For a problem class C, the complementary class co-C is defined as $\{\bar{A} | A \in C\}$. The following set inclusions hold:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE}.$$

It is unknown so far whether these inclusions are proper or not. Given two problems $P_1 = (L_1, Y_1)$ and $P_2 = (L_2, Y_2)$, P_1 is *polynomial-time many-to-one reducible* to P_2, denoted by $P_1 \succeq P_2$, iff there exists a function f that maps each problem instance $I_1 \in L_1$ to an instance $I_2 \in L_2$ in polynomial time such that $I_1 \in L_1$ iff $f(I_1) \in L_2$. For a class C, a problem A is called *hard with respect to* C, or C*-hard*, iff for each $B \in C$, it holds that $B \succeq A$. A is called *complete with respect to* C, or C*-complete*, iff it is hard for C and a member of C, i.e., $A \in C$. To prove completeness for a problem A with respect to a class C, it is sufficient to

1. show that $B \succeq A$ for some problem B that is hard for C, and
2. show membership of A, i.e., $A \in C$.

Many problems, especially in the field of propositional nonmonotonic reasoning, are hard for NP (resp., co-NP) and members of the class PSPACE without being hard for it. For those problems, it is appropriate to make use of the concept of *oracle classes*. An oracle for a class C can be regarded as a procedure that solves problems $A \in C$ in constant time. For a complexity class C, the class P^C (resp., NP^C) denotes the class of problems that can be solved by a DTM (resp., NDTM) with the help of an oracle for class C in polynomial time.

The *polynomial hierarchy* (PH) [80] consists of classes $\Sigma_k^P, \Pi_k^P, \Delta_k^P$, $k \geq 0$, and is inductively defined as follows: for $k = 0$,

$$\Sigma_0^P = \Pi_0^P = \Delta_0^P = \text{P},$$

and, for $k > 0$,

$$\begin{aligned} \Delta_{k+1}^P &= \text{P}^{\Sigma_k^P}, \\ \Sigma_{k+1}^P &= \text{NP}^{\Sigma_k^P}, \\ \Pi_{k+1}^P &= \text{co-}\Sigma_{k+1}^P. \end{aligned}$$

The following relations hold:

$$\begin{aligned} \Delta_k^P &\subseteq (\Sigma_k^P \cap \Pi_k^P), \\ (\Sigma_k^P \cup \Pi_k^P) &\subseteq \Delta_{k+1}^P, \text{ and} \\ \bigcup_{k=0}^{\infty} \Sigma_k^P &\subseteq \text{PSPACE}. \end{aligned}$$

For QBFs, the following property is essential:

Proposition 4 (Meyer and Stockmeyer [50]) *For every $k \geq 0$, deciding the truth of a given $(k, \exists)$-QBF is Σ_k^P-complete.*

Likewise, deciding the truth of a given $(k, \forall)$-QBF is 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. Let $T[\cdot]$ be an encoding from decision problems into QBFs. Following Besnard et al. [8], $T[\cdot]$ is *adequate* iff, for each decision problem $\mathcal{D}$,

(i) $T[\mathcal{D}]$ evaluates to true iff $\mathcal{D}$ holds,

(ii) $T[\mathcal{D}]$ is computable in polynomial time with respect to the size of $\mathcal{D}$, and

(iii) determining the truth value of the QBF resulting from $T[\mathcal{D}]$ is not computationally harder than checking whether $\mathcal{D}$ holds.

CHAPTER 3

Propositional Query Implication and Equivalence Problems

Eiter, Tompits, and Woltran [30] focused on two important instantiations of their framework, viz. on problems of form $(P, Q, \mathcal{P}_A, \subseteq_B)$ and of form $(P, Q, \mathcal{P}_A, =_B)$, where $A, B \subseteq \mathcal{U}$ are sets of atoms fixing the alphabet of the context class $\mathcal{P}_A$ and the alphabet relevant in comparing the answer sets, respectively. Our interest here are correspondence problems likewise parameterised by A and B as above, but where the context class is given by sets of facts from A rather than $\mathcal{P}_A$.

3.1 Basic Definitions and Properties

Definition 1 *Let $\mathcal{U}$ be a set of atoms, $A, B \subseteq \mathcal{U}$, and $P, Q \in \mathcal{P}_\mathcal{U}$. Then,*

(i) $(P, Q, \mathcal{F}_A, \subseteq_B)$ *is a* propositional query implication problem (over $\mathcal{U}$)*, or* PQIP*, and*

(ii) $(P, Q, \mathcal{F}_A, =_B)$ *is a* propositional query equivalence problem (over $\mathcal{U}$)*, or* PQEP.

Example 2 Consider the programs

$$P = \left\{ \begin{array}{l} a \vee b \leftarrow, \\ a \leftarrow c \end{array} \right\} \quad \text{and} \quad Q = \left\{ \begin{array}{l} a \leftarrow \text{not } b\,, \\ b \leftarrow \text{not } a\,, \\ c \leftarrow a \end{array} \right\}.$$

The answer sets of P and Q are given by $AS(P) = \{\{a\}, \{b\}\}$ and $AS(Q) = \{\{a, c\}, \{b\}\}$. Choosing $B = \{a, b\}$, we then have that

$$AS(P)|_B = AS(Q)|_B = \{\{a\}, \{b\}\}\,.$$

In fact, for $A = B = \{a, b\}$, the PQEP $(P, Q, \mathcal{F}_A, =_B)$ holds. ◊

Note that $(P, Q, \mathcal{P}_A, \subseteq_B)$ holds only if $(P, Q, \mathcal{F}_A, \subseteq_B)$ holds, but not vice versa. Indeed, the PQIP $(P, Q, \mathcal{F}_A, \subseteq_B)$ from Example 2 holds but $(P, Q, \mathcal{P}_A, \subseteq_B)$ does not hold, as witnessed by the context program

$$\left\{ \begin{array}{l} a \leftarrow b, \\ b \leftarrow a \end{array} \right\} \in \mathcal{P}_A \, .$$

It is convenient to assemble the objects witnessing the violation of a PQIP into a single concept. We introduce two versions of such a concept.

Definition 2 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP over $\mathcal{U}$. Then,*

(i) *a pair (X, Y) with $X \subseteq A$ and $Y \subseteq \mathcal{U}$ is an* explicit counterexample (over $\mathcal{U}$) *for Π iff $Y \in AS(P \cup X)$ and no Y' with $Y'|_B = Y|_B$ is contained in $AS(Q \cup X)$; and*

(ii) *a pair (X, Y) with $X \subseteq A$ and $Y \subseteq B$ is a* projective counterexample *for Π iff $Y \in AS(P \cup X)|_B$ and $Y \notin AS(Q \cup X)|_B$.*

Lemma 1 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP over $\mathcal{U}$. For any explicit counterexample (X, Y) for Π, $(X, Y|_B)$ is a projective counterexample for Π.*

Conversely, for any projective counterexample (X, Y) for Π, there exists an explicit counterexample (X, Y') with $Y'|_B = Y$.

Proof. Assume (X, Y) is an explicit counterexample for Π. Hence, by definition, (i) $Y \in AS(P \cup X)$ and (ii) there is no Y' with $Y'|_B = Y|_B$ such that $Y' \in AS(Q \cup X)$. From (i), we immediately get that $Y|_B \in AS(P \cup X)|_B$ and (ii) implies that $Y|_B$ itself is not an answer set of $Q \cup X$, hence $Y|_B \notin AS(Q \cup X)|_B$. But then, $(X, Y|_B)$ is a projective counterexample for Π by definition.

For the second part of the lemma, assume that (X, Y) is a projective counterexample for Π. Therefore, (i) $Y \in AS(P \cup X)|_B$ but (ii) $Y \notin AS(Q \cup X)|_B$. Condition (i) implies the existence of a $Y' \in AS(P \cup X)$ with $Y'|_B = Y$. It follows from Condition (ii) that no Z with $Z|_B = Y$ is contained in $AS(Q \cup X)$, and hence, no Z with $Z|_B = Y'|_B$ is contained in $AS(Q \cup X)$. Thus, (X, Y') is an explicit counterexample for Π. □

Theorem 1 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP over $\mathcal{U}$. Then, the following conditions are equivalent:*

(i) *Π does not hold;*

(ii) *Π has an explicit counterexample; and*

(iii) *Π has a projective counterexample.*

Proof. We show that (i) implies (ii), (ii) implies (iii), and (iii) implies (i). Assume that Condition (i) holds, i.e., Π does not hold. Thus, there exists an $X \in \mathcal{F}_A$ such that

$$AS(P \cup X) \subseteq_B AS(Q \cup X)$$

does not hold, i.e., there exists a set Y such that $Y \in AS(P \cup X)$ but there exists no Y' such that $Y'|_B = Y|_B$ and $Y' \in AS(Q \cup X)$. Then, by definition, (X, Y) is an explicit counterexample for Π, thus (i) implies (ii).

That Condition (iii) follows from Condition (ii) is immediately established by Lemma 1. It remains to show that (i) follows from (iii). Assume that Π holds. Hence, for each $X \in \mathcal{F}_A$,

$$AS(P \cup X) \subseteq_B AS(Q \cup X)\,.$$

It follows that for each $Y \subseteq \mathcal{U}$ and for each $X \in \mathcal{F}_A$ it holds that if $Y \in AS(P \cup X)$, then there exists a Y' such that $Y' =_B Y$ and $Y' \in AS(Q \cup X)$. Thus, for any $X \in \mathcal{F}_A$ and any $Y \subseteq \mathcal{U}$, (X, Y) is not an explicit counterexample for Π. By Lemma 1, there cannot be a projective counterexample for Π as well. Hence, Condition (iii) implies Condition (i). □

Example 3 Consider P and Q from Example 2. For $A = \{a, b, c\}$ and $B = \{a, b\}$, the PQIP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ does not hold. This is witnessed by $(\{b, c\}, \{a, b, c\})$ which is the unique explicit counterexample (over $\{a, b, c\}$) for Π. The corresponding projective counterexample for Π is $(\{b, c\}, \{a, b\})$. ◊

As far as PQEPs are concerned, we introduce the following notation:

Definition 3 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$*,*

$$\Pi^{\rightarrow} = (P, Q, \mathcal{F}_A, \subseteq_B) \text{ and}$$
$$\Pi^{\leftarrow} = (Q, P, \mathcal{F}_A, \subseteq_B)$$

are the PQIPs associated with Π.

Obviously, a PQEP Π holds iff both $\Pi^{\rightarrow}$ and $\Pi^{\leftarrow}$ hold. We extend Definition 2 straightforwardly to PQEPs.

Definition 4 *A pair* (X, Y) *is an* explicit counterexample *for a PQEP* Π *if* (X, Y) *is an explicit counterexample for* $\Pi^{\rightarrow}$ *or* $\Pi^{\leftarrow}$.

Likewise, (X, Y) *is a* projective counterexample *for a PQEP* Π *if* (X, Y) *is a projective counterexample for* $\Pi^{\rightarrow}$ *or* $\Pi^{\leftarrow}$.

Next, we introduce the novel concept of an A-B*-wedge* for programs over $\mathcal{U}$, where $A, B \subseteq \mathcal{U}$. A-B-wedges decide problems of form $(P, Q, \mathcal{F}_A, \odot_B)$, for $\odot \in \{\subseteq, =\}$, in such a way that they can be computed *separately* for P and Q. In particular, A-B-wedges for a program P collect the projected answers sets for all possible extensions of P.

Definition 5 *Let* $A, B \subseteq \mathcal{U}$ *be two sets of atoms, and* $P \in \mathcal{P}_{\mathcal{U}}$ *a DLP. A pair* (X, Y) *of interpretations with* $X, Y \subseteq \mathcal{U}$ *is an* A-B-wedge (over $\mathcal{U}$) *of* P *iff*

(i) $X \subseteq A$ *and*

(ii) $Y \in AS(P \cup X)|_B$.

The set of all A-B-wedges of P is denoted by $\omega_{A,B}(P)$.

Lemma 2 *A pair (X, Y) is a projective counterexample for a PQIP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ iff*

(i) $(X, Y) \in \omega_{A,B}(P)$ *and*

(ii) $(X, Y) \notin \omega_{A,B}(Q)$.

Proof. ($\Rightarrow$) Assume that (X, Y) is a projective counterexample for Π. Hence, $X \subseteq A$, $Y \in AS(P \cup X)|_B$, and $Y \notin AS(Q \cup X)|_B$ which implies that (X, Y) is an A-B-wedge of P but not of Q.

($\Leftarrow$) Assume that $(X, Y) \in \omega_{A,B}(P) \setminus \omega_{A,B}(Q)$. Thus, (X, Y) is an A-B-wedge of P but not of Q. Then, by the definition of A-B-wedges, $X \subseteq A$, $Y \in AS(P \cup X)|_B$, and $Y \notin AS(Q \cup X)|_B$. From this follows that (X, Y) is a projective counterexample for Π. □

Now, the following central property is easily shown:

Theorem 2 *Let $P, Q \in \mathcal{P}_\mathcal{U}$ be two programs and $A, B \subseteq \mathcal{U}$ two sets. Then,*

(i) *the PQIP $(P, Q, \mathcal{F}_A, \subseteq_B)$ holds iff $\omega_{A,B}(P) \subseteq \omega_{A,B}(Q)$, and*

(ii) *the PQEP $(P, Q, \mathcal{F}_A, =_B)$ holds iff $\omega_{A,B}(P) = \omega_{A,B}(Q)$.*

Proof. First, we show Part (i). By Theorem 1, a PQIP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ holds iff there exists no projective counterexample for Π. Hence, by Lemma 2, Π holds iff there exists no pair $(X, Y) \in \omega_{A,B}(P) \setminus \omega_{A,B}(Q)$ which holds iff $\omega_{A,B}(P) \subseteq \omega_{A,B}(Q)$.

As for showing Part (ii), consider the PQEP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ and the associated PQIPs $\Pi^{\leftarrow}$ and $\Pi^{\rightarrow}$. The PQEP Π holds iff $\Pi^{\leftarrow}$ and $\Pi^{\rightarrow}$ jointly hold. As shown above, this is the case iff $\omega_{A,B}(P) \subseteq \omega_{A,B}(Q)$ and $\omega_{A,B}(Q) \subseteq \omega_{A,B}(P)$, i.e., iff $\omega_{A,B}(P) = \omega_{A,B}(Q)$. □

Example 4 Reconsider the programs P and Q from Example 2. First, consider $A = B = \{a, b\}$. One can verify that $\omega_{A,B}(P) = \omega_{A,B}(Q) = \mathcal{S}$, where

$$\mathcal{S} = \{(\emptyset, \{a\}), (\emptyset, \{b\}), (\{a\}, \{a\}), (\{b\}, \{b\}), (\{a, b\}, \{a, b\})\}\,.$$

Hence, the PQEP $(P, Q, \mathcal{F}_A, =_B)$ holds.

Second, consider the PQEP $(P, Q, \mathcal{F}_{A'}, =_B)$ with $A' = A \cup \{c\}$. We now obtain

$$\omega_{A',B}(P) = \mathcal{S} \cup \{(\{c\}, \{a\}), (\{a, c\}, \{a\}), (\{b, c\}, \{a, b\}), (\{a, b, c\}, \{a, b\})\},$$
$$\omega_{A',B}(Q) = \mathcal{S} \cup \{(\{c\}, \{a\}), (\{c\}, \{b\}), (\{a, c\}, \{a\}), (\{b, c\}, \{b\}), (\{a, b, c\}, \{a, b\})\}.$$

By Theorem 2, the PQEP $(P, Q, \mathcal{F}_{A'}, =_B)$ does not hold. In fact, all projective counterexamples are given by the symmetric difference

$$\omega_{A',B}(P) \triangle \omega_{A',B}(Q) = \{(\{c\}, \{b\}), (\{b, c\}, \{b\}), (\{b, c\}, \{a, b\})\}$$

and the corresponding explicit counterexamples are $(\{c\}, \{b, c\})$, $(\{b, c\}, \{b, c\})$, and $(\{b, c\}, \{a, b, c\})$. ◊

3.2 Model-Theoretic Characterisations

Next, we deal with semantic characterisations for explicit counterexamples and A-B-wedges in the style of UE-models and A-UE-models. Recall that UE-models and A-UE-models have been introduced to capture uniform equivalence and uniform equivalence relative to A, respectively.

First, let us give a more direct characterisation of UE-models, resp., A-UE-models, than in Chapter 2 by direct tests over program reducts:

Proposition 5 (Eiter and Fink [25]) *A pair (X, Y) is a UE-model of a program P iff*

(i) $X \subseteq Y$, $Y \models P$, $X \models P^Y$, *and*

(ii) *for each X' with $X \subset X' \subset Y$, $X' \not\models P^Y$.*

Note that Condition (i) expresses that (X, Y) is an SE-model of P, while Condition (ii) expresses, in a sense, a maximality condition regarding X and the property that $X \models P^Y$.

Proposition 6 (Woltran [87]) *A pair (X, Y) is an A-UE-model of P iff*

(i) $X \subseteq Y$, $Y \models P$,

(ii) *for each $X' \subset Y$ with $X|_A \subset X'|_A$ or $X'|_A = Y|_A$, $X' \not\models P^Y$, and*

(iii) $X \subset Y$ *implies the existence of an $X' \subseteq Y$ with $X'|_A = X|_A$ and $X' \models P^Y$.*

Now, let us introduce an characterisation of explicit counterexamples in the style of the model-theoretic characterisations of UE-models and A-UE-models as given by Propositions 5 and 6.

Theorem 3 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP over $\mathcal{U}$ and consider $X, Y \subseteq \mathcal{U}$. Then, (X, Y) is an explicit counterexample over $\mathcal{U}$ for Π iff*

(i) $Y \models P$ *and* $X \subseteq Y|_A$,

(ii) *for each Y' with $X \subseteq Y' \subset Y$, $Y' \not\models P^Y$, and*

(iii) *for each Z with $X \subseteq Z$, $Z|_B = Y|_B$, and $Z \models Q$, there is some Z' with $X \subseteq Z' \subset Z$ such that $Z' \models Q^Z$.*

Proof. ($\Rightarrow$) Assume that (X, Y) is an explicit counterexample for Π Hence, it holds that (1) $X \subseteq A$, (2) $Y \in AS(P \cup X)$, and (3) for each Z, $Z =_B Y$ implies $Z \notin AS(Q \cup X)$. According to Proposition 1, Condition (2) is equivalent to $Y \models P$, $X \subseteq Y$, and for each Y', if $X \subseteq Y' \subset Y$, then $Y' \not\models P^Y$. But then, since Condition (1) together with $X \subseteq Y$ is equivalent to $X \subseteq Y|_A$, Items (i) and (ii) hold. It remains to show that Condition (3) implies Item (iii). In fact, we show that Condition (3) is equivalent to Item (iii). Again, we apply Proposition 1: $Z \notin AS(Q \cup X)$ from Condition (3) holds iff, whenever $Z \models P$ and $X \subseteq Z$, there is some Z' with $X \subseteq Z' \subset Z$ and $Z' \models Q^Z$. Hence, Condition (3) holds iff, for each Z with $X \subseteq Z$, $Z|_B = Y|_B$, and $Z \models Q$, there is some Z' with $X \subseteq Z' \subset Z$ such that $Z' \models Q^Z$.

($\Leftarrow$) Assume that Items (i) to (iii) hold. We show that (X, Y) is an explicit counterexample for Π, i.e., Conditions (1) to (3) from above hold. If $X \subseteq Y|_A$ from Item (i) holds, then both $X \subseteq Y$ and $X \subseteq A$. Hence, Condition (1) holds. Moreover, $X \subseteq Y$ together with Item (ii) imply Condition (2) by virtue of Proposition 1. We showed already the equivalence of Condition (3) and Item (iii), thus also Item (iii) holds which concludes the proof. □

Next, we characterise A-B-wedges.

Theorem 4 *A pair (X, Y) is an A-B-wedge of a program P iff*

(i) $X \subseteq A$ *and*

(ii) *there is a Y' with $X \subseteq Y'$ and $Y = Y'|_B$ such that $Y' \models P$ and, for each X' with $X \subseteq X' \subset Y'$, $X' \not\models P^{Y'}$.*

Proof. According to Definition 5, (X, Y) is an A-B-wedge of P iff $X \subseteq A$ and $Y \in AS(P \cup X)|_B$. It thus remains to show that the latter condition is equivalent to (ii). Now, $Y \in AS(P \cup X)|_B$ iff there is some Y' with $Y = Y'|_B$ and $Y' \in AS(P \cup X)$. By Proposition 1, the latter is equivalent to $Y' \models P$, $X \subseteq Y'$, and, for each X' with $X \subseteq X' \subset Y'$, $X' \not\models P^{Y'}$. Thus, we showed that (X, Y) is an A-B-wedge of P iff Conditions (i) and (ii) hold. □

Since uniform equivalence between programs over $\mathcal{U}$ is captured by PQEPs over $\mathcal{U}$ of form $(P, Q, \mathcal{F}_\mathcal{U}, =_\mathcal{U})$, let us now describe the relation between UE-models and A-B-wedges with $A = B = \mathcal{U}$.

First of all, a pair (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of some program only if $X \subseteq Y$. Now, for a program P, (Y, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P iff $Y \models P$. Furthermore, for $X \subset Y$, (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P iff $Y \models P$ and, for all X' with $X \subseteq X' \subset Y$, $X' \not\models P^Y$ holds. So, there is only a subtle difference between $\mathcal{U}$-$\mathcal{U}$-wedges and UE-models laid down in detail by the next result.

Theorem 5 *Let $X \subseteq Y \subseteq \mathcal{U}$ and $P \in \mathcal{P}_\mathcal{U}$. Then:*

(i) *(Y, Y) is a UE-model of P iff (Y, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P. Moreover, if (Y, Y) is a UE-model of P but no (X, Y) with $X \subset Y$ is a UE-model of P (i.e., Y is an answer set of P), then, for all $X \subseteq Y$, (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P.*

(ii) *If (X, Y) is a UE-model of P and $X \subset Y$, then (X', Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for any $X \subset X' \subseteq Y$.*

(iii) *If (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P and $(\emptyset, Y)$ is not a $\mathcal{U}$-$\mathcal{U}$-wedge of P, then there exists a UE-model (X', Y) of P with $X' \subset X$.*

Proof. We start with Item (i). By definition of a UE-model, (Y, Y) is a UE-model of P iff $Y \models P$. By definition of $\mathcal{U}$-$\mathcal{U}$-wedges, (Y, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P iff $Y \models P$. Thus, (Y, Y) is a UE-model of P iff (Y, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge of P. Furthermore, Item (i) states that for any answer set Y of P, each pair (X, Y) with $X \subseteq Y$ is a $\mathcal{U}$-$\mathcal{U}$-wedge of P. So, assume that Y is an answer set of P. We show that $Y \models P$ and, for each X' with $X \subseteq X' \subset Y$, $X' \not\models P^Y$. Since Y is an answer

set of P, by definition, $Y \models P$ and, for each $Y' \subset Y$, $Y' \not\models P^Y$. The latter implies that, for each X' with $X \subseteq X' \subset Y$, $X' \not\models P^Y$. Thus, (X, Y) is $\mathcal{U}$-$\mathcal{U}$-wedge for P.

We proceed with Item (ii). Consider a UE-model (X, Y) with $X \subset Y$. Hence, $Y \models P$, $X \models P^Y$, and, for each Z with $X \subset Z \subset Y$, $Z \not\models P^Y$. From the latter it follows that, for each X' with $X \subset X' \subset Y$, (X', Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for P. Moreover, (Y, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for P since $Y \models P$. Thus, if (X, Y) is a UE-model and $X \subset Y$, then (X', Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for any X' with $X \subset X' \subseteq Y$.

We conclude with Item (iii). Assume that (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for P and $(\emptyset, Y)$ is not a $\mathcal{U}$-$\mathcal{U}$-wedge of P. Since (X, Y) is a $\mathcal{U}$-$\mathcal{U}$-wedge for P, it follows that $Y \models P$, hence (Y, Y) is a UE-model of P. As $(\emptyset, Y)$ is not a $\mathcal{U}$-$\mathcal{U}$-wedge of P, it follows that Y is not an answer set of P. Since $Y \models P$, there has to exist at least one $X' \subset Y$ such that $X' \models P^Y$. If we take a maximal such X', we obtain a UE-model (X', Y) of P. □

Example 5 Consider the programs

$$P = \{a \vee b \leftarrow\} \quad \text{and} \quad Q = \left\{ \begin{array}{l} a \leftarrow \text{not } b\,, \\ b \leftarrow \text{not } a \end{array} \right\},$$

which are uniformly equivalent. The UE-models of P and Q are

$$(\{a\}, \{a\}), (\{b\}, \{b\}), (\{a\}, \{a, b\}), (\{b\}, \{a, b\}), (\{a, b\}, \{a, b\}),$$

but the $\mathcal{U}$-$\mathcal{U}$-wedges of the two programs are

$$(\emptyset, \{a\}), (\{a\}, \{a\}), (\emptyset, \{b\}), (\{b\}, \{b\}), (\{a, b\}, \{a, b\}).$$

◊

While UE-models have been defined with the aim to select a subset of SE-models (which characterise strong equivalence), wedges are not designed in this respect. Rather, they have a much closer relation to projective counterexamples.

3.3 Relations Between Correspondence Problems

Next, we investigate the interplay between different notions of program correspondence. More concretely, we will study the relations between different instantiations of PQIPs and PQEPs with respect to the specified context class and the comparison relation.

The first result states in effect an anti-monotonicity property for PQIPs and PQEPs with respect to their context classes and comparison relations.

Theorem 6 *For any problem $\Pi = (P, Q, \mathcal{F}_A, \odot_B)$, for $\odot \in \{\subseteq, =\}$, whenever Π holds, then $(P, Q, \mathcal{F}_{A'}, \odot_{B'})$ holds as well, for any $A' \subseteq A$ and $B' \subseteq B$.*

Proof. We first prove the result for PQIPs. Assume that $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ holds. We show that then $\Pi' = (P, Q, \mathcal{F}_{A'}, \subseteq_{B'})$ holds as well, for $A' \subseteq A$, $B' \subseteq B$. Clearly, $A' \subseteq A$ implies that the

context class $\mathcal{F}_{A'}$ is a subset of $\mathcal{F}_A$. From $B' \subseteq B$, it follows that $\mathcal{S} \subseteq_B \mathcal{S}'$ implies $\mathcal{S} \subseteq_{B'} \mathcal{S}'$, for all sets $\mathcal{S}, \mathcal{S}'$ of interpretations. By assumption, for any $R \in \mathcal{F}_A$, $AS(P \cup R) \subseteq_B AS(Q \cup R)$. Since $\mathcal{F}_{A'} \subseteq \mathcal{F}_A$, it follows that for any $R \in \mathcal{F}_{A'}$, $AS(P \cup R) \subseteq_B AS(Q \cup R)$. Moreover, since $B' \subseteq B$, for any $R \in \mathcal{F}_{A'}$, we have that $AS(P \cup R) \subseteq_{B'} AS(Q \cup R)$, thus $(P, Q, \mathcal{F}_{A'}, \subseteq_{B'})$ holds as well.

The result for PQIPs extends to PQEPs as follows. Assume the PQEP $\Pi = (P, Q, \mathcal{F}_A, =_B)$ holds. Thus, the associated PQIPs $\Pi^{\leftarrow}$ and $\Pi^{\rightarrow}$ hold as well. Since we have shown the result for PQIPs already, it follows that $\Pi'^{\leftarrow} = (Q, P, \mathcal{F}_{A'}, \subseteq_{B'})$ and $\Pi'^{\rightarrow} = (P, Q, \mathcal{F}_{A'}, \subseteq_{B'})$ hold as well, thus $\Pi' = (P, Q, \mathcal{F}_{A'}, =_{B'})$ holds. □

Lemma 3 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP. For any $V \subseteq \mathcal{U}$ with $At(P \cup Q) \cap V = \emptyset$, if Π holds, then $(P, Q, \mathcal{F}_A, \subseteq_{B \cup V})$ holds.*

Proof. Assume that problem Π holds. Thus, for each $X \in \mathcal{F}_A$, $Y \in AS(P \cup X)$ implies that $Y' \in AS(Q \cup X)$, where $Y' =_B Y$. Note that, by Proposition 1, $Y \in AS(P \cup X)$ implies that $X \subset Y$. Moreover, as no atom from V occurs in P, $Y|_V = X|_V$. Likewise, $Y' \in AS(Q \cup X)$ implies that $X \subseteq Y$. Since no atom in V occurs in Q, $Y'|_V = X|_V$ follows. It remains to show that $Y' =_{B \cup V} Y$. Observe that

$$Y'|_{B \cup V} \;=\; Y'|_B \cup Y'|_V \;=\; Y|_B \cup X|_V \;=\; Y|_B \cup Y|_V \;=\; Y|_{B \cup V}\,.$$

We showed that for each $X \in \mathcal{F}_A$, $Y \in AS(P \cup X)$ implies that $Y' \in AS(Q \cup X)$, where $Y' =_{B \cup V} Y$. Thus, $(P, Q, \mathcal{F}_A, \subseteq_{B \cup V})$ holds. □

Lemma 4 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP. For any $V \subseteq \mathcal{U}$ with $At(P \cup Q) \cap V = \emptyset$, if Π holds, then $(P, Q, \mathcal{F}_{A \cup V}, \subseteq_B)$ holds.*

Proof. Assume that Π holds. Thus, for each $X \in \mathcal{F}_A$, $Y \in AS(P \cup X)$ implies $Y' \in AS(Q \cup X)$, for some Y' with $Y' \mid =_B Y$. We show that $(P, Q, \mathcal{F}_{A \cup V}, \subseteq_B)$ holds as well. Take an arbitrary $X \in \mathcal{F}_{A \cup V}$ and assume that $Y \in AS(P \cup X)$ holds. We show that $Y' \in AS(Q \cup X)$, for some $Y'|_B = Y|_B$. Define $X' = X \setminus (V \setminus A)$ and $Z = Y \setminus (V \setminus A)$. Since $Y \in AS(P \cup X)$, Proposition 1 implies that $X \subseteq Y$. As no atom in V occurs in P, we get that $Y|_V = X|_V$. Hence, $Z \in AS(P \cup X')$. Since $X' \in \mathcal{F}_A$ and $Z \in AS(P \cup X')$, it follows by assumption that $Z' \in AS(Q \cup X')$, for some Z' with $Z'|_B = Z|_B$. The latter and the fact that no atom in $V \setminus A$ occurs in Q implies that $Y' \in AS(Q \cup X)$, where $Y' = Z' \cup (V \setminus A)$. It remains to show that $Y'|_B = Y|_B$. Observe that

$$Y'|_B \;=\; Z'|_B \cup (V \setminus A)|_B \;=\; Z|_B \cup (V \setminus A)|_B \;=\; (Y \setminus (V \setminus A))|_B \cup (V \setminus A)|_B \;=\; Y|_B\,.$$

We showed that for each $X \in \mathcal{F}_{A \cup V}$, $Y \in AS(P \cup X)$ implies that $Y' \in AS(Q \cup X)$, for some Y' with $Y' =_B Y$. Thus, $(P, Q, \mathcal{F}_{A \cup V}, \subseteq_B)$ holds. □

Theorem 7 *For every problem $\Pi = (P, Q, \mathcal{F}_A, \odot_B)$, $\odot \in \{\subseteq, =\}$, and all sets V and W such that $At(P \cup Q) \subseteq V \subseteq \mathcal{U}$ and $At(P \cup Q) \subseteq W \subseteq \mathcal{U}$, Π holds iff $(P, Q, \mathcal{F}_{A'}, \odot_{B'})$ holds, where $A' = A|_V$ and $B' = B|_W$.*

Proof. The only-if direction follows directly from Theorem 6. The if direction follows from Lemmata 3 and 4 since $A = A' \cup (A \setminus V)$, $(A \setminus V) \cap At(P \cup Q) = \emptyset$, $B = B' \cup (B \setminus V)$, and $(B \setminus V) \cap At(P \cup Q) = \emptyset$. □

The significance of the above theorem is that, for any problem $(P, Q, \mathcal{F}_A, \odot_B)$, $\odot \in \{\subseteq, =\}$, we can always safely assume that both A and B contain only atoms that occur in program P or program Q.

Eiter, Tompits, and Woltran [30] showed an invariance result for correspondence problems based on strong equivalence: Let Π be an equivalence problem $(P, Q, \mathcal{P}_\mathcal{U}, =_B)$ and $B \subseteq \mathcal{U}$ an arbitrary set of atoms, then Π holds iff P and Q are strongly equivalent. More generally, $(P, Q, \mathcal{P}_A, \odot_B)$ holds iff $(P, Q, \mathcal{P}_A, \odot_{A \cup B})$ holds. It is to mention that such invariance results fail for PQEPs, as illustrated by the following example.

Example 6 Consider the two programs $P = \{a\}$ and $Q = \emptyset$. Then, the PQEP $(P, Q, \mathcal{F}_\mathcal{U}, =_\emptyset)$ holds. Nevertheless, this does not imply that P and Q are uniformly equivalent, as they are not even equivalent in the ordinary sense. ◊

The next theorem relates PQIPs and PQEPs to special cases of correspondence problems of form $(P, Q, \mathcal{P}_A, \odot_B)$, $\odot \in \{\subseteq, =\}$. It shows that any PQIP, resp., PQEP, can be reduced to ordinary implication, resp., equivalence, with projection.

Theorem 8 *Given* $\Pi = (P, Q, \mathcal{F}_A, \odot_B)$*, for* $\odot \in \{\subseteq, =\}$*, we have that* Π *holds iff*

$$\Pi' = (P \cup G_A, Q \cup G_A, \{\emptyset\}, \odot_{B'})$$

holds, where

$$\begin{aligned} G_A &= \{a' \vee a'' \leftarrow;\ a \leftarrow a' \mid a \in A\}, \\ B' &= B \cup \{a', a'' \mid a \in A\}, \end{aligned}$$

and all a', a'' *are new atoms.*

Proof. Note that the answer sets of G_A correspond to the subsets of A: indeed, $S \subseteq A$ iff $S \cup S' \cup (A \setminus S)'' \in AS(G_A)$, where the notation S', resp., S'', denotes the set $\{x' \mid x \in S\}$, resp., $\{x'' \mid x \in S\}$. Moreover, the set of disjunctive rules from G_A is the bottom for both P and Q with respect to the splitting set $A' \cup A''$. The non-disjunctive rules in G_A simply enforce that whenever a primed atom is in an answer set of $G_A \cup P$ or $G_A \cup Q$, then also the unprimed atom must be in that answer set.

Now, Π holds iff, for each $X \subseteq A$, $AS(P \cup X) \odot_B AS(Q \cup X)$. The latter holds iff the sets

$$\mathcal{R} = \{Y \cup X' \cup (A \setminus X)'' \mid Y \in AS(P \cup X), X \subseteq A\}$$

and

$$\mathcal{S} = \{Y \cup X' \cup (A \setminus X)'' \mid Y \in AS(Q \cup X), X \subseteq A\}$$

satisfy $\odot_{B'}$. Taking the properties of G_A into account, it follows from Proposition 2 that $\mathcal{R} \odot_{B'} \mathcal{S}$ iff $AS(P \cup G_A) \odot_{B'} AS(Q \cup G_A)$ which is equivalent to the condition that Π' holds. □

Example 7 Consider the two programs from Example 2 and the PQEP $\Pi = (P, Q, \mathcal{F}_A, =_B)$ where $A = B = \{a, b\}$. The programs possess the following answer sets (with projection to B).

$$\begin{aligned} AS(P)|_B &= \{\{a\},\ \{b\}\}; & AS(Q)|_B &= \{\{a, c\},\ \{b\}\}|_B = \{\{a\},\ \{b\}\}; \\ AS(P \cup \{a\})|_B &= \{\{a\}\}; & AS(Q \cup \{a\})|_B &= \{\{a, c\}\}|_B = \{\{a\}\}; \\ AS(P \cup \{b\})|_B &= \{\{b\}\}; & AS(Q \cup \{b\})|_B &= \{\{b\}\}; \\ AS(P \cup \{a, b\})|_B &= \{\{a, b\}\}; & AS(Q \cup \{a, b\})|_B &= \{\{a, b, c\}\}|_B = \{\{a, b\}\}. \end{aligned}$$

Clearly, Π holds. In view of Theorem 8, let us define

$$\begin{aligned} \Pi' &= (P \cup G_A, Q \cup G_A, \{\emptyset\}, \odot_{B'})\,, \text{ where} \\ G_A &= \{a' \vee a'' \leftarrow a; a \leftarrow a'; b' \vee b'' \leftarrow; b \leftarrow b'\}\,, \text{ and} \\ B' &= \{a, b, a', a'', b', b''\}\,. \end{aligned}$$

The (projected) answer sets of $P \cup G_A$ and $Q \cup G_A$ are:

$$\begin{aligned} AS(P \cup G_A)|_{B'} &= \{\{a'', b'', a\},\ \{a'', b'', b\},\ \{a', b'', a\},\ \{a'', b', b\},\ \{a', b', a, b\}\}; \\ AS(Q \cup G_A)|_{B'} &= \{\{a'', b'', a, c\},\ \{a'', b'', b\},\ \{a', b'', a, c\}, \\ &\qquad \{a'', b', b\},\ \{a', b', a, b, c\}\}|_{B'} \\ &= \{\{a'', b'', a\},\ \{a'', b'', b\},\ \{a', b'', a\},\ \{a'', b', b\},\ \{a', b', a, b\}\}. \end{aligned}$$

Indeed, also Π' holds. $\Diamond$

So far, we only considered sets of facts as context class for PQEPs and PQIPs. Of course, it is legitimate to ask whether PQEPs or PQIPs are suitable to decide problems where the context class is defined by sets of *disjunctive facts*, i.e., the set of all rules r of form (2.1), where $n = l$. Denote by D^A the set of disjunctive facts over A. By definition, we have that, for all programs P and Q, all alphabets A and B, and $\odot \in \{\subseteq, =\}$, $(P, Q, \mathcal{F}_A, \odot_B)$ holds whenever $(P, Q, D^A, \odot_B)$ holds.

The converse direction however does not hold in general, as the following example shows.

Example 8 Consider the following two programs

$$P = \left\{ \begin{array}{l} a \leftarrow b\,, \\ b \leftarrow a \end{array} \right\} \quad \text{and} \quad Q = \left\{ \begin{array}{l} b \leftarrow a, c\,, \\ a \leftarrow b, d\,, \\ c \leftarrow a, \text{not } d\,, \\ d \leftarrow b, \text{not } c\,, \\ a \leftarrow c\,, \\ b \leftarrow d \end{array} \right\},$$

and let $A = B = \{a, b\}$. The following relations can be easily checked:

$$\begin{aligned} AS(P)|_A &= \{\{\emptyset\}\}; & AS(Q)|_A &= \{\{\emptyset\}\}; \\ AS(P \cup \{a\})|_A &= \{\{a, b\}\}; & AS(Q \cup \{a\})|_A &= \{\{a, b, c\}\}|_A = \{\{a, b\}\}; \\ AS(P \cup \{b\})|_A &= \{\{a, b\}\}; & AS(Q \cup \{b\})|_A &= \{\{a, b, d\}\}|_A = \{\{a, b\}\}; \\ AS(P \cup \{a, b\})|_A &= \{\{a, b\}\}; & AS(Q \cup \{a, b\})|_A &= \{\{a, b, c\}, \{a, b, d\}\}|_A \\ & & &= \{\{a, b\}\}. \end{aligned}$$

On the other hand, we have

$$AS(P \cup \{a \vee b \leftarrow\})|_A = \{\{a, b\}\} \text{ and } AS(Q \cup \{a \vee b \leftarrow\})|_A = \emptyset.$$

Hence, $(P, Q, D^A, \subseteq_B)$ does not hold, although $(P, Q, \mathcal{F}_A, \subseteq_B)$ holds. $\Diamond$

However, in a setting without projection, i.e., relativised uniform equivalence, disjunctive facts do not play a special role. In our setting, this result can be formulated in case of PQIPs as follows.

Proposition 7 *Problem $(P, Q, D^A, \subseteq_{\mathcal{U}})$ holds iff $(P, Q, \mathcal{F}_A, \subseteq_{\mathcal{U}})$ holds.*

Proof. The only-if direction is by definition. For the if direction, suppose that $(P, Q, D^A, \subseteq_{\mathcal{U}})$ does not hold. We show that $(P, Q, \mathcal{F}_A, \subseteq_{\mathcal{U}})$ does not hold. Since $(P, Q, D^A, \subseteq_{\mathcal{U}})$ does not hold, there is a set D of disjunctive facts over A and an interpretation Y such that $Y \in AS(P \cup D)$ but $Y \notin AS(Q \cup D)$.

We first show the following result. Define a *Y-selection of D* to be any set $F \subseteq Y|_A$ of facts such that, for each $a_1 \vee \cdots \vee a_n \in D$, $\{a_1, \ldots, a_n\} \cap F \neq \emptyset$. Then, $Y \in AS(P \cup D)$ implies $Y \in AS(P \cup F)$, for each Y-selection F of D. Towards a contradiction, suppose there exists a Y-selection F of D such that $Y \notin AS(P \cup F)$. By assumption, $Y \models P \cup D$, and, by definition, $Y \models F$. Thus, $Y \models P \cup F$. It follows that there exists a set $Y' \subset Y$ such that $Y' \models P^Y \cup F$. Again, it is clear that $Y' \models F$ implies $Y' \models D$. But then, $Y' \models P^Y \cup D$. A contradiction to $Y \in AS(P \cup D)$.

To show the assertion, it thus remains to show that there exists a Y-selection F of D such that $Y \notin AS(Q \cup F)$. Clearly, if $Y \not\models Q$ we are done. So suppose $Y \models Q$. We already know that $Y \models D$, hence (by assumption that $Y \notin AS(Q \cup D)$), there exists a $Y' \subset Y$ such that $Y' \models Q^Y \cup D$. Since $Y' \models D$, there exists also a Y-selection F of D such that $Y' \models F$. Hence, $Y' \models Q^Y \cup F$. We obtain that $Y \notin AS(Q \cup F)$. $\square$

We conclude this section by introducing the following theorem which directly relates PQIPs and PQEPs by showing how to compute PQIPs by means of PQEPs. This result slightly generalises a result by Eiter, Tompits, and Woltran [30].

Theorem 9 *The PQIP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ holds iff the PQEP*

$$\Pi' = (L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\}, L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\}, \mathcal{F}_C, =_B)$$

holds, where $A \subseteq C \subseteq A \cup \{g_P, g_Q\}$, g_P and g_Q are new atoms, and

$$L_{P,Q} = \{\leftarrow g_P, g_Q\} \cup \{H \leftarrow g_R, B \mid R \in \{P, Q\}, H \leftarrow B \in R\}.$$

Proof. ($\Rightarrow$) Assume that the PQIP Π holds. We show that the PQIP Π' holds as well. By Proposition 6, we only need to consider $A \cup \{g_P, g_Q\}$ for C. Hence, we show that, for any set $F \subseteq A \cup \{g_P, g_Q\}$,

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B\,.$$

Now, consider an arbitrary $F \subseteq C$. We distinguish four cases for F (recall that g_P and g_Q are new atoms):

(i) $g_P \in F, g_Q \in F$: Since $L_{P,Q}$ contains the rule $\leftarrow g_P, g_Q$,

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F) = AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F) = \emptyset\,.$$

(ii) $g_P \in F, g_Q \notin F$: Then,

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(P \cup F)|_B\,, \text{ resp.,}$$
$$AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B = AS(P \cup F)|_B\,.$$

(iii) $g_P \notin F, g_Q \in F$: Similar to the previous case, we have that

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(Q \cup F)|_B\,, \text{ resp.,}$$
$$AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B = AS(Q \cup F)|_B\,.$$

(iv) $g_P \notin F, g_Q \notin F$: Here,

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(Q \cup F)|_B \text{ and}$$
$$AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B = AS(P \cup F)|_B \cup AS(Q \cup F)|_B\,.$$

By assumption, for each set of facts $F' \subseteq A$,

$$AS(P \cup F')|_B \subseteq AS(Q \cup F')|_B\,.$$

This is equivalent to the condition that for each set of facts $F' \subseteq A$,

$$AS(Q \cup F')|_B = AS(P \cup F')|_B \cup AS(Q \cup F')|_B\,.$$

Then, it holds in particular for F that

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B\,.$$

Thus in all cases, for F, we obtain

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B = AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B\,,$$

and hence Π' holds.

($\Leftarrow$) Assume that the PQIP Π does not hold. Hence, there is a set $F \subseteq A$ and there exists a $Y \in AS(P \cup F)|_B$ but $Y \notin AS(Q \cup F)|_B$. As for the PQEP Π', for any set $F' \subseteq A$,

$$AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F')|_B = AS(Q \cup F')|_B \text{ and}$$
$$AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F')|_B = AS(P \cup F')|_B \cup AS(Q \cup F')|_B\,.$$

Thus, $Y \in AS(L_{P,Q} \cup \{g_Q \vee g_P \leftarrow\} \cup F)|_B$ but $Y \notin AS(L_{P,Q} \cup \{g_Q \leftarrow \text{not } g_P\} \cup F)|_B$, and therefore Π' does not hold. □

Roughly speaking, the above theorem yields two properties: First, it maps PQIPs into PQEPs via two new atoms. Second, it shows that these new atoms can be arbitrarily fixed in the context of the resulting PQEP. These points will have a certain significance for our subsequent complexity results.

3.4 Computational Complexity

We now analyse the complexity of deciding PQIPs and PQEPs. Let us first summarise some results from the literature.

Proposition 8 (Eiter, Tompits, and Woltran [30]) *Given programs $P, Q \in \mathcal{P}_\mathcal{U}$, $A, B \subseteq \mathcal{U}$, and $\odot \in \{\subseteq, =\}$, deciding whether $(P, Q, \mathcal{P}_A, \odot_B)$ holds is Π_4^P-complete. The problem is coNP-complete if $A = \mathcal{U}$.*

Moreover, it can be shown that $(P, Q, \mathcal{P}_\mathcal{U}, \odot_B)$ holds just in case that $(P, Q, \mathcal{P}_\mathcal{U}, \odot_\mathcal{U})$ holds, for $\odot \in \{\subseteq, =\}$. In other words, projection does not play a role when assuming an unrestricted context [30]. The coNP-hardness of the problem of deciding whether $(P, Q, \mathcal{P}_\mathcal{U}, \odot_B)$ holds is thus immediate by known complexity results for strong equivalence (see, e.g., related articles [28, 48, 68]). Indeed, one reduction to show coNP-hardness of strong equivalence (as used by Eiter, Fink, and Woltran in their proof of Theorem 6.17 [28]), also applies to the inclusion problems $(P, Q, \mathcal{P}_\mathcal{U}, \subseteq_\mathcal{U})$. On the other hand, the coNP-membership for the problem of deciding $(P, Q, \mathcal{P}_\mathcal{U}, \subseteq_B)$ follows from the known coNP-membership of deciding $(P, Q, \mathcal{P}_\mathcal{U}, =_B)$ and Theorem 9.[1]

The following result by Eiter, Tompits, and Woltran [30] is in particular relevant for our purpose, since it provides complexity bounds for PQIPs and PQEPs as a special case.

Proposition 9 (Eiter, Tompits, and Woltran [30]) *Given programs $P, Q \in \mathcal{P}_\mathcal{U}$, $B \subseteq \mathcal{U}$, $\odot \in \{\subseteq, =\}$, and $\mathcal{C} \subseteq \mathcal{P}_\mathcal{U}$, where each $R \in \mathcal{C}$ is polynomial in the size of $P \cup Q$, deciding whether $(P, Q, \mathcal{C}, \odot_B)$ holds is Π_3^P-complete.*

Another relevant previous result concerns the complexity of checking relativised uniform equivalence [28, 87], which thus provides us complexity bounds for PQIPs and PQEPs without projection (again, the complexity results for implication problems follow from Theorem 9 and by inspecting the respective hardness proofs for equivalence problems, which turn out to be applicable to implication problems, as well).

Proposition 10 (Woltran [87]) *Given programs $P, Q \in \mathcal{P}_\mathcal{U}$, $A \subseteq \mathcal{U}$, and $\odot \in \{\subseteq, =\}$, deciding whether the problem $(P, Q, \mathcal{F}_A, \odot_\mathcal{U})$ holds is Π_2^P-complete. Hardness holds even for arbitrary but fixed A.*

For general PQIPs and PQEPs, we expect an increase in complexity but, in view of Proposition 9, it cannot be beyond Π_3^P. However, Proposition 9 does not provide details about the hardness of such problems. In fact, Eiter, Tompits, and Woltran [30] report Π_3^P-hardness for ordinary equivalence with projection, i.e., PQEPs of the form $(P, Q, \mathcal{F}_A, =_B)$ with $A = \emptyset$. Our main result below shows that nearly all parameterisations for PQIPs and PQEPs result in a matching lower bound. In particular, we show that Π_3^P-hardness holds even if the context alphabet A is fixed arbitrarily. This is in stark contrast to Proposition 8, which shows that considering programs over A (instead of *sets of facts* over A) remains in coNP, providing $A = \mathcal{U}$.

[1] We would like to thank Chiaki Sakama and Katsumi Inoue for pointing out that these result do not follow immediately from results by Eiter, Tompits, and Woltran [30].

Theorem 10 *Given programs $P, Q \in \mathcal{P}_\mathcal{U}$ and sets $A, B \subseteq \mathcal{U}$ of atoms, deciding whether the PQIP $(P, Q, \mathcal{F}_A, \subseteq_B)$ holds is Π_3^P-complete. Hardness holds even for arbitrary but fixed A.*

Proof. Membership in Π_3^P follows from Proposition 9. We show Π_3^P-hardness by reducing the Π_3^P-hard problem of checking validity of a $(3, \forall)$-QBF in PDNF into PQIPs.

The reduction is as follows: Let $\Phi = \forall Z \exists X \forall Y \phi$ be a QBF of the described form with $\phi = \bigvee_{i=1}^{n} C_i$ being a formula in DNF. Define $\Pi_\Phi = (P_\Phi, Q_\Phi, \mathcal{F}_A, \subseteq_Z)$, where A is an arbitrary set of atoms and P_Φ, Q_Φ are given as follows:

$$\begin{aligned} P_\Phi = {} & \{z \vee \bar{z} \leftarrow ; \leftarrow z, \bar{z} \mid z \in Z\} \cup \\ & \{\leftarrow v ; \leftarrow \bar{v} \mid v \in X \cup Y\}; \\ Q_\Phi = {} & \{v \vee \bar{v} \leftarrow ; \leftarrow v, \bar{v} \mid v \in Z \cup X\} \cup \\ & \{y \vee \bar{y} \leftarrow; y \leftarrow a; \bar{y} \leftarrow a; a \leftarrow y, \bar{y} \mid y \in Y\} \cup \\ & \{a \leftarrow C_i^* \mid 1 \leq i \leq n\} \cup \{a \leftarrow \text{not } a\}. \end{aligned}$$

Here, C^* is a sequence of atoms containing each atom w occurring positively in C, and $\bar{w}$ for each w occurring negatively in C. Moreover, a and all $\bar{v}$'s are new distinct atoms.

We show that Φ is valid iff Π_Φ holds. For the if direction, suppose that Φ is not valid. We show that $(P_\Phi, Q_\Phi, \mathcal{F}_A, \subseteq_Z)$ does not hold, for $A = \emptyset$. By definition, $(P_\Phi, Q_\Phi, \mathcal{F}_A, \subseteq_Z)$ then cannot hold for arbitrary A. Since Φ is not valid, there exists an interpretation $I_Z \subseteq Z$ such that $\exists X \forall Y \phi[I_Z]$ is not valid, where $\phi[I_Z]$ simplifies ϕ with respect to interpretation I_Z. We show that $I_Z \in AS(P_\Phi)|_Z$ but $I_Z \notin AS(Q_\Phi)|_Z$. The former is clear. For the latter, towards a contradiction, suppose that Q_Φ possesses an answer set $K \in AS(Q_\Phi)$ with $K|_Z = I_Z$. By definition of Q_Φ, we further have that for each $z \in Z$, $\bar{z} \in K$ iff $z \notin I_Z$. From this, one can show that the program

$$\begin{aligned} Q_\Phi[I_Z] = {} & \{x \vee \bar{x} \leftarrow ; \leftarrow x, \bar{x} \mid x \in X\} \cup \\ & \{y \vee \bar{y} \leftarrow; y \leftarrow a; \bar{y} \leftarrow a; a \leftarrow y, \bar{y} \mid y \in Y\} \cup \\ & \{a \leftarrow C_i^{I_Z} \mid 1 \leq i \leq n\} \cup \{a \leftarrow \text{not } a\}, \end{aligned}$$

where $C_i^{I_Z}$ is as C_i^* but using the conjuncts from $\phi[I_Z]$ instead of ϕ, has an answer set. However, the latter holds iff $\exists X \forall Y \phi[I_Z]$ is valid. This can be seen by inspecting the original proof for Σ_2^P-hardness of deciding whether a program has at least one answer set [29], which uses essentially the same program as $Q_\Phi[I_Z]$. Since $\exists X \forall Y \phi[I_Z]$ is not valid, however, we get a contradiction.

For the only-if direction, suppose Π_Φ does not hold. We show that Φ is not valid. Since Π_Φ does not hold, by Theorem 1 there is a projective counterexample (J, I) for Π_Φ, i.e., $J \subseteq A$, $I \in AS(P_\Phi \cup J)|_Z$, and $I \notin AS(Q_\Phi \cup J)|_Z$ hold. From $I \in AS(P_\Phi \cup J)|_Z$, we derive the following properties: for any $v \in X \cup Y$, neither v nor $\bar{v}$ is contained in J, and $(J \cap Z) \subseteq I \subseteq Z$. Similar arguments as before yield the following chain of equivalences: $I \notin AS(Q_\Phi \cup J)|_Z$ iff $Q_\Phi[I]$ has no answer set iff $\exists X \forall Y \phi[I]$ is not valid. But then, the QBF $\Phi = \forall Z \exists X \forall Y \phi$ cannot be valid. □

Since a PQEP Π holds iff its associated PQIPs $\Pi^{\rightarrow}$ and $\Pi^{\leftarrow}$ both hold, it follows that the complexity of checking PQEPs is in Π_3^P as well. Moreover, the matching lower bounds for PQEPs

$(P, Q, \mathcal{F}_A, =_B)$ for arbitrary A follow in view of Theorem 9 since it it maps PQIPs into PQEPs via two new atoms and it shows that these new atoms can be arbitrarily fixed in the context of the resulting PQEP. Thus, hardness carries over also for arbitrary but fixed alphabets.

Corollary 1 *Given programs $P, Q \in \mathcal{P}_\mathcal{U}$ and sets $A, B \subseteq \mathcal{U}$ of atoms, deciding whether the PQEP $(P, Q, \mathcal{F}_A, =_B)$ holds is Π_3^P-complete. Hardness holds even for arbitrary but fixed A.*

We observe that thus also the special case when $A = B$, which amounts to notions similar to modular equivalence [65], and database-like settings, where A is a subset of extensional predicates and B is set of intensional query predicates, remain hard for Π_3^P.

CHAPTER 4

Translations into Quantified Propositional Logic

In this section, we discuss issues for computing PQIPs and PQEPs. We adopt a reduction approach here, translating the problems under consideration into problems for which solvers already exist. Naturally, the translations we seek should be constructible in polynomial time.

First of all, we remark that since checking PQIPs and PQEPs is Π_3^P-complete, these tasks cannot be efficiently reduced to (propositional) DLPs under the answer-set semantics, unless the polynomial hierarchy collapses to the second level [29]. Hence, a more expressive language is required. This leads us to quantified propositional logic as a suitable target language, as any decision problem in PSPACE can be efficiently reduced to QBFs [66].

Recall that 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 (Proposition 4). In what follows, we will seek for reductions from correspondence problems to QBFs that are adequate in the sense of Besnard et al. [8], cf. Section 2.4.

Besides the structural complexity of the considered problems that suggests a reduction approach, there are several efficient solvers for QBFs available which can be used as back-end inference engines for solving the encoded problems.

In fact, a similar reduction approach to QBFs was already adopted for realising a first version of the system cc$\top$ [55], which allows to verify the kinds of correspondence problems studied by Eiter, Tompits, and Woltran [30]. In principle, we could use cc$\top$ as such to verify PQIPs and PQEPs, because the latter problems can be reduced to the former, in view of Theorem 8. However, verifying PQIPs and PQEPs that way would involve two reduction steps, as cc$\top$ relies itself on a reduction to QBFs. The direct encodings described next avoid this.

4.1 Ancillary Modules

In what follows, we make use of sets of globally new atoms in order to refer to different assignments for 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 $i \geq 1$. In fact, we make use of such superscripts as a general renaming schema for formulas and rules as well. 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 (2.1), we also 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 $\top$.

The following construct will be central for our purposes.

Definition 6 *Let P be a program and $i, j \geq 1$. Then,*

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

The following property holds:

Theorem 11 *Let P be a program with $At(P) = V$, 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, $I \models P^{\langle i,j \rangle}$ iff $X \models P^Y$.*

Proof. By definition of a program reduct and of a model of a program, $X \models P^Y$ holds iff for each rule $r \in P$ of form (2.1), whenever (i) $\{a_{l+1}, \ldots, a_m\} \subseteq X$ and (ii) $Y \cap \{a_{m+1}, \ldots, a_n\} = \emptyset$, then (iii) $X \cap \{a_1, \ldots, a_l\} \neq \emptyset$. Furthermore, as $I|_{V^i} = X^i$ and $I|_{V^j} = Y^j$ by assumption, for each rule $r \in P$ of form (2.1), (i) holds iff $I \models B^+(r^i)$, (ii) holds iff $I \not\models a^j_{m+1} \vee \cdots \vee a^j_n$, which in turn holds iff $I \models B^-(r^j)$, and (iii) holds iff $I \models H(r^i)$. Putting all together, $X \models P^Y$ iff $I \models \bigwedge_{r \in P} ((B^+(r^i) \wedge B^-(r^j)) \rightarrow H(r^i))$. Hence, $X \models P^Y$ iff $I \models P^{\langle i,j \rangle}$. □

Example 9 Consider the program

$$Q = \left\{ \begin{array}{l} a \leftarrow \text{not } b\,, \\ b \leftarrow \text{not } a \end{array} \right\}.$$

Then, for instance,

$$Q^{\langle 1,2 \rangle} = (\neg b^2 \rightarrow a^1) \wedge (\neg a^2 \rightarrow b^1)\,.$$

For any $X^1 \subseteq \{a^1, b^1\}$, the interpretation $\{a^2, b^2\} \cup X^1$ is a model of $Q^{\langle 1,2 \rangle}$, reflecting the fact that any X (over $\{a, b\}$) is a model of the reduct $Q^{\{a,b\}}$. ◊

Next, we introduce operators which allow to compare different subsets of atoms from a common set V under subset inclusion, proper-subset inclusion, and set equality, respectively:

Definition 7 *Let V be a set of atoms and $i, j \geq 1$. Then,*

(i) $(V^i \leq V^j) = \bigwedge_{v \in V} (v^i \rightarrow v^j)$,

(ii) $(V^i < V^j) = (V^i \leq V^j) \wedge \neg(V^j \leq V^i)$, *and*

(iii) $(V^i = V^j) = (V^i \leq V^j) \wedge (V^j \leq V^i)$.

Observe that the latter is equivalent to $\bigwedge_{v \in V}(v^i \leftrightarrow v^j)$.

Theorem 12 *Let V be a set of atoms. Given an interpretation I and $X, Y \subseteq V$ such that, for some i, j, $I|_{V^i} = X^i$ and $I|_{V^j} = Y^j$, we have that*

(i) $I \models (V^i \leq V^j)$ *iff* $X \subseteq Y$,

(ii) $I \models (V^i < V^j)$ *iff* $X \subset Y$, *and*

(iii) $I \models (V^i = V^j)$ *iff* $X = Y$.

Proof. We proof only Item (i) since Items (ii) and Item (iii) are immediate consequences.

($\Rightarrow$) Consider interpretation I and sets $X, Y \subseteq V$ such that, for some i, j, $I|_{V^i} = X^i$ and $I|_{V^j} = Y^j$. We show that if $I \models (V^i \leq V^j)$ then $X \subseteq Y$. Suppose that $a \in X$. Then $a^i \in X^i$, and since $I|_{V^i} = X^i$, it follows that $I \models a^i$. Since $I \models V^i \leq V^j$ by assumption, $I \models a^j$ follows. But then $a^j \in Y^j$ because $I|_{V^j} = Y^j$. Hence, $a \in Y$. Consequently, $X \subseteq Y$.

($\Leftarrow$) We show that if $X \subseteq Y$ then $I \models (V^i \leq V^j)$. Take an arbitrary $a^i \in V^i$. We distinguish two cases: either $I \models a^i$ or $I \not\models a^i$. In the first case, $I|_{V^i} = X$ implies $a^i \in X^i$. Hence, $a \in X$. Then, $a \in Y$ by assumption, and $a^j \in Y^j$. Since $I|_{V^j} = Y^j$, $I \models a^j$ follows. Thus, $I \models a^i \rightarrow a^j$. In the second case, we immediately get that $I \models a^i \rightarrow a^j$. We showed that $I \models v^i \rightarrow v^j$ for all $v \in V$, hence $I \models V^i \leq V^j$. □

4.2 Main Translations

With the above building blocks at hand, we proceed with our central encoding.

Definition 8 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, and $A, B \subseteq V$. Then,*

$$\Phi_\Pi = P^{\langle 1,1 \rangle} \wedge (A^2 \leq A^1)\,;$$
$$\Phi'_\Pi = \forall V^3 \Big(((A^2 \leq A^3) \wedge (V^3 < V^1)) \rightarrow \neg P^{\langle 3,1 \rangle} \Big)\,;$$
$$\Phi''_\Pi = \forall V^4 \Big((B^4 = B^1) \rightarrow \Psi_\Pi \Big) \text{ where}$$
$$\Psi_\Pi = ((Q^{\langle 4,4 \rangle} \wedge (A^2 \leq A^4)) \rightarrow \exists V^5 (((A^2 \leq A^5) \wedge (V^5 < V^4)) \wedge Q^{\langle 5,4 \rangle})) \text{ and}$$
$$\mathsf{S}[\Pi] = \Phi_\Pi \wedge \Phi'_\Pi \wedge \Phi''_\Pi\,.$$

Observe that the free variables of $\mathsf{S}[\Pi]$ are given by $V^1 \cup A^2$. For models of $\mathsf{S}[\Pi]$, the assignments to $V^1 \cup A^2$ yield the explicit counterexamples for Π. More specifically, $\mathsf{S}[\Pi]$ expresses the conditions of Theorem 3, where assignments for V^1, A^2, V^3, V^4, and V^5 correspond to Y, X, Y', Z, and Z', respectively. Taking the semantics of the introduced building blocks into account, $Y^1 \cup X^2 \models \Phi_\Pi$ iff X and Y satisfy the first item of Theorem 3, $Y^1 \cup X^2 \models \Phi'_\Pi$ iff X and Y satisfy the second item of Theorem 3, and $Y^1 \cup X^2 \models \Phi''_\Pi$ iff X and Y satisfy the third item of Theorem 3. Formally, we obtain the following result:

Lemma 5 *Let* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *be a PQIP,* $At(P \cup Q) = V$*,* $A, B \subseteq V$*,* $X \subseteq A$*, and* $Y \subseteq V$*. Then,* (X, Y) *is an explicit counterexample for* Π *iff* $Y^1 \cup X^2 \models \mathsf{S}[\Pi]$*.*

Proof. We show that $Y^1 \cup X^2 \models \mathsf{S}[\Pi]$ iff (X, Y) is an explicit counterexample for Π, i.e., Items (i)–(iii) of Theorem 3 hold. More precisely, we show that

1. $Y^1 \cup X^2 \models \Phi_\Pi$ iff Item (i) holds,
2. $Y^1 \cup X^2 \models \Phi'_\Pi$ iff Item (ii) holds, and
3. $Y^1 \cup X^2 \models \Phi''_\Pi$ iff Item (iii) holds.

We start to prove Condition 1. By definition of Φ_Π, $Y^1 \cup X^2 \models \Phi_\Pi$ iff $Y^1 \cup X^2 \models P^{\langle 1,1 \rangle}$ and $Y^1 \cup X^2 \models A^2 \leq A^1$. By Theorem 11, since $(Y^1 \cup X^2)|_{Y^1} = Y^1$, the former holds iff $Y \models P^Y$, which in turn holds iff $Y \models P$. In view of Theorem 12, since $(Y^1 \cup X^2)|_{Y^1} = Y^1$, $(Y^1 \cup X^2)|_{X^2} = X^2$, and $X \subseteq A$ by assumption, the latter holds iff $X \subseteq Y|_A$.

Next, we prove Condition 2. We have that $Y^1 \cup X^2 \models \Phi'_\Pi$ iff for each $Y' \subseteq V$,

$$Y^1 \cup X^2 \cup Y'^3 \models ((A^2 \leq A^3) \wedge (V^3 < V^1)) \rightarrow \neg P^{\langle 3,1 \rangle}\,.$$

The latter holds iff, for each $Y' \subseteq V$, whenever (a) $Y^1 \cup X^2 \cup Y'^3 \models A^2 \leq A^3$ and (b) $Y^1 \cup X^2 \cup Y'^3 \models V^3 < V^1$ then (c) $Y^1 \cup X^2 \cup Y'^3 \not\models P^{\langle 3,1 \rangle}$. Since, by Theorem 12, (a) holds iff $X \subseteq Y|_A$, which in turn holds iff $X \subseteq Y$ (since $X \subseteq A$), (b) holds iff $Y \subset Y'$, and, by Theorem 11, (c) holds iff $Y' \not\models P^Y$, it follows that Condition 2 holds iff, for each $Y' \subseteq V$, $X \subseteq Y' \subseteq Y$ implies that $Y' \not\models P^Y$, i.e., iff Item (ii) of Theorem 3 holds.

Finally, we prove Condition 3. To begin with, $Y^1 \cup X^2 \models \Phi''_\Pi$ iff, for each $Z \subseteq V$, if $Y^1 \cup X^2 \cup Z^4 \models B^4 = B^1$, $Y^1 \cup X^2 \cup Z^4 \models Q^{\langle 4,4 \rangle}$, and $Y^1 \cup X^2 \cup Z^4 \models A^2 \leq A^4$, then there exists a $Z' \subseteq V$ with $Y^1 \cup X^2 \cup Z^4 \cup Z'^5 \models A^2 \leq A^5$, $Y^1 \cup X^2 \cup Z^4 \cup Z'^5 \models V^5 < V^4$, and $Y^1 \cup X^2 \cup Z^4 \cup Z'^5 \models Q^{\langle 5,4 \rangle}$. Then, in view of Theorem 12, Condition 3 holds iff, for each $Z \subseteq V$, if $Y|_B = Z|_B$, $Z \models Q^Z$, and $X \subseteq Z|_A$, then there exists a $Z' \subseteq V$ with $X \subseteq Z'|_A$, $Z' \subset Z$, and $Z' \models Q^Z$. Recall that $Z \models Q^Z$ iff $Z \models Q$, and $X \subseteq Z'|_A$ iff $X \subseteq Z'$, since $X \subseteq A$. Thus, Condition 3 holds iff Item (iii) of Theorem 3 holds. □

Expressing the task whether a PQIP holds is now a simple matter to realise:

Theorem 13 *For any PQIP* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $At(P \cup Q) = V$*,* Π *holds iff* $\neg\exists V^1 \exists A^2 \mathsf{S}[\Pi]$ *is valid.*

Proof. ($\Rightarrow$) Assume that $\neg\exists V^1 \exists A^2 \mathsf{S}[\Pi]$ is not valid. Hence, there exists a pair (X, Y) with $Y \subseteq V$ and $X \subseteq A$ such that $Y^1 \cup X^2 \models \mathsf{S}[\Pi]$. Then, by Lemma 5, (X, Y) is an explicit counterexample for Π. Hence, in view of Theorem 1, Π does not hold.

($\Leftarrow$) Suppose that $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ does not hold. Therefore, by Theorem 1, there exists an explicit counterexample (X, Y) for Π. It follows by Lemma 5 that $Y^1 \cup X^2 \models \mathsf{S}[\Pi]$. Since $X \subseteq A$ by definition of an explicit counterexample, the closed QBF $\exists V^1 \exists A^2 \mathsf{S}[\Pi]$ is true and, consequently, $\neg\exists V^1 \exists A^2 \mathsf{S}[\Pi]$ is not valid. □

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

Lemma 6 *Let $\Pi = (P, Q, \mathcal{F}_A, =_B)$ be a PQEP, $At(P \cup Q) = V$, $A, B \subseteq V$, $X \subseteq A$, and $Y \subseteq V$. Then, (X, Y) is an explicit counterexample for Π iff $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$.*

Proof. We have the following chain of equivalences: $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$ iff $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\rightarrow}]$ or $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\leftarrow}]$ iff (according to Lemma 5) (X, Y) is an explicit counterexample for $\Pi^{\rightarrow}$ or for $\Pi^{\leftarrow}$. □

Theorem 14 *For any PQEP $\Pi = (P, Q, \mathcal{F}_A, =_B)$ with $At(P \cup Q) = V$, problem Π holds iff $\neg\exists V^1 \exists A^2 (\mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}])$ is valid.*

Proof. ($\Rightarrow$) Assume that $\neg\exists V^1 \exists A^2 (\mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}])$ is not valid. Thus, there exists a pair (X, Y), with $Y \subseteq V$ and $X \subseteq A$, such that $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$. Then, by Lemma 6, (X, Y) is an explicit counterexample for $\Pi^{\rightarrow}$ or $\Pi^{\leftarrow}$, and, by Theorem 1, $\Pi^{\rightarrow}$ or $\Pi^{\leftarrow}$ does not hold. Thus, Π does not hold.

($\Leftarrow$) Assume that $\Pi = (P, Q, \mathcal{F}_A, =_B)$ does not hold. Therefore, there exists an explicit counterexample (X, Y) for $\Pi^{\rightarrow}$ or $\Pi^{\leftarrow}$. Then, by Lemma 6, $Y^1 \cup X^2 \models \mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$, and since $X \subseteq A$ by definition of an explicit counterexample, the closed QBF $\exists V^1 \exists A^2 \mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$ is true. Thus, $\neg\exists V^1 \exists A^2 (\mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}])$ is not valid. □

4.3 Simplifications and Special Cases

In this section, we introduce an alternative QBF encoding that can be regarded as a simplification of $\mathsf{S}[\cdot]$. Then, we discuss how this simplification behaves when used for different special cases of correspondence problems like uniform equivalence or ordinary equivalence.

Alternative Encoding

The benefit of the alternative encoding discussed now 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. [8]) 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 8. 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 9 *Let $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ be a PQIP, $At(P \cup Q) = V$, and $A, B \subseteq V$. Then,*

$$\mathsf{T}[\Pi] = \Phi_\Pi \wedge \Phi'_\Pi \wedge \forall (V^4 \setminus B^4)\, \Psi_\Pi[B^4/B^1]\,,$$

where Φ_Π and Φ'_Π are defined as in Definition 8 and $\Psi_\Pi[B^4/B^1]$ denotes the QBF that results from Ψ_Π when, for each atom $v \in B$, each occurrence of v^4 in Ψ_Π is replaced by v^1.

Next, we illustrate the different outcomes of the two encodings.

Example 10 Consider the PQIP $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ from Example 3 with $A = B = \{a, b\}$. Then,

$$\begin{aligned}\mathsf{S}[\Pi] = \Phi_\Pi \wedge \forall a^4 b^4 c^4 (&(a^4 \leftrightarrow a^1) \wedge (b^4 \leftrightarrow b^1) \rightarrow (((\neg b^4 \rightarrow a^4) \wedge (\neg a^4 \rightarrow b^4) \wedge (a^4 \rightarrow c^4) \wedge \\ &(a^2 \rightarrow a^4) \wedge (b^2 \rightarrow b^4)) \rightarrow \exists a^5 b^5 c^5 ((a^2 \rightarrow a^5) \wedge (b^2 \rightarrow b^5) \wedge \\ &(a^5 \rightarrow a^4) \wedge (b^5 \rightarrow b^4) \wedge (c^5 \rightarrow c^4) \wedge \neg((a^4 \rightarrow a^5) \wedge (b^4 \rightarrow b^5) \wedge \\ &(c^4 \rightarrow c^5)) \wedge (\neg b^4 \rightarrow a^5) \wedge (\neg a^4 \rightarrow b^5) \wedge (a^5 \rightarrow c^5)))),\end{aligned}$$

$$\begin{aligned}\mathsf{T}[\Pi] = \Phi_\Pi \wedge \forall c^4 (&(((\neg b^1 \rightarrow a^1) \wedge (\neg a^1 \rightarrow b^1) \wedge (a^1 \rightarrow c^4) \wedge \\ &(a^2 \rightarrow a^1) \wedge (b^2 \rightarrow b^1)) \rightarrow \exists a^5 b^5 c^5 ((a^2 \rightarrow a^5) \wedge (b^2 \rightarrow b^5) \wedge \\ &(a^5 \rightarrow a^1) \wedge (b^5 \rightarrow b^1) \wedge (c^5 \rightarrow c^4) \wedge \neg((a^1 \rightarrow a^5) \wedge (b^1 \rightarrow b^5) \wedge (c^4 \rightarrow c^5)) \wedge \\ &(\neg b^1 \rightarrow a^5) \wedge (\neg a^1 \rightarrow b^5) \wedge (a^5 \rightarrow c^5))).\end{aligned}$$

$\Diamond$

Theorem 15 *For any PQIP* Π*, the QBFs* $\mathsf{S}[\Pi]$ *and* $\mathsf{T}[\Pi]$ *are equivalent.*

Proof. Observe that for any PQIP Π, $\mathsf{S}[\Pi]$ and $\mathsf{T}[\Pi]$ differ only in the definition of the respective subformula $\phi = \forall V^4((B^4 = B^1) \rightarrow \Psi_\Pi)$ in $\mathsf{S}[\Pi]$ and $\phi' = \forall(V^4 \setminus B^4)\Psi_\Pi[B^4/B^1]$ in $\mathsf{T}[\Pi]$. Hence, by Proposition 3, it suffices to show that ϕ is equivalent to ϕ', i.e., for any interpretation I, $I \models \phi$ iff $I \models \phi'$. Furthermore, note that the free variables of both ϕ and ϕ' are given by $A^2 \cup B^1$.

($\Rightarrow$) Assume that $X^2 \cup Y^1 \not\models \phi'$, for some $X^2 \subseteq A^2$ and $Y^1 \subseteq B^1$. Thus, for some $Z^4 \subseteq V^4 \setminus B^4$, $X^2 \cup Y^1 \cup Z^4 \not\models \Psi_\Pi[B^4/B^1]$. We show that then $X^2 \cup Y^1 \not\models \phi$ as well. Indeed, for the interpretation $I = X^2 \cup Y^1 \cup Z^4 \cup Y^4$, $I \models B^4 = B^1$ but $I \not\models \Psi_\Pi$. The former is by Theorem 12 since $I|_{B^4} = Y^4$ and $I|_{B^1} = Y^1$. The latter clearly follows from $X^2 \cup Y^1 \cup Z^4 \not\models \Psi_\Pi[B^4/B^1]$, because, for any $a \in B$, $I \models a^4$ iff $I \models a^1$.

($\Leftarrow$) For the other direction, we show that $X^2 \cup Y^1 \not\models \phi'$ follows from $X^2 \cup Y^1 \not\models \phi$, for some $X^2 \subseteq A^2$ with $Y^1 \subseteq B^1$. By assumption, $I \not\models (B^4 = B^1) \rightarrow \Psi_\Pi$, for some $I = X^2 \cup Y^1 \cup Z^4$ with $Z^4 \subseteq V^4$. Thus, $I \models B^4 = B^1$ and $I \not\models \Psi_\Pi$. The former implies that $Z \cap B = Y$ since $I|_{B^4} = (Z \cap B)^4$ and $I|_{B^1} = Y^1$ (cf. Theorem 12). It remains to show that $X^2 \cup Y^1 \not\models \phi'$. In fact, $X^2 \cup Y^1 \cup (Z \setminus B)^4 \not\models \Psi_\Pi[B^4/B^1]$, which directly follows from $I \not\models \Psi_\Pi$ and $Z \cap B = Y$. This concludes the proof. $\square$

As an immediate consequence from the previous theorem, we thus obtain results similar to those for the encoding $\mathsf{S}[\cdot]$ from the previous section:

Theorem 16 *Let* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *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* $Y^1 \cup X^2 \models \mathsf{T}[\Pi]$*, and* Π *holds iff the closed QBF* $\neg\exists V^1 \exists A^2 \mathsf{T}[\Pi]$ *is valid.*

Moreover, for any PQEP $\Pi = (P, Q, \mathcal{F}_A, =_B)$*,* (X, Y) *is a counterexample for* Π *iff* $Y^1 \cup X^2 \models \mathsf{T}[\Pi^{\rightarrow}] \vee \mathsf{T}[\Pi^{\leftarrow}]$*, and* Π *holds iff* $\neg\exists V^1 \exists A^2 (\mathsf{T}[\Pi^{\rightarrow}] \vee \mathsf{T}[\Pi^{\leftarrow}])$ *is valid.*

Proof. Respective results for encoding $\mathsf{S}[\cdot]$ appear as Lemma 5, Theorem 13, Lemma 6, and Theorem 14. The theorem follows from these results, the equivalence of $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$ by Theorem 15, and the replacement property as stated in Proposition 3. $\square$

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. [8].

Empirical experiments with the different encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$ (cf. Chapter 6 below) indicate that using the simplified encoding $\mathsf{T}[\cdot]$ does not always lead to better running times compared to the $\mathsf{S}[\cdot]$ encoding. Performance is also considerably influenced by specifics of the used QBF solvers. From a different point of view, experiments with both encodings can be used to reveal interesting differences between QBF solvers with respect to their ability to deal with implicit simplifications, in our case, with fixed assignments for universally quantified variables.

Simplifications

We proceed with a discussion on how our new reduction can be simplified for special cases. Recall that by a proper parameterisation of a PQIP (resp., PQEP) also important special cases of equivalence problems known from the literature can be specified.

Ordinary Equivalence with Projection

For problems $(P, Q, \mathcal{F}_A, \subseteq_B)$ with $A = \emptyset$ and B arbitrary, the translation $\mathsf{T}[\cdot]$ can be simplified as follows.

Definition 10 *Let Π be a PQIP $(P, Q, \mathcal{F}_A, \subseteq_B)$ with $At(P \cup Q) = V$, $A = \emptyset$, and $B \subseteq V$, then*

$$\mathsf{T}_{oep}[\Pi] = \Big(P^{\langle 1,1\rangle} \wedge \forall V^3((V^3 < V^1) \rightarrow \neg P^{\langle 3,1\rangle})\Big) \wedge \forall(V^4 \setminus B^4)\Big(Q^{\langle 4,4\rangle} \rightarrow \exists V^5((V^5 < V^4) \wedge Q^{\langle 5,4\rangle})\Big)[B^4/B^1]\,.$$

Lemma 7 *Given a problem $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ with $A = \emptyset$, the QBF $\mathsf{T}[\Pi]$ is equivalent to $\mathsf{T}_{oep}[\Pi]$.*

Proof. Since $A = \emptyset$, all terms $(A^i \leq A^j)$ in $\mathsf{T}[\Pi]$ are equivalent to $\top$. Furthermore, $(A^i \leq A^j)$ only occurs in $\mathsf{T}[\Pi]$ within a subformula of $\mathsf{T}[\Pi]$ which is of form $(A^i \leq A^j) \wedge \phi$ or $\phi \wedge (A^i \leq A^j)$. In either case, we can replace that subformula by ϕ in $\mathsf{T}[\Pi]$ and obtain an equivalent formula by Proposition 3. Let Φ_Π, Φ'_Π, and Ψ_Π be the subformulas of $\mathsf{T}[\Pi]$ from Definition 9. Then, Φ_Π simplifies to $P^{\langle 1,1\rangle}$, Φ'_Π simplifies to

$$\forall V^3((V^3 < V^1) \rightarrow \neg P^{\langle 3,1\rangle})\,,$$

and Ψ_Π simplifies to

$$\Big(Q^{\langle 4,4\rangle} \rightarrow \exists V^5((V^5 < V^4) \wedge Q^{\langle 5,4\rangle})\Big)$$

after respective replacements. We thus obtain $\mathsf{T}_{oep}[\Pi]$. □

Theorem 17 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $A = \emptyset$ *and* $At(P \cup Q) = V$, Π *holds iff* $\neg\exists V^1(\mathsf{T}_{oep}[\Pi^{\rightarrow}] \vee \mathsf{T}_{oep}[\Pi^{\leftarrow}])$ *is valid.*

Proof. Note that the free variables of $\mathsf{T}_{oep}[\Pi]$ are given by V^1 only. By Theorem 16, Lemma 7, and Proposition 3, it follows that Π holds iff $\neg\exists V^1(\mathsf{T}_{oep}[\Pi^{\rightarrow}] \vee \mathsf{T}_{oep}[\Pi^{\leftarrow}])$ is valid. □

Observe that on each branch of the formula tree of $\mathsf{T}_{oep}[\cdot]$ there are at most two quantifier alternations, witnessing the Π_3^P-complexity of this special case (cf. Theorem 10).

Relativised Uniform Equivalence

Next, we analyse special settings without projection, i.e., problems of form $(P, Q, \mathcal{F}_A, \subseteq_B)$ 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 equivalence problems without projection (cf. Proposition 10), we expect that the number of quantifier alternations in the resulting QBFs decreases by one.

Definition 11 *Let* Π *be a PQIP* $(P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $At(P \cup Q) = V$, $B = \mathcal{U}$, *and* $A \subseteq V$, *then*

$$\mathsf{T}_{rue}[\Pi] = \Phi_\Pi \wedge \Phi'_\Pi \wedge \left(Q^{\langle 1,1\rangle} \rightarrow \exists V^5((A^2 \leq A^5) \wedge (V^5 < V^1) \wedge Q^{\langle 5,1\rangle})\right),$$

where Φ_Π *and* Φ'_Π *are defined as in Definition 9.*

Lemma 8 *Given a problem* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $B = \mathcal{U}$, *the QBF* $\mathsf{T}[\Pi]$ *is equivalent to* $\mathsf{T}_{rue}[\Pi]$.

Proof. Let Ψ'_Π be the following subformula of $\mathsf{T}[\Pi]$:

$$\Psi'_\Pi = \forall(V^4 \setminus B^4)\,((Q^{\langle 4,4\rangle} \wedge (A^2 \leq A^4)) \rightarrow \exists V^5(((A^2 \leq A^5) \wedge (V^5 < V^4)) \wedge Q^{\langle 5,4\rangle}))[B^4/B^1]\,.$$

Observe that the quantifier block $\forall(V^4 \setminus B^4)$ in Ψ'_Π vanishes since $V \setminus B = \emptyset$. Since $B = \mathcal{U}$, all atoms v^4 in the scope of $\forall(V^4 \setminus B^4)$ are replaced by v^1. Hence, Ψ'_Π is of form

$$((Q^{\langle 1,1\rangle} \wedge (A^2 \leq A^1)) \rightarrow \exists V^5(((A^2 \leq A^5) \wedge (V^5 < V^1)) \wedge Q^{\langle 5,1\rangle}))\,.$$

Observe that Φ_Π is of form $\psi \wedge (A^2 \leq A^1)$. Therefore, the condition $(A^2 \leq A^1)$ in the antecedent of Ψ'_Π is redundant in $\mathsf{T}[\Pi]$. It follows that $\mathsf{T}[\Pi]$ is equivalent to $\mathsf{T}_{rue}[\Pi]$ because the latter results from $\mathsf{T}[\Pi]$ by replacing $(Q^{\langle 1,1\rangle} \wedge (A^2 \leq A^1))$ in Ψ'_Π by $Q^{\langle 1,1\rangle}$. □

Theorem 18 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $B = \mathcal{U}$, $At(P \cup Q) = V$, *and* $A \subseteq V$, Π *holds iff* $\neg\exists V^1 \exists A^2(\mathsf{T}_{rue}[\Pi^{\rightarrow}] \vee \mathsf{T}_{rue}[\Pi^{\leftarrow}])$ *is valid.*

Proof. The result immediately follows from Theorem 16, Lemma 8, and Proposition 3. □

Note that the structure of the formula $\mathsf{T}_{rue}[\cdot]$ indeed matches the Π_2^P-complexity result for relativised uniform equivalence.

Interestingly, QBF $\mathsf{T}_{rue}[\cdot]$ is satisfiability equivalent to an even simpler formula.

Definition 12 *Let Π be a PQIP $(P, Q, \mathcal{F}_A, \subseteq_B)$ with $At(P \cup Q) = V$, then*

$$\mathsf{T}^{\circ}[\Pi] = \Phi_{\Pi} \wedge \Phi'_{\Pi} \wedge (Q^{\langle 1,1 \rangle} \rightarrow ((V^2 < V^1) \wedge Q^{\langle 2,1 \rangle}))\,,$$

where Φ_{Π} and Φ'_{Π} are defined as in Definition 9.

Lemma 9 *For any problem $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ with $B = \mathcal{U}$, the QBF $\mathsf{T}_{rue}[\Pi]$ is satisfiability equivalent to $\mathsf{T}^{\circ}[\Pi]$.*

Proof. Assume $I \models \mathsf{T}_{rue}[\Pi]$, where $I = Y^1 \cup X^2$, $Y \subseteq V$, and $X \subseteq A$. We show that there exists an interpretation I' such that $I' \models \mathsf{T}^{\circ}[\Pi]$. If $I \not\models Q^{\langle 1,1 \rangle}$, then clearly $I \models \mathsf{T}^{\circ}[\Pi]$. So assume that $I \models Q^{\langle 1,1 \rangle}$. It follows that $I \models \exists V^5(((A^2 \leq A^5) \wedge (V^5 < V^1)) \wedge Q^{\langle 5,1 \rangle})$. Hence, for some $J \subseteq V$, $I \cup J^5 \models ((A^2 \leq A^5) \wedge (V^5 < V^1)) \wedge Q^{\langle 5,1 \rangle}$. By Theorem 12, we get that $X \subseteq J|_A$ and $J \subset Y$. Define $I' = I \cup J^2$. Note that $I'|_{V^2} = J^2$ and $I'|_{A^2} = J|_A{}^2$. Since $I \models \Phi_{\Pi}$, $I|_{V^1} = I'|_{V^1}$, and $J_A \subseteq Y_A$, it follows that $I' \models \Phi_{\Pi}$. Regarding Φ'_{Π}, it holds that for each Z with $X \subseteq Z$ and $Z \subset V$, $I \cup Z^3 \models \neg P^{\langle 3,1 \rangle}$. As $I'|_{V^2} = J^2$, $I'|_{A^2} = J|_A{}^2$, $X \subseteq J$, and $J \subset Y$, also $I' \models \Phi'_{\Pi}$. It remains to show that $I' \models Q^{\langle 1,1 \rangle} \rightarrow ((V^2 < V^1) \wedge Q^{\langle 2,1 \rangle})$. That $I' \models Q^{\langle 1,1 \rangle}$ follows from $I \models Q^{\langle 1,1 \rangle}$, $I' \models (V^2 < V^1)$ follows from $J \subset Y$ and Theorem 12, and $I \models Q^{\langle 2,1 \rangle}$ follows from $I'|_{V^2} = J^2$, $I \cup J^5 \models Q^{\langle 5,1 \rangle}$, and Theorem 11. Hence, satisfiability of $\mathsf{T}_{rue}[\Pi]$ entails satisfiability of $\mathsf{T}^{\circ}[\Pi]$.

For the other direction, assume $I \models \mathsf{T}^{\circ}[\Pi]$, where $I = Y^1 \cup X^2$, $Y \subseteq V$, and $X \subseteq V$. As before, we assume that $I \models Q^{\langle 1,1 \rangle}$. We show that $I \models \mathsf{T}_{rue}[\Pi]$. To this end, we only need to show that $I \models \exists V^5(((A^2 \leq A^5) \wedge (V^5 < V^1)) \wedge Q^{\langle 5,1 \rangle})$, i.e., for some $Z \subseteq V$, $Y^1 \cup X|_A{}^2 \cup Z^5 \models (A^2 \leq A^5) \wedge (V^5 < V^1) \wedge Q^{\langle 5,1 \rangle}$. We take X for Z. From $I \models (V^2 < V^1)$ follows, by Theorem 12, $X \subset Y$. It holds that $X|_A{}^2 \cup Z^5 \models (A^2 \leq A^5)$ because $X = Z$ and thus $X|_A \subseteq Z$. Likewise, $Y^1 \cup Z^5 \models (V^5 < V^1)$ follows from $X \subset Z$. Finally, by Theorem 11, $I \models Q^{\langle 2,1 \rangle}$, $I = Y^1 \cup X^2$, and $Z = X$ implies that $Y^1 \cup Z^5 \models Q^{\langle 5,1 \rangle}$. Hence, I is indeed a model of $\mathsf{T}_{rue}[\Pi]$. □

Theorem 19 *For any PQEP $\Pi = (P, Q, \mathcal{F}_A, =_B)$ with $B = \mathcal{U}$ and $At(P \cup Q) = V$, Π holds iff $\neg\exists V^1 \exists V^2(\mathsf{T}^{\circ}[\Pi^{\rightarrow}] \vee \mathsf{T}^{\circ}[\Pi^{\leftarrow}])$ is valid.*

Proof. It follows from Lemma 9 that $\phi = \mathsf{T}^{\circ}[\Pi^{\rightarrow}] \vee \mathsf{T}^{\circ}[\Pi^{\leftarrow}]$ and $\psi = \mathsf{T}_{rue}[\Pi^{\rightarrow}] \vee \mathsf{T}_{rue}[\Pi^{\leftarrow}]$ are satisfiability equivalent. Note that $\exists V^1 \exists V^2\, \phi$ is the existential closure of ϕ, and $\exists V^1 \exists A^2\, \psi$ is the existential closure of ψ. Hence, $\exists V^1 \exists V^2\, \phi$ and $\exists V^1 \exists A^2\, \psi$ are equivalent, and so are $\neg\exists V^1 \exists V^2\, \phi$ and $\neg\exists V^1 \exists A^2\, \psi$. Now, the theorem follows directly from Theorem 18. □

Note that, although we can get rid off the quantifier block $\exists V^5$ in $\mathsf{T}_{rue}[\cdot]$ as well, this simplification does not influence the number of quantifier alternations.

Uniform Equivalence

For the case of uniform equivalence, the associated PQIPs are of form $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ with $A = B = \mathcal{U}$. For this setting, however, we get no further simplifications compared to $\mathsf{T}_{rue}[\Pi]$. In any case, we have the following corollary from Theorem 18.

Corollary 2 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $A = B = \mathcal{U}$ *and* $At(P \cup Q) = V$, Π *holds iff* $\neg\exists V^1 \exists V^2 (\mathsf{T}_{rue}[\Pi^{\rightarrow}] \vee \mathsf{T}_{rue}[\Pi^{\leftarrow}])$ *is valid.*

As uniform equivalence is a special case of relativised uniform equivalence, also this QBF is satisfiability equivalent to $\mathsf{T}^{\circ}[\Pi]$. The following result is a corollary of Theorem 19.

Corollary 3 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $A = B = \mathcal{U}$ *and* $At(P \cup Q) = V$, Π *holds iff* $\neg\exists V^1 \exists V^2 (\mathsf{T}^{\circ}[\Pi^{\rightarrow}] \vee \mathsf{T}^{\circ}[\Pi^{\leftarrow}])$ *is valid.*

Again, the formula structure of the respective QBF encodings reflect the complexity of verifying uniform equivalence.

Ordinary Equivalence

Finally, for ordinary equivalence, the associated PQIPs are problems of form $(P, Q, \mathcal{F}_A, \subseteq_B)$ with $A = \emptyset$ and $B = \mathcal{U}$. Here, we observe similar effects as in the encoding for ordinary equivalence with projection.

The following lemma directly relates QBFs and answer sets of a logic program. In fact, the following translations coincides with the encoding for computing answer sets via QBFs as described by Egly et al. [18] modulo a different notation for new atoms.

Lemma 10 *Let* P *be a DLP with* $At(P) = V$ *and* Y *an interpretation. Then,* $Y \in AS(P)$ *iff*

$$Y^1 \models P^{\langle 1,1\rangle} \wedge \forall V^2((V^2 < V^1) \rightarrow \neg P^{\langle 2,1\rangle}) .$$

Proof. Interpretation Y^1 is a model of the above formula iff (i) $Y^1 \models P^{\langle 1,1\rangle}$ and (ii) for each $Z \subseteq V$, $Y^1 \cup Z^2 \models (V^2 < V^1)$ implies that $Y^1 \cup Z^2 \not\models P^{\langle 2,1\rangle}$. By Theorem 11, (i) holds iff $Y \models P^Y$. Furthermore, by Theorem 12 and Theorem 11, (ii) holds iff, for each Z with $Z \subset Y$, $Z \not\models P^Y$. Hence, (i) and (ii) hold iff $Y \in AS(P)$. □

Definition 13 *Let* Π *be a PQIP with* $(P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $At(P \cup Q) = V$, *then*

$$\mathsf{T}_{oe}[\Pi] = \Big(P^{\langle 1,1\rangle} \wedge \forall V^3((V^3 < V^1) \rightarrow \neg P^{\langle 3,1\rangle})\Big) \wedge \Big(Q^{\langle 1,1\rangle} \rightarrow \exists V^5((V^5 < V^1) \wedge Q^{\langle 5,1\rangle})\Big).$$

Lemma 11 *Given a problem* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $A = \emptyset$ *and* $B = \mathcal{U}$, $\mathsf{T}[\Pi]$ *is equivalent to* $\mathsf{T}_{oe}[\Pi]$.

Proof. Note that, since $A = \emptyset$, the free variables of $\mathsf{T}[\Pi]$ as well as of $\mathsf{T}_{oe}[\Pi]$ reduce to V^1. Moreover, for each $Y \subseteq V$, $Y^1 \models \mathsf{T}[\Pi]$ iff $Y \in AS(P)$ but $Y \notin AS(Q)$ (cf. Theorem 16). Note that

$$Q^{\langle 1,1\rangle} \rightarrow \exists V^5((V^5 < V^1) \wedge Q^{\langle 5,1\rangle})$$

in $\mathsf{T}_{oe}[\Pi]$ is the negation of

$$Q^{\langle 1,1\rangle} \wedge \forall V^5((V^5 < V^1) \rightarrow \neg Q^{\langle 5,1\rangle}) .$$

By Lemma 10, $Y^1 \models \mathsf{T}_{oe}[\Pi]$ iff $Y \in AS(P)$ but $Y \notin AS(Q)$, hence $\mathsf{T}_{oe}[\Pi]$ and $\mathsf{T}[\Pi]$ are equivalent. □

Theorem 20 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $A = \emptyset$*,* $B = \mathcal{U}$*, and* $At(P \cup Q) = V$*,* Π *holds iff* $\neg\exists V^1(\mathsf{T}_{oe}[\Pi^{\rightarrow}] \vee \mathsf{T}_{oe}[\Pi^{\leftarrow}])$ *is valid.*

Proof. The result immediately follows from Theorem 16, Lemma 11, and Proposition 3. □

Note that the structure of the closed QBF $\neg\exists V^1(\mathsf{T}_{oe}[\Pi^{\rightarrow}] \vee \mathsf{T}_{oe}[\Pi^{\leftarrow}])$ witnesses the Π_2^P-membership of ordinary equivalence. Moreover, ordinary equivalence is also a special case of relativised strong equivalence with projection. Recall that similar QBF reductions for the latter problem have been introduced and implemented in previous work [55, 81]. It turns out that the resulting QBFs for PQEPs and relativised strong equivalence coincide for the special case of ordinary equivalence.

As ordinary equivalence is a special case of relativised uniform equivalence, we can obtain a further simplification in terms of $\mathsf{T}^{\circ}[\cdot]$.

Definition 14 *Let* Π *be a PQIP* $(P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $At(P \cup Q) = V$*, Then,*

$$\mathsf{T}^{\triangle}[\Pi] = \left(P^{\langle 1,1\rangle} \wedge \forall V^3((V^3 < V^1) \rightarrow \neg P^{\langle 3,1\rangle})\right) \wedge \left(Q^{\langle 1,1\rangle} \rightarrow ((V^5 < V^1) \wedge Q^{\langle 5,1\rangle})\right).$$

Lemma 12 *For any PQIP* $\Pi = (P, Q, \mathcal{F}_A, \subseteq_B)$ *with* $A = \emptyset$ *and* $B = \mathcal{U}$*,* $\mathsf{T}_{oe}[\Pi]$ *is satisfiability equivalent to* $\mathsf{T}^{\triangle}[\Pi]$*.*

Proof. Since no atom from V^5 occurs outside the scope of $\exists V^5$ in $\mathsf{T}_{oe}[\Pi]$, we can shift $\exists V^5$ and write $\mathsf{T}_{oe}[\Pi]$ as $\exists V^5\, \mathsf{T}^{\triangle}[\Pi]$. Hence, as $\mathsf{T}_{oe}[\Pi]$ and $\exists V^5\, \mathsf{T}^{\triangle}[\Pi]$ are equivalent, $\mathsf{T}_{oe}[\Pi]$ and $\mathsf{T}^{\triangle}[\Pi]$ are *a fortiori* satisfiability equivalent. □

Theorem 21 *For any PQEP* $\Pi = (P, Q, \mathcal{F}_A, =_B)$ *with* $A = \emptyset$*,* $B = \mathcal{U}$*, and* $At(P \cup Q) = V$*,* Π *holds iff* $\neg\exists V^1 \exists V^5(\mathsf{T}^{\triangle}[\Pi^{\rightarrow}] \vee \mathsf{T}^{\triangle}[\Pi^{\leftarrow}])$ *is valid.*

Proof. The result follows from Theorem 20 by analogous arguments as in the proof of Theorem 19. □

We have shown that all special cases with $B = \mathcal{U}$ have in common that the respective encodings always yield QBFs with at most one quantifier alternation in each branch of the formula, witnessing the Π_2^P-membership of those problems. Thus, all presented simplifications are adequate in the sense that, after putting them into PNF, the number of quantifier alternations in the prenex always matches the complexity for the respective notion of program correspondence.

4.4 Transformations into Normal Forms

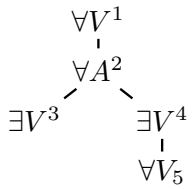
Most available QBF solvers require its input formula to be in a certain normal form, viz. in prenex conjunctive normal form (PCNF). Hence, in order to employ these solvers, the translations described above have to be transformed by a further two-phased normalisation step which consists of the following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

1. translating the given QBF into prenex normal form (PNF), and

2. translating the propositional part of the resulting 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, a QBF does not have a unique PNF in general. In this sense, the step of translating a QBF into PNF, also known as *prenexing*, is not deterministic. As discussed by Egly et al. [19], there are numerous so-called *prenexing strategies*. The concrete selection of such a strategy, also depending on the specific QBF solver used, crucially influences the running times of the QBF solver. When 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 $\neg\exists V^1\exists A^2\mathsf{S}[\cdot]$ and $\neg\exists V^1\exists A^2\mathsf{T}[\cdot]$. First, observe, however, that both encodings have negation as their outermost connective which has to be shifted into the formula by applying suitable equivalence-preserving transformations which are similar to ones well 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 $\neg\exists V^1\exists A^2\mathsf{S}[\cdot]$ can then be illustrated as follows:



Here, the left branch results from the subformula Φ and the right one results from the subformula $\forall V^4((B^4 = B^1) \rightarrow \Psi)$ in $\mathsf{S}[\cdot]$.

Inspecting these quantifier dependencies, we can only group together $\forall V^1$ with $\forall A^2$ and $\exists V^3$ with $\exists V^4$. This yields the following way for prenexing our encodings:

$$\forall(V^1 \cup A^2)\exists(V^3 \cup V^4)\forall V_5.$$

Concerning the transformation of the propositional part of a prenex QBF into CNF, we use a method following Tseitin [83] in which new atoms, so-called *labels*, are introduced abbreviating subformula occurrences. More precisely, we consider an optimisation due to Plaisted and Greenbaum [70], where the polarities of the subformulas are taken into account. This so called *structure-preserving normal-form transformation* 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 with respect to the original formula in the worst case. More information on structure-preserving normal-form transformations can also be found in related work [42, 86].

Towards the transformation, we introduce the following definitions. Let ϕ be a propositional formula. The set $\Sigma(\phi)$ denotes the set of all subformulas of ϕ. By $\Sigma^+(\phi)$ we denote the set of

all subformulas occurring positively in ϕ, and by $\Sigma^-(\phi)$ we denote the set of all subformulas occurring negatively in ϕ. The set $L(\phi)$ of labels is defined as $\{L_\psi \mid \psi \in \Sigma(\phi)\}$. We assume that all labels are globally new atoms.

Definition 15 *Let ϕ be a propositional formula and $\psi, \delta, \gamma \in \Sigma(\phi)$. For each $\psi \in \Sigma(\phi)$, a* label formula *for ψ is introduced as follows:*

1. *if ψ is a propositional atom, then*

$$d^+_\psi = \neg L_\psi \vee \psi,$$
$$d^-_\psi = L_\psi \vee \neg\psi;$$

2. *if $\psi = \neg\delta$, then*

$$d^+_\psi = \neg L_\psi \vee \neg L_\delta,$$
$$d^-_\psi = L_\psi \vee L_\delta;$$

3. *if $\psi = \delta \wedge \gamma$, then*

$$d^+_\psi = (\neg L_\psi \vee L_\delta) \wedge (\neg L_\psi \vee L_\gamma),$$
$$d^-_\psi = L_\psi \vee \neg L_\delta \vee \neg L_\gamma;$$

4. *if $\psi = \delta \vee \gamma$, then*

$$d^+_\psi = \neg L_\psi \vee L_\delta \vee L_\gamma,$$
$$d^-_\psi = (L_\psi \vee \neg L_\delta) \wedge (L_\psi \vee \neg L_\gamma).$$

Furthermore, we define

$$D(\phi) = L_\phi \wedge \left(\bigwedge_{\psi \in \Sigma^+(\phi)} d^+_\psi \right) \wedge \left(\bigwedge_{\psi \in \Sigma^-(\phi)} d^-_\psi \right).$$

Obviously, $D(\phi)$ is in conjunctive normal form and $D(\phi)$ is linear in the size of ϕ. Note that the normal-form translation into CNF using labels is not validity preserving like the one based on distributivity laws but only satisfiability equivalent with respect to the introduced labels. More formally, the following proposition holds.

Proposition 11 (Plaisted and Greenbaum [70]) *Given a propositional formula ϕ, ϕ is satisfiable iff $D(\phi)$ is satisfiable.*

Recall that for QBFs, satisfiability equivalence can easily be lifted to logical equivalence by considering the existential closure.

Corollary 4 *For each propositional formula ϕ, ϕ and $\exists L(\phi) D(\phi)$ are equivalent.*

Moreover, the following essential result for closed QBFs is an immediate consequence of Corollary 4 and the replacement property for QBFs from Proposition 3.

Proposition 12 (Klotz [42], Woltran [86]) *Let $\Phi = \mathsf{Q}_n P_n \cdots \mathsf{Q}_1 P_1 \phi$, 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. Then, Φ and $\mathsf{Q}_n P_n \ldots \mathsf{Q}_1 P_1 \exists L(\phi) D(\phi)$ are logically equivalent, where $L(\phi)$ are the new labels introduced by the above CNF transformation.*

For Φ as in the above proposition, we have that $\mathsf{Q}_1 = \exists$. Hence, in this case,

$$\mathsf{Q}_n P_n \cdots \mathsf{Q}_1 P_1 \exists L(\phi)\, D(\phi)$$

is the desired PCNF, equivalent to Φ, used as input for QBF solvers requiring PCNF format for evaluating Φ. 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 Proposition 12 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⊤`—the respective tool that implements the translations from this chapter—that either the original correspondence problem or the complementary problem is encoded whenever an input yields a QBF like Ψ. The system `cc⊤` is described in more detail in the next chapter.

For the entire normal-form transformation, one can use the quantifier-shifting tool `qst` [89]. It accepts arbitrary QBFs as input and returns an equivalent PCNF QBF in *Q-DIMACS* format which is a de-facto standard for PCNF-QBF solvers [74]. The tool `qst` implements 14 different prenexing strategies and uses the mentioned structure-preserving normal-form transformation for the transformation to CNF.

CHAPTER 5

The Reasoning Tool cc⊤

The encodings from correspondence problems to QBFs discussed in the previous chapter were implemented as an extension of the system cc⊤ [55]. This includes also all simplifications outlined in the previous chapter. The abbreviation "cc⊤" stands for "correspondence-checking tool". Instead of "t", we use the symbol ⊤ in "cc⊤" which is consequently pronounced as "ze-ze-top" (with all due respect to the famous rock group). In this chapter, we describe the methodology behind cc⊤ in more detail, and we illustrate the usage of the new component of cc⊤ for deciding PQIPs and PQEPs. Experimental evaluations using different QBF solvers are reported in Chapter 6. In Chapter 7, we describe an application of cc⊤ in a real-world application.

5.1 System Methodology

The tool is entirely developed in *ANSI C*, using *LEX* and *YACC* for the parser. In its current form, the entire package consists of more than 2500 lines of code. The system is publicly available (including the source code); it can be downloaded from the Web at

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

The methodology of cc⊤ to verify correspondence problems is to reduce them to the satisfiability problem of quantified propositional logic and to use external QBF solvers [43, 52] for the latter language as backend-inference engines. The reductions required for this approach to decide PQIPs and PQEPs are described in Chapter 4.

The overall application framework for cc⊤ is depicted in Fig. 5.1. In order to decide a problem $(P, Q, \mathcal{F}_A, \odot_B)$, for $\odot \in \{\subseteq, =\}$, cc⊤ takes as input two programs, P and Q, as well as sets A and B. We refer to A as *context set* and to B as *projection set*. Command-line options select between two kinds of reductions from correspondence problems to QBFs, namely the direct encoding $\mathsf{S}[\cdot]$ and the alternative encoding $\mathsf{T}[\cdot]$. Also, the comparison relation $\odot \in \{\subseteq, =\}$ is specified via command-line arguments. Detailed invocation syntax of cc⊤ can be requested with option '`-h`'.

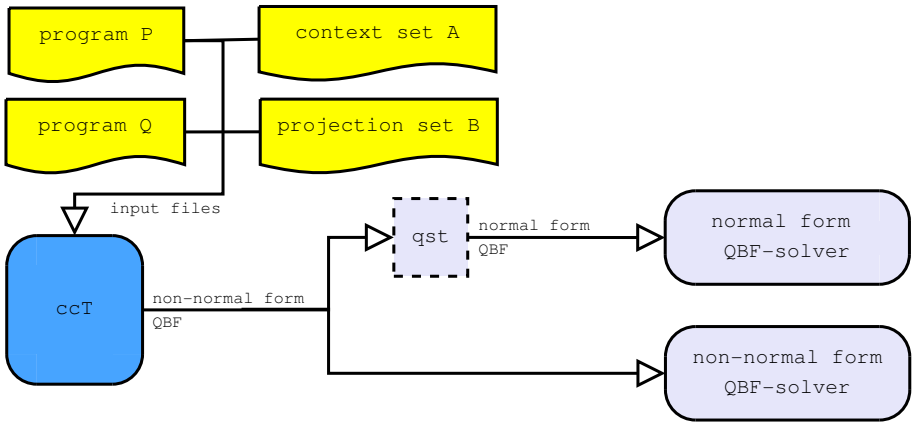


Figure 5.1: Overall application framework for ccT.

The input for ccT, consisting of programs P and Q as well as the context set A and the projection set B, is then transformed into a QBF according to the selected encodings. Afterwards, the resulting QBF is handed to a QBF solver. Validity of the resulting QBF reflects the outcome of the original problem. Since the QBFs generated by ccT are not in a particular normal form, for solvers requiring normal-form QBFs—this is the case for the vast majority of QBF solvers—, a corresponding normaliser, qst [89], is needed.

5.2 Illustrating Example

As an illustrating example, consider the programs

$$P = \left\{ \begin{array}{rcl} sad \vee happy\,, & & \\ sappy & \leftarrow & sad, happy\,, \\ confused & \leftarrow & sappy \end{array} \right\} \quad \text{and} \quad Q = \left\{ \begin{array}{rcl} sad & \leftarrow & \text{not } happy\,, \\ happy & \leftarrow & \text{not } sad\,, \\ confused & \leftarrow & sad, happy \end{array} \right\}$$

which express some knowledge about the moods of a person. While program Q is defined over atoms $\{sad, happy, confused\}$, program P additionally uses an auxiliary atom $sappy$. The intended meaning of program P is that a person always feels happy or sad. If a person is happy and sad at the same time, then we refer to this special mood as "sappy". In case a person feels sappy, then it feels confused. For program Q, the intended meaning is quite similar. A person feels happy if it does not feel sad. Likewise, a person feels sad if it does not feel happy. As for program P, a person feels confused if it is both happy and sad.

Programs P and Q can be seen as queries over a propositional database consisting of facts from, e.g., $\{happy, sad\}$. For the output, it would be natural to consider the common intensional atom $confused$. To decide whether P and Q are interchangeable in such a setting, we thus consider $\Pi = (P, Q, \mathcal{F}_A, =_B)$ as a suitable PQEP, specifying $A = \{happy, sad\}$ and $B = \{confused\}$. It is a straightforward matter to check that Π, defined in this way, holds. Note however, though P and Q are equivalent in the ordinary sense, P and Q are not uniformly equivalent and also not

uniformly equivalent relative to A. This is because answer sets of P may contain *sappy* which never occurs in any answer set of Q. This shows the relevance of projecting answer sets to a set of dedicated output atoms when comparing programs that make use of auxiliary atoms.

Next, let us turn our attention to the concrete usage of ccT. The syntax of the programs is the basic DLV syntax [17,44]. In this syntax, the two programs P and Q from the above example look as follows:

$$P = \left\{ \begin{array}{l} \texttt{sad v happy.} \\ \texttt{sappy :- sad, happy.} \\ \texttt{confused :- sappy.} \end{array} \right\}, \quad Q = \left\{ \begin{array}{l} \texttt{sad :- not happy.} \\ \texttt{happy :- not sad.} \\ \texttt{confused :- sad, happy.} \end{array} \right\}.$$

Let us assume that the two programs are stored in the files P.dl and Q.dl, respectively. The two sets A and B from the example are written as comma separated lists within brackets:

context set A: `(happy, sad),`
projection set B: `(confused).`

We assume them to be stored in files A and B, respectively. The concrete invocation syntax for translating the problem $\Pi = (P, Q, \mathcal{F}_A, =_B)$ into a corresponding QBF is

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

The command-line options "-u" together with "-e" enforce that ccT interprets input parameters P.dl, Q.dl, A, and B as a problem of form $\Pi = (P, Q, \mathcal{F}_A, =_B)$, thus as a PQEP. Without option "-u", the tool would assume $\mathcal{P}_A$ instead of $\mathcal{F}_A$ in Π and thus interpret the input files as a problem generalising strong equivalence instead of uniform equivalence. To decide an associated implication problem to Π, one has to replace "-e" by "-i" or omit the parameter. Then, ccT will assume $\subseteq_B$ instead of $=_B$ in Π. By default, ccT will use encoding $\mathsf{T}[\cdot]$ for translating Π into a QBF; option "-S" can be used to force the tool to use encoding $\mathsf{S}[\cdot]$ instead.

The resulting output QBF will be written directly to the standard-output device from where it can serve as input for QBF solvers. Besides that, ccT will write a message to the standard-output device that clarifies how the truth value of this QBF has to be interpreted, i.e., if the specified correspondence problems holds if the QBF is true, or if it holds when the QBF is false.

Since ccT does not output QBFs in a specific normal form, for using solvers requiring normal-form QBFs, the additional normaliser qst [89] can be used. For illustration, assume that nf-solver is a QBF solver that requires its input to be in PCNF, and solver is a QBF solver that can process arbitrary QBFs, then respective tool pipes to decide problem Π from above could be realised as follows:

```
ccT -u -e P.dl Q.dl A B | qst | nf-solver
```
, resp.,

```
ccT -u -e P.dl Q.dl A B | solver.
```

Important special cases of PQEPs, e.g., ordinary equivalence or uniform equivalence (possibly relative to some set of atoms), can easily be specified using respective command-line arguments. If we omit A or B when launching ccT, the tool takes $At(P \cup Q)$ for the respective set. Hence, we can use ccT to decide if P and Q are uniformly equivalent by the following command:

```
ccT -u -e P.dl Q.dl.
```

If we write `0` instead of a filename, `ccT` assumes the empty set for the respective input element. Note that it can be necessary to explicitly state which file corresponds to which input element, as illustrated by the following example. To check for ordinary equivalence, we can use `ccT` as follows:

```
ccT -u -e P.dl Q.dl -A 0.
```

Here, we use option "`-A`" to express that `0` is the context set. The projection set equals $At(P \cup Q)$. In general, we can use options "`-P`", "`-Q`", "`-A`", and "`-B`" and write

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

to state that file `P.dl` defines P, `Q.dl` defines Q, `A` defines A, and `B` defines B in a PQEP $(P, Q, \mathcal{F}_A, =_B)$. These options can be used for defining PQIPs likewise. Observe that the following invocations would have been equivalent to the one above:

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

5.3 Obtaining Counterexamples

When dealing with PQEPs and PQIPs, often one is not only interested in the outcome of the respective decision problems. In particular, if a correspondence problem does not hold, it can be important to know *why* it is not holding. Hence, it can be necessary to compute concrete counterexamples that witness that a correspondence problem does not hold.

Recall that, by Lemma 5, the explicit counterexamples for a PQIP Π correspond to the models of $\mathsf{S}[\Pi]$, and, by Lemma 6, the explicit counterexamples for a PQEP Π correspond to the models of $\mathsf{S}[\Pi^{\rightarrow}] \vee \mathsf{S}[\Pi^{\leftarrow}]$. For encoding $\mathsf{T}[\cdot]$, there is an analogous relation to explicit counterexamples with is detailed in Theorem 16. Therefore, we can use our reduction approach to QBFs not only to decide PQIPs and PQEPs but also to compute counterexamples in case program correspondence does not hold. This feature, in terms of modified QBF translations, has been incorporated into `ccT`.

For illustration, reconsider programs P, Q and files `P.dl`, `Q.dl` from the previous section. Programs P and Q are ordinarily equivalent but not uniformly equivalent. To reveal an explicit counterexample, `ccT` can be launched as follows:

```
ccT -u -e -c P.dl Q.dl.
```

Here, parameter "`-c`" specifies that the tool should generate a QBF ψ such that, in any model of ψ, the truth assignments to the free variables of ψ encode an explicit counterexample for $\Pi^{\rightarrow}$ or $\Pi^{\leftarrow}$, where $\Pi = (P, Q, \mathcal{F}_U, =_U)$. In this case, $\psi = \mathsf{T}[\Pi^{\rightarrow}] \vee \mathsf{T}[\Pi^{\leftarrow}]$, and the free variables of ψ are $V^1 \cup V^2$, where $V = At(P \cup Q)$. For any explicit counterexample (X, Y) for Π and for any model I of ψ, the atoms over V^1 that are true in I encode X, and the atoms over V^2 that are

true in I encode Y according to Theorem 16. Now, any QBF solver that can compute satisfying assignments for QBFs with free variables can be used to compute counterexamples. One such assignment would reveal that (X, Y), with $X = \{sad, happy\}$ and Y = $\{sad, happy, confused\}$, is an counterexample for $\Pi^{\leftarrow}$ since $Y \in AS(Q \cup X)$ but $Y \notin AS(P \cup X)$.

CHAPTER 6

Empirical Evaluation of cc$\top$

We now give a performance evaluation of the implemented extension of cc$\top$ for testing PQIPs and PQEPs. The goal of the experiments is to clarify the interplay of different QBF solvers, different encodings, and different problem settings in terms of running-time performance.

6.1 Experimental Setup

In the spirit of previous experiments with cc$\top$ [55], we use the reduction from QBFs to PQIPs given by the Π_3^P-hardness proof of Theorem 10 (Section 3.4) for deciding PQIPs. This provides us with a class of random benchmark problems for cc$\top$ which captures the inherent hardness of the problem.

The method to generate benchmark instances is as follows:

1. generate a random $(3, \forall)$-QBF Φ in PDNF;
2. reduce Φ to Π_Φ according to the transformation of Theorem 10, where Π_Φ holds iff Φ is valid; and
3. apply cc$\top$ to derive a corresponding QBF encoding Ψ for Π.

A particular advantage of this method is that it allows in a straightforward way to verify the correctness of the overall system: just check whether the QBF Φ, that serves as a seed in the process of generating a benchmark instance, and the QBF Ψ, that is given as output of cc$\top$, have the same truth value. Indeed, with the help of this feature, we were able to find errors in some QBF solvers.

Our benchmark set consists of 1000 instances. Each randomly generated QBF Φ of Step 1 contains 24 different atoms. From those 24 atoms, each quantifier block binds 8 of them. Each term in the PDNF QBF Φ 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 patterns in mind, we suppose the benchmark set to be located in a rather hard region.

From each QBF Φ in our benchmark set, we construct the PQIP $\Pi_\Phi = (P, Q, \mathcal{F}_A, \subseteq_B)$ such that Φ is true iff Π holds according to Step 2. 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 regarding the choice of the context set A, namely

1. the empty context $A = \emptyset$,
2. the full context $A = \mathcal{U}$, and
3. an in-between setting $\emptyset \subseteq A \subseteq \mathcal{U}$.

For the last setting, each atom in $At(P \cup Q)$ 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 choices for the context set. The QBFs stemming from $\mathsf{S}[\cdot]$ possess 197 atoms each for the empty context; 221 atoms (on average) for the in-between 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 four QBF solvers `semprop` [45] (release 24/02/02), `qube-bj` [35] (v1.2), `quantor` [9] (release 25/01/04), and `qpro` [20, 21]. We selected these solvers because they proved to be competitive in previous QBF evaluations [43, 52] 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 PCNF. Thus, for those solvers, an intermediate prenexing step is necessary. 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 (cf. Section 4.4). Recall that for both translations, $\mathtt{cc}\top$ encodes the *complementary problem* if projection is used. The reason is to avoid an additional quantifier alternation after the transformation to PCNF—details are discussed in Section 4.4.

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 in-between 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 in-between 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.

6.2 Results

Figure 6.1 summarises the results of the comparison. The different QBF solvers, encodings ($\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$, respectively), and settings for the context (empty, in-between, and full, respectively) are given on the abscissa, and the median running times in seconds are depicted on the ordinate.

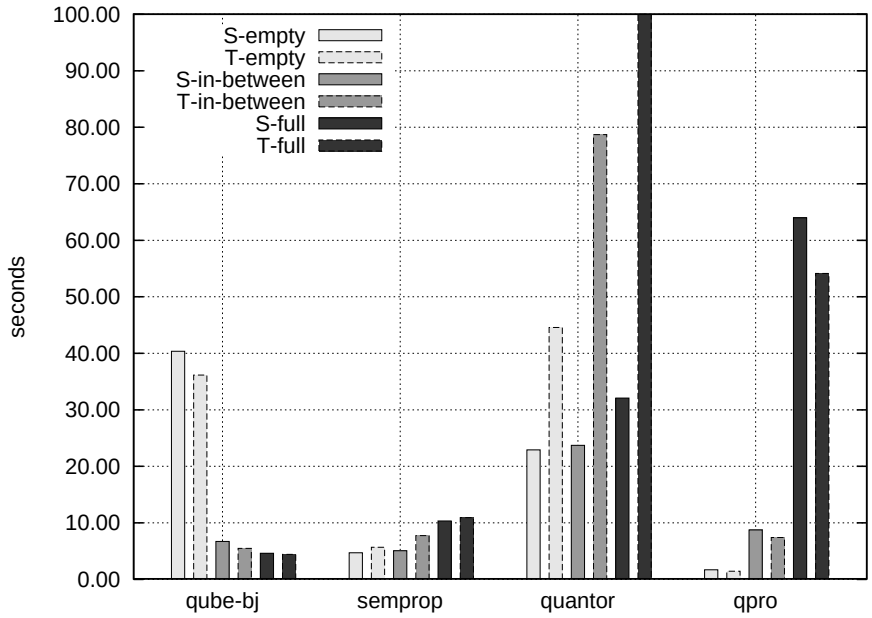


Figure 6.1: Median running times for different solvers, encodings, and problem settings.

A rather surprising observation is that the alternative encoding $\mathsf{T}[\cdot]$ does not achieve faster running times for all solvers, although it uses less propositional atoms. 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 seconds).

The next point that deserves some attention 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 $(P, Q, \mathcal{F}_A, \subseteq_B)$ either holds for any A, or does not hold for any A. The assignments of atoms from X^1 in the encodings $\mathsf{S}[\cdot]$ and $\mathsf{T}[\cdot]$ 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 6.2–6.9 provide some deeper insights concerning the running-time behaviour of the non-normal-form solver `qpro` as well as of the normal-form solvers `semprop`, `qube-bj`, and `quantor`, respectively. For those figures, the abscissa gives the running time in seconds (scaled

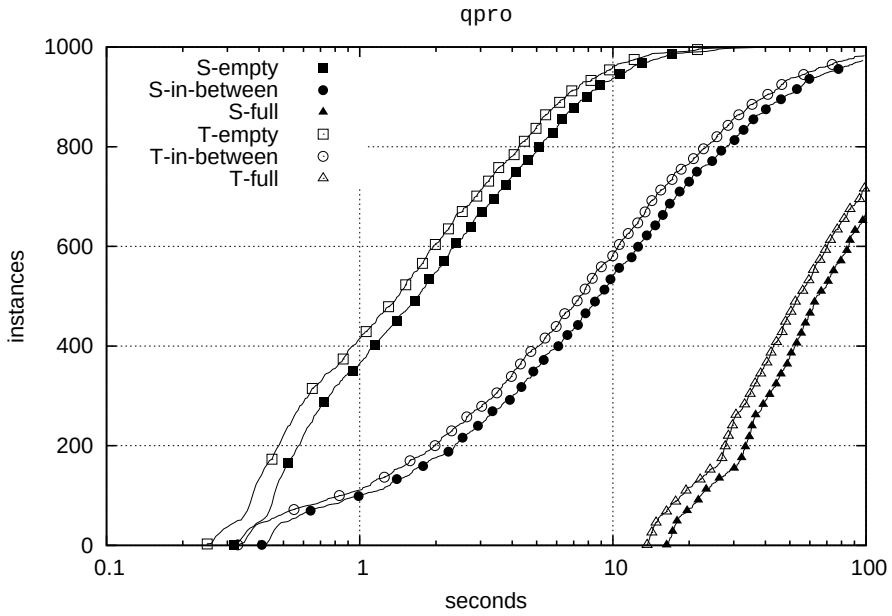


Figure 6.2: Running-time distribution for `qpro`.

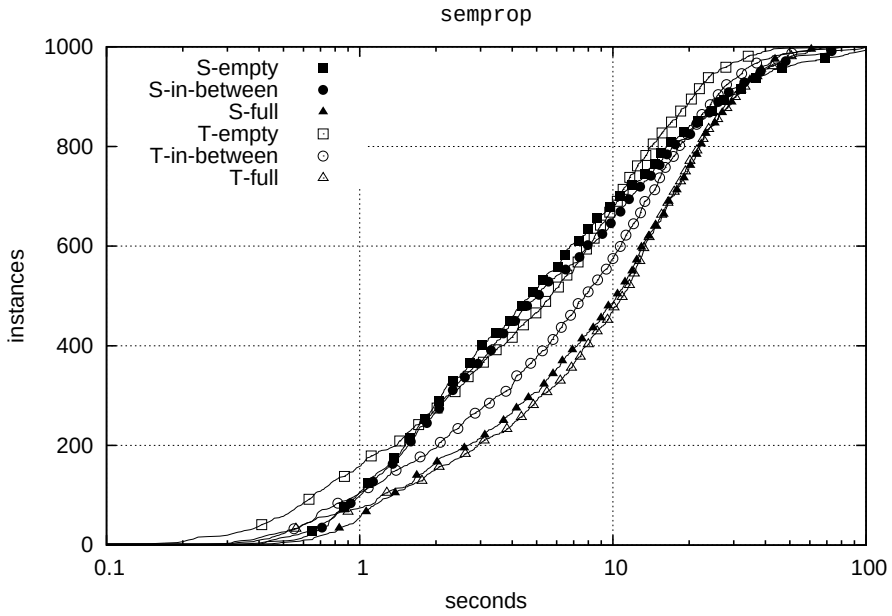


Figure 6.3: Running-time distribution for `semprop`.

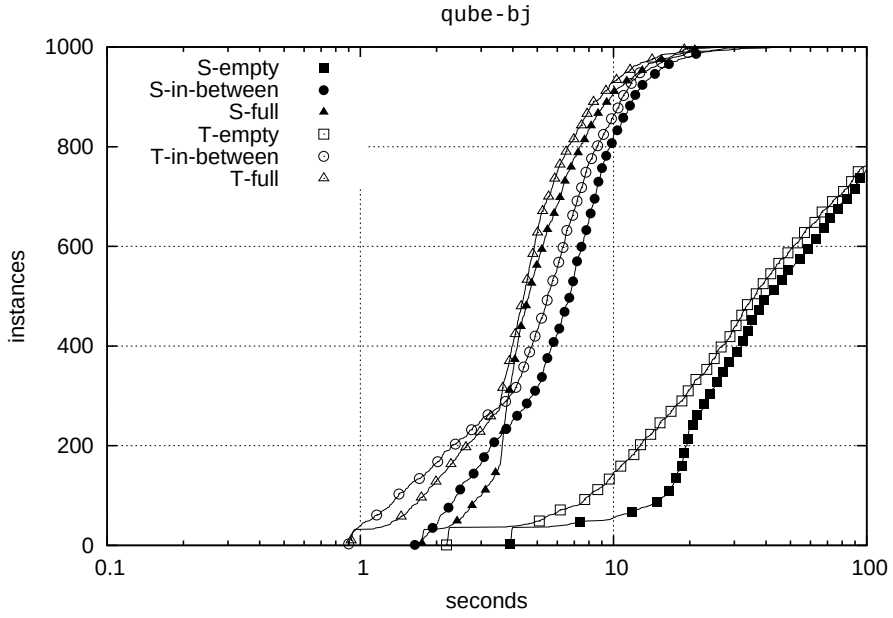


Figure 6.4: Running-time distribution for `qube-bj`.

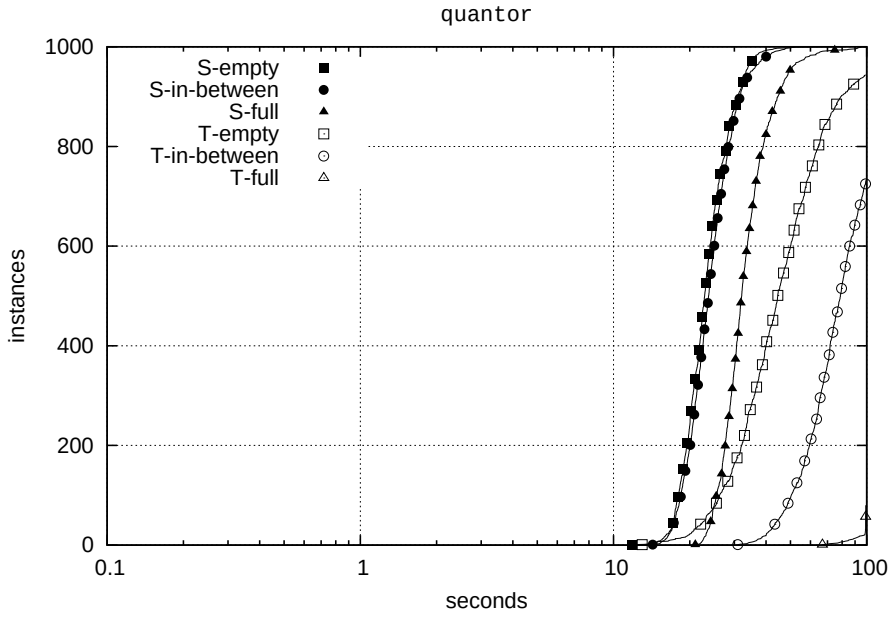


Figure 6.5: Running-time distribution for `quantor`.

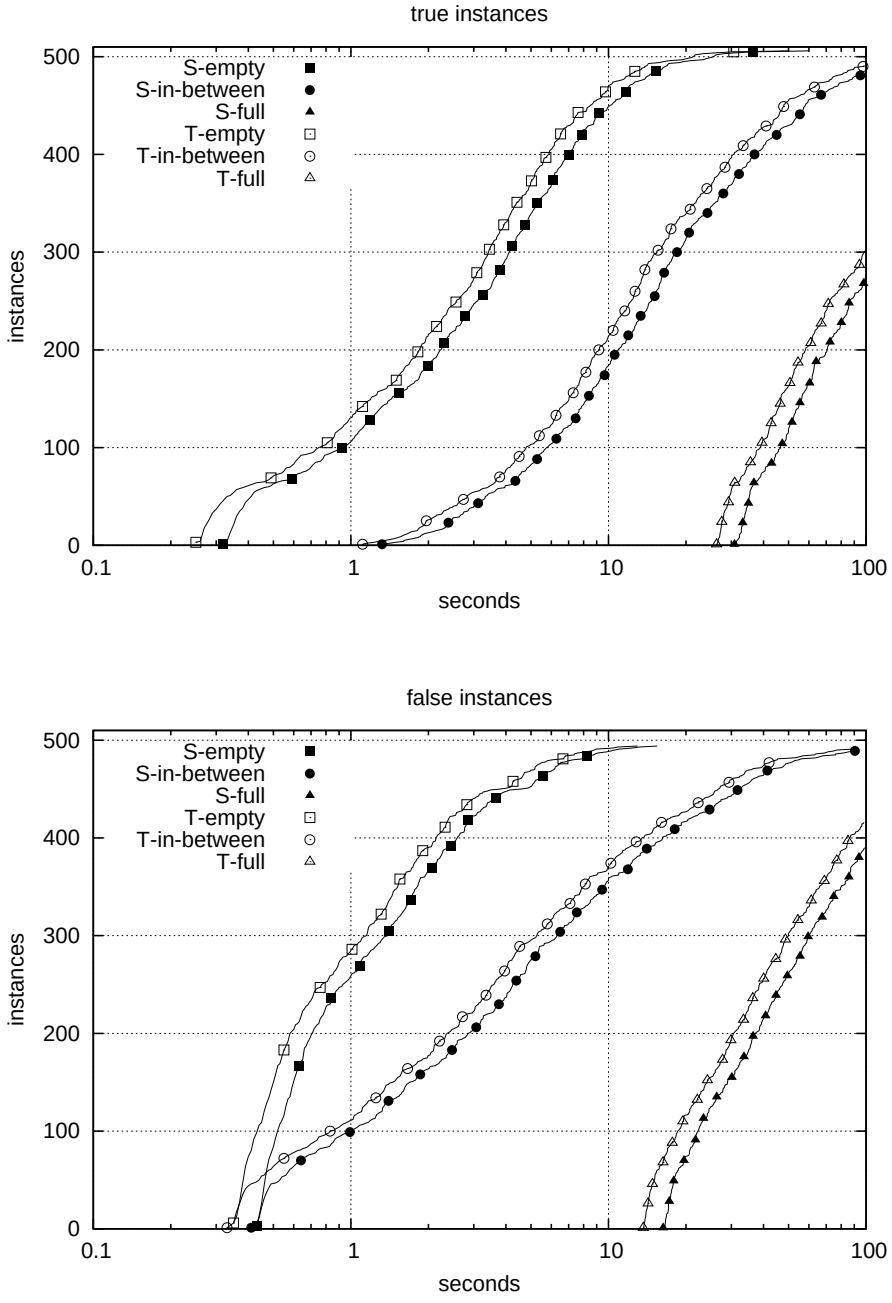


Figure 6.6: Running-time distributions for `qpro`: true and false instances.

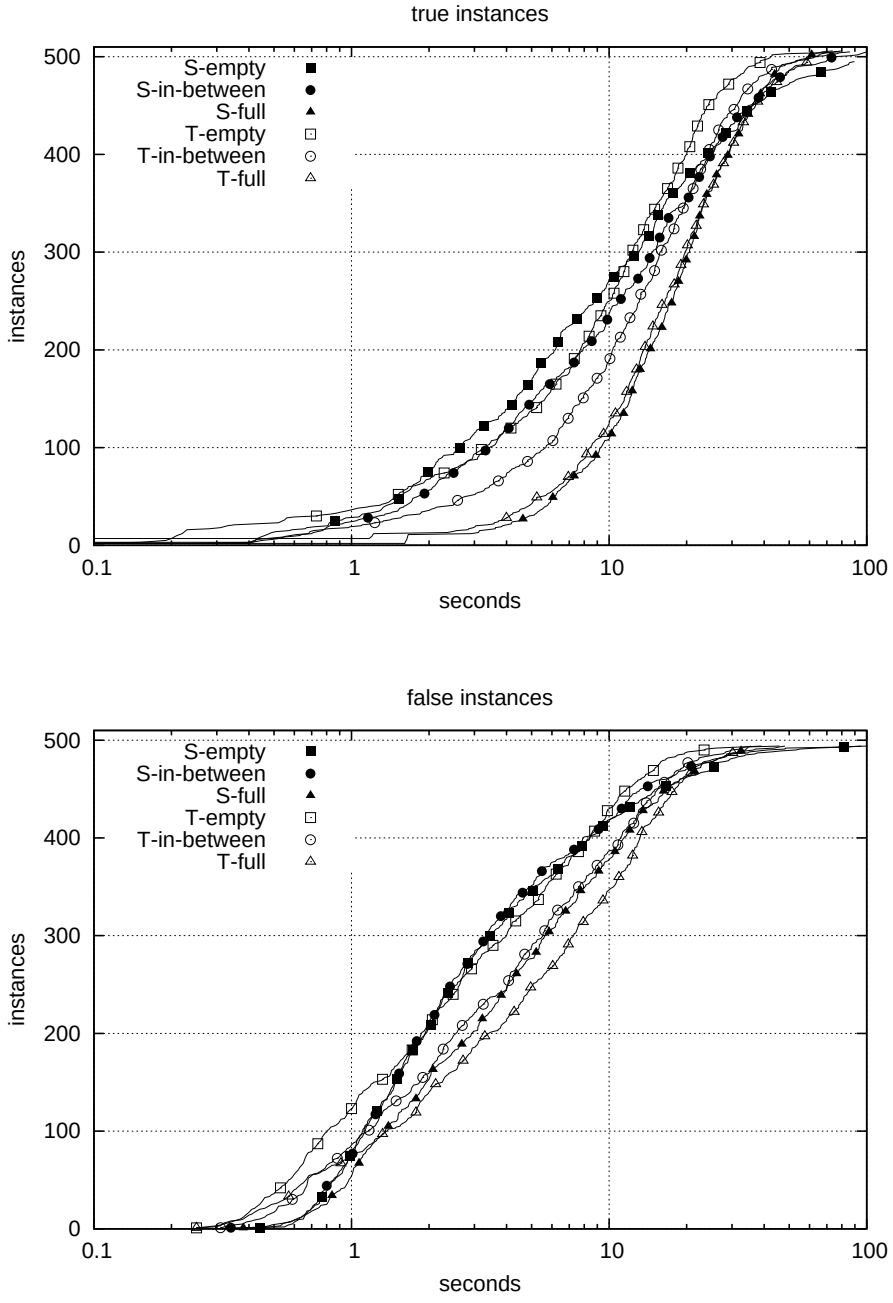


Figure 6.7: Running-time distributions for `semprop`: true and false instances.

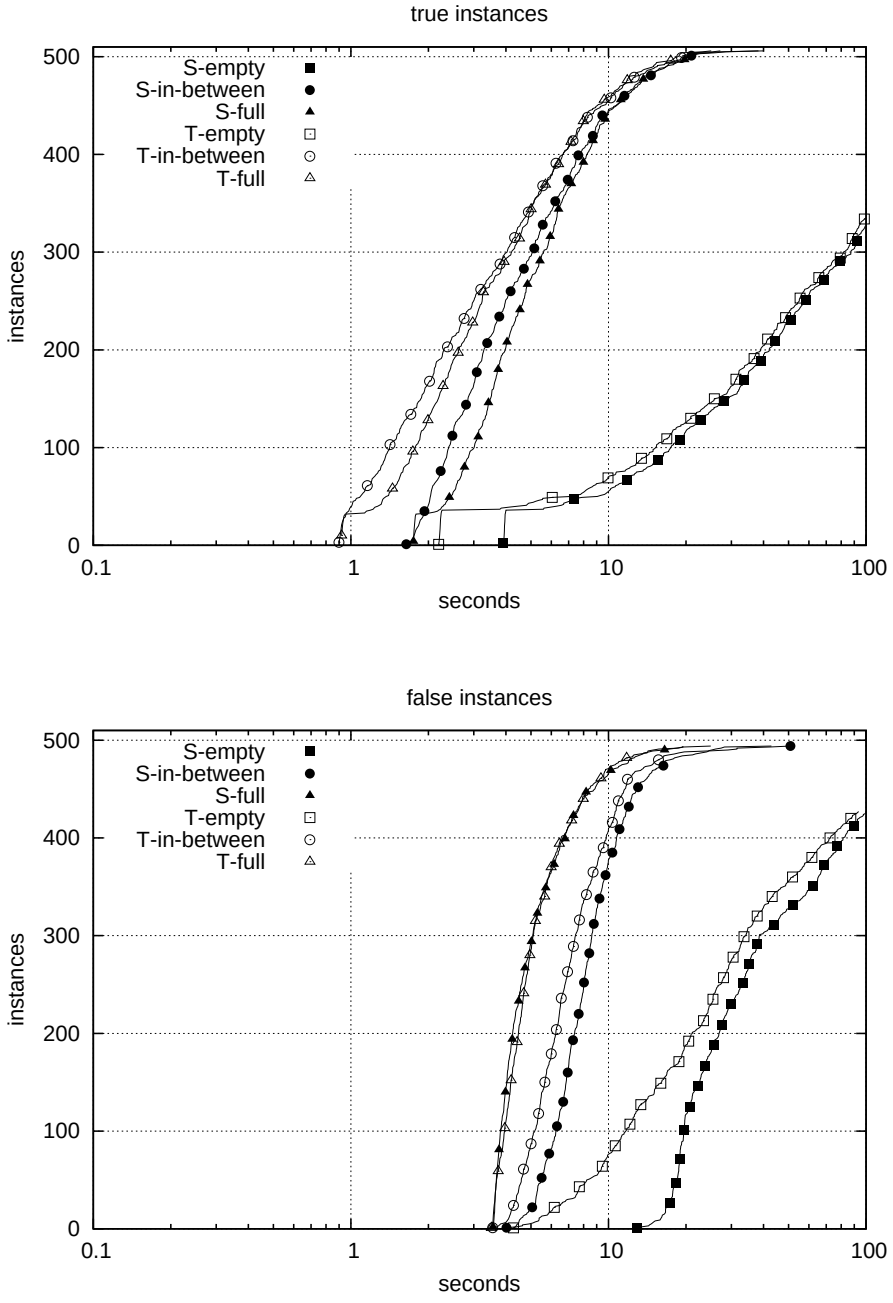


Figure 6.8: Running-time distributions for `qube-bj`: true and false instances.

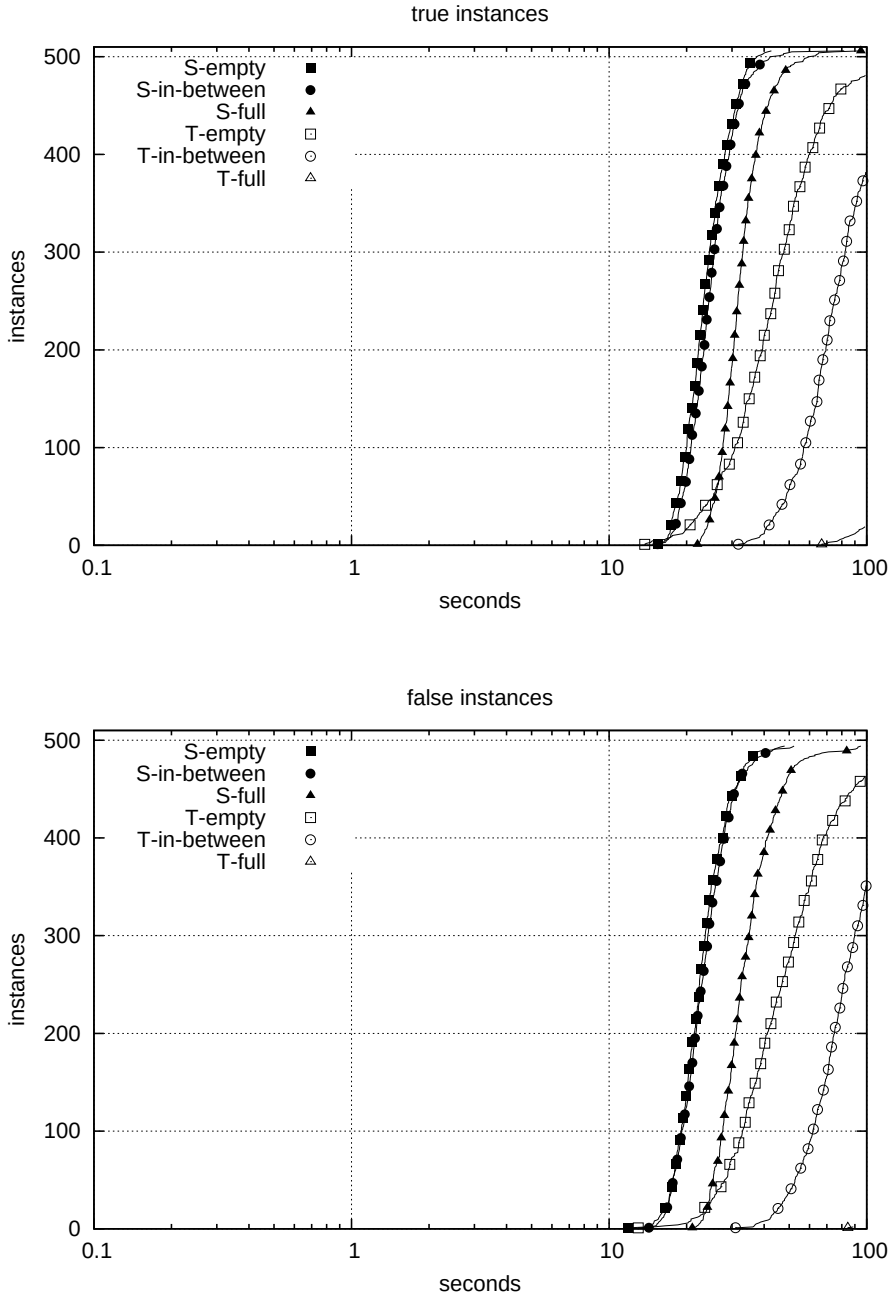


Figure 6.9: Running-time distributions for `quantor`: true and false instances.

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. Figures 6.2–6.5 depict the running-time distributions for `qpro`, `semprop`, `qube-bj`, and `quantor`, respectively. Figures 6.6–6.9 allow for a even more thorough analysis by separately depicting respective distribution for true and false problem instances.

Figure 6.2 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 6.1 extends to quite similar curves in Figure 6.3 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 6.4 provides some insight into the rather odd behaviour of `qube-bj` on this set of problem instances. While the curves for full and in-between 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 6.5) 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.

An analysis of the running times separated by true and false instances given in Figures 6.6–6.9 reveals that the tendency is that false instances are solved faster on average. However, for empty and in-between context, `qube-bj` is faster on the true instances (cf. Figure 6.8).

CHAPTER 7

cc⊤ on Stage: A Verification Application

We next discuss an application of cc⊤ for verifying the correctness of logic programs. In particular, these programs represent the solutions of students as part of their assignments for a laboratory course on logic programming and knowledge-based systems at the Vienna University of Technology. We compare these solutions relative to a reference program based on verifying certain PQEPs. As the involved programs are non-ground, i.e., they contain variables, we need special techniques to take this into account. Hence, this also demonstrates how our reduction approach to QBFs can be applied to non-ground programs as well.

7.1 Programs with Variables

So far, we only considered propositional programs, i.e., programs consisting of rules of form (2.1), where all atoms come from a fixed propositional universe. In practice, programs may (and usually do) contain variables. We say that an expression (atom, rule, or program) is *ground* if it does not contain variables, and it is *non-ground* otherwise. In this chapter, we deal with non-ground programs. That we focused on propositional programs so far is justified because the semantics of non-ground programs is defined, as we will see, by means of propositional ones.

A *non-ground program*, or *program* for short, is a finite set of rules of form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \text{not } a_{m+1}, \ldots, \text{not } a_n\,, \tag{7.1}$$

where all a_i, $1 \leq i \leq n$, are function-free first-order atoms over some fixed vocabulary. We adhere to the convention that names of constant symbols start with a lower case letter, and variables start with an upper case letter. We adopt the conventions regarding facts and constraints from rules of form (2.1). A rule r of form (7.1) is *safe* if each variable occurring in in a rule r of form (7.1) also occurs in some atom $\{a_{l+1}, \ldots, a_m\}$ in r. A program P is safe if each rule $r \in P$ is safe. We assume that all programs are safe. As usual, the *Herbrand universe*, HU_P, of a program P is

the set of all constant symbols occurring in P. If P does not contain any constant symbol, HU_P contains an arbitrary one. The *Herbrand base*, HB_P, is the set of all atoms constructible from the predicate symbols in P and the constants in HU_P.

For any rule r of form (7.1), the *grounding* of r with respect to a set C of ground terms, in symbols $grnd(r, C)$, is the smallest set of rules that contains each rule r' that can be obtained from r by uniformly replacing each variable in r by a term in C. For a program P, the grounding of P with respect to C, $grnd(P, C)$, is defined as

$$grnd(P, C) = \{r' \mid r' \in grnd(r, C), r \in P\}\,.$$

Note that $grnd(P, C)$ is always a ground program. Following custom, we identify ground programs with propositional ones.

Let P be a non-ground program. Then, the answer sets of P are defined as the answer sets of $grnd(P, HU_P)$. As in the propositional case, we use $AS(P)$ to denote the collection of all answer sets of P.

The assumption that programs are safe guarantees a property known as *language independence* [3], i.e., it is always sufficient to consider the language implicitly defined by a program for grounding the program. This is formalised by the next lemma.

Lemma 13 *Let P be a program and C a set of constant symbols such that $HU_P \subseteq C$. Then,*

$$AS(grnd(P, HU_P)) = AS(grnd(P, C))\,.$$

Proof. We start with some basic observations. Let X be an answer set of a ground program Q. First, each atom $a \in X$ occurs in the head of some rule in Q. Otherwise, X cannot be a minimal model of Q^X. Second, if Q contains a rule r such that some atom a occurs in the positive body of r and a does not occur in the head of any rule in Q, then X is answer set of $Q \setminus \{r\}$. This can be seen as follows: Since X is a minimal model of Q^X, a occurs in the positive body of r by assumption, and, by our first observation, $a \notin X$, it follows that X must be a minimal model of $Q \setminus \{r\}$ as well. By analogous arguments, one can show that for any rule r such that some atom a occurs in the positive body of r and a occurs in the head of any rule in Q, X is an answer set of $Q \cup r$.

Define $R = grnd(P, C) \setminus grnd(P, HU_P)$. Each rule $r \in R$ contains a constant symbol c in $C \setminus HU_P$, otherwise $r \in grnd(P, HU_P)$ would hold. Safety implies that some body atom a of r contains c. However, a cannot occur in the head of any rule in $grnd(P, HU_P)$, otherwise $c \in HU_P$ would hold. If follows, by our second observation, that each answer set of $grnd(P, HU_P)$ is an answer set of $grnd(P, HU_P) \cup R$ and vice versa. Observe that $grnd(P, HU_P) \cup R = grnd(P, C)$, thus $AS(grnd(P, HU_P)) = AS(grnd(P, C))$ follows. □

Note that state-of-the-art answer-set solvers like `DLV` [17,44] or `clasp` [33,72] support many language extensions like *weak constrains*, *optimisation statements*, *integer arithmetics*, *built-in comparison predicates*, *function symbols*, and *aggregates*.

Example 11 We next illustrate how instances of the 3-colourability problem (3COL) can be solved using ASP. Given a graph $G = (V, E)$ consisting of vertices V and edges $E \subseteq V \times V$, a solution for 3COL for G is a mapping from V into a set of three colours, say red, green, and

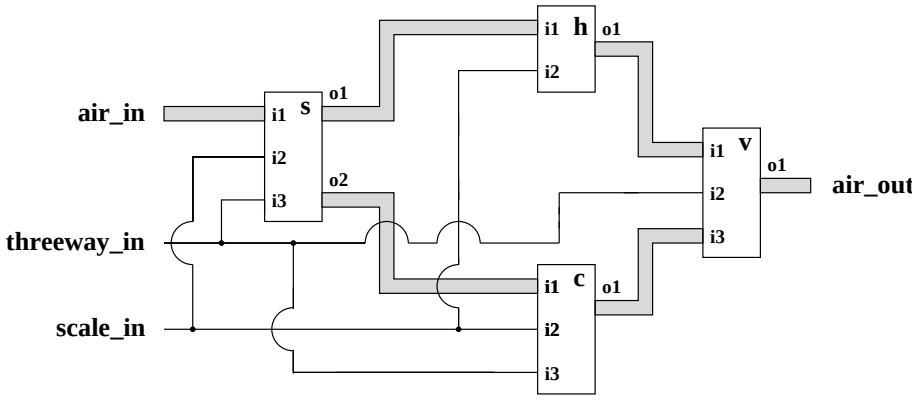


Figure 7.1: Overall architecture of an air-conditioning system considered in one of two courses of the Vienna University of Technology.

blue, such that no two adjacent vertices get assigned the same colour. The following program P encodes 3COL:

$$colour(X, red) \vee colour(X, green) \vee colour(X, blue) \leftarrow vertex(X)\,,$$
$$\leftarrow edge(X, Y), colour(X, C), colour(Y, C)\,.$$

Roughly speaking, the first rule non-deterministically assigns a colour (red, green, or blue) to a each vertex, and the second rule excludes answer sets where two adjacent vertices have the same colour. A graph is then encoded by means of facts over $vertex/1$ and $edge/2$. For instance, a graph $G = (V, E)$, with $V = \{a, b\}$ and $E = \{(a, b), (b, a)\}$, is encoded my means of the following set F of facts:

$$\{vertex(a), vertex(b), edge(a, b), edge(b, a)\}.$$

The answer sets of $P \cup F$ and the solutions of 3COL for G are in one-to-one correspondence. $\Diamond$

7.2 Problem Specification

One of the objectives of the course on knowledge-based systems mentioned above is to model a simple air-conditioning system by means of logic programs and, based on this model, to solve Reiter-style diagnosis tasks [75] with the dedicated diagnosis front-end of `DLV` [22]. The programs we consider here are students' attempts to model the components of the air-conditioning system by means of ASP. They are taken from three installments of the course between 2006 and 2008. The problem description changed only slightly from year to year.

Figure 7.1 illustrates the specification of an air-conditioning system used in 2008. This system consists of four components, viz. a heater "h", a cooler "c", a switch "s", and a valve "v". They are connected by air lines (grey bars) and data lines (ordinary lines). The students' task is to model each component of the system as well as the connections between the components and some additional constraints required for diagnosing by respective programs.

The problem description provides detailed specifications of the system and its components and defines the predicate symbols to be used for the input and output of the single components and the whole system. The system's input airstream "air_in" is modeled by a temperature value, ranging from 0 to 60, and a value specifying whether or not air is streaming ("on" or "*off*"). The same holds for the output airstream "air_out". The input values of "$threeway\_in$" is one of "$cool$", "$heat$", and "*off*". This value determines if the system should cool or heat incoming air or if the system is turned off. The value of "$threeway\_in$" is input for all components. Performance of the air-conditioning system is regulated by the input value of "$scale\_in$" which can be 0, 1, 2, or 3. Here, 3 means maximal performance. The value of "$scale\_in$" is input for the switch, the heater, and the cooler component.

The behaviour of each component as well as of the entire system is precisely specified. We illustrate such a specification along with an ASP model that represents this specification for the heater component. The following specification determines the behaviour of the heater component if it is functional:

> The heater raises the temperature of the incoming airstream by three times the value set on the data line but only to a maximum value of 45. If the incoming airstream is already warmer than 45, then it is propagated unaltered to the heater's output. Air is streaming at the component's output iff it is streaming at the component's input. If it is not streaming at the output, the temperature is set to the value defined by a dedicated predicate $ambient\_t/1$.

To model this component by means of ASP, students have to use the following predicates:

- Predicate $t(C, I, V)$ defines the temperature of an incoming our outgoing airstream of a component. In particular, $C \in \{s, h, v, c\}$ is the name of the component, $I \in \{i1, i2, i3, o1, o2\}$ is the measuring point, and $V \in \{0, \ldots, 60\}$ is the measured temperature value.

- Predicate $s(C, I, V)$ states whether or not air is streaming. Analogous to $t(C, I, V)$, $C \in \{s, h, v, c\}$ is the name of the component, $I \in \{i1, i2, i3, o1, o2\}$ is the measuring point, and $V \in \{on, \mathit{off}\}$ indicates whether air is streaming.

- Predicate $d(C, I, V)$ represents a data value of an input line. Again, $C \in \{s, h, v, c\}$ is the name of the component, and $I \in \{i1, i2, i3, o1, o2\}$ is the measuring point. For the heater component, $V \in \{0, 1, 2, 3\}$ are the possible constants for the data values.

- Predicate $ambient\_t(T)$ defines the ambient temperature, hence $T \in \{0, \ldots, 60\}$.

- Predicates $heater(C)$, $cooler(C)$, $switch(C)$, and $valve(C)$, for $C \in \{s, h, v, c\}$, define which constant symbols represent which component.

A program that represents the heater specification from above is given in Figure 7.2. Note that $ab/1$ is a special predicate employed by `DLV` referring to a defective component. An example

$\%$ relates output airstream (on, off) with input airstream
$s(H, o1, X) \leftarrow heater(H), s(H, i1, X), \text{not } ab(H)\,,$

$\%$ heating the airstream according to the specification
$t(H, o1, X) \leftarrow heater(H), t(H, i1, Y), d(H, i2, Z), s(H, o1, on),$
$\quad A = Z * 3, X = Y + A, X \leq 45, \text{not } ab(H)\,,$

$t(H, o1, 45) \leftarrow heater(H), t(H, i1, Y), d(H, i2, Z), s(H, o1, on),$
$\quad A = Z * 3, X = Y + A, X > 45, t(H, i1, T), T \leq 45, \text{not } ab(H)\,,$

$t(H, o1, T) \leftarrow heater(H), t(H, i1, Y), d(H, i2, Z), s(H, o1, on),$
$\quad A = Z * 3, X = Y + A, X > 45, t(H, i1, T), 45 < T, \text{not } ab(H)\,,$

$\%$ temperature of the output airstream when air is not streaming
$t(H, o1, X) \leftarrow heater(H), s(H, o1, \mathit{off}), ambient\_t(X), \text{not } ab(H)$

Figure 7.2: Program representing the specification of the heater component.

input for this program could be given by the following facts:

$$heater(h)\,,$$
$$ambient\_t(20)\,,$$
$$s(h, i1, on)\,,$$
$$t(h, i1, 10)\,,$$
$$d(h, i1, 2)\,.$$

Hence, the incoming airstream has a temperature value of 10 and air is floating. Moreover, the value of the data input is 3. We would expect, from the specification of the heater component, that the outgoing airstream has the temperature value 16 and air is floating as well. Indeed, if we join these input facts with the program in Figure 7.2, the unique answer set of the resulting program contains the atoms $s(h, o1, 1)$ and $t(h, o1, 16)$.

7.3 Program Verification

We next consider the verification task of determining the correctness of the students' solutions. First, we give a formal definition of a program being correct with respect to a specification. Then, we show how to reduce problems of program correctness to PQEPs, and we discuss pragmatic issues related to solving such problems.

Program Correctness

Let us denote by σ a (natural language) specification that we want to model by a logic program, and let $Ref(\sigma)$ be a reference solution, i.e., a logic program that is assumed to adequately represent σ.

Furthermore, we associate two sets, $I(\sigma)$ and $O(\sigma)$, with $Ref(\sigma)$. Both $I(\sigma)$ and $O(\sigma)$ are finite sets of ground atoms. We refer to $I(\sigma)$ as the *input signature* of $Ref(\sigma)$ and to $O(\sigma)$ as the *output signature* of $Ref(\sigma)$. Note that $Ref(\sigma)$ usually is a non-ground program. We assume that no atom in $I(\sigma)$ is intensional in $grnd(Ref(\sigma), C)$, for any set C of constant symbols.

Example 12 If σ_H is the natural language specification for the heater component from the above, $Ref(\sigma_H)$ can be defined as the program from Figure 7.2. Input and output signatures can then be fixed as follows:

$$\begin{aligned} I(\sigma_H) = &\{\, heater(C) \mid C \in Comp \,\} \cup \\ &\{\, ambient\_t(T) \mid T \in Temp \,\} \cup \\ &\{\, d(C, I, Z) \mid C \in Comp, I \in Inp, Z \in Scale \,\} \cup \\ &\{\, t(C, I, T) \mid C \in Comp, I \in Inp, T \in Temp \,\} \cup \\ &\{\, s(C, I, S) \mid C \in Comp, I \in Inp, Z \in Strm \,\} \text{ and} \end{aligned}$$

$$\begin{aligned} O(\sigma_H) = &\{\, t(C, o1, T) \mid C \in Comp, T \in Temp \,\} \cup \\ &\{\, s(C, o1, S) \mid C \in Comp, S \in Strm \,\}, \text{ where} \end{aligned}$$

$$\begin{aligned} Comp &= \{s, h, c, v\}, \\ Inp &= \{i1, i2\}, \\ Strm &= \{on, off\}, \\ Temp &= \{0, \ldots, 60\}, \text{ and} \\ Scale &= \{0, 1, 2, 3\}. \end{aligned}$$

$\Diamond$

Usually, specifications make further (sometimes implicit) assumptions concerning the input, e.g., some predicates need to be defined always or are restricted to be functional in some argument. We call a set $A \subseteq I(\sigma)$ satisfying such assumptions *admissible with respect to* σ. Analogously to specifications and reference programs, we assume that the admissible inputs of a specification σ are modelled adequately by a logic program $C(\sigma)$. Moreover, we assume that $C(\sigma)$ contains, besides atoms from $I(\sigma)$, only atoms over predicates that are globally new. Program $C(\sigma)$ defines admissible inputs as follows:

Definition 16 *Let σ be a specification. Then, $S \subseteq I(\sigma)$ is* admissible with respect to *σ iff*

$$S \in AS(C(\sigma) \cup S)|_{I(\sigma)}.$$

Example 13 Let us reconsider specification σ_H for the heater component. For σ_H, we assume that any subset $A \subseteq I(\sigma_H)$ is admissible if A contains exactly one atom over $ambient\_t/1$ and never specifies two different temperature values, data values, or streaming values by means of predicates $t/3$, $d/3$, or $s/3$, respectively. These conditions can be modelled by program $C(\sigma_H)$, given in Figure 7.3. $\Diamond$

We next give the central definition of when we say that a program P is correct with respect to a specification σ.

$\%$ one value for the ambient temperature
$def\_ambient\_t \leftarrow ambient\_t(T)$,
$\leftarrow$ not $def\_ambient\_t$,
$\leftarrow ambient\_t(T_1), ambient\_t(T_2), T_1 <> T_2$,

$\%$ s, t, and d are functional in their third argument
$fail \leftarrow s(C, I, S_1), s(C, I, S_2), S_1 <> S_2$,
$fail \leftarrow t(C, I, T_1), s(C, I, T_2), T_1 <> T_2$,
$fail \leftarrow d(C, I, D_1), s(C, I, D_2), D_1 <> D_2$,
$\leftarrow fail$

Figure 7.3: A program defining admissible inputs for the heater component.

Definition 17 *Let σ be a specification. A program P is* correct with respect to *σ iff, for each set $A \subseteq I(\sigma)$ that is admissible with respect to σ,*

$$AS(P \cup A) =_{O(\sigma)} AS(Ref(\sigma) \cup A) .$$

In the above definition, we assume that no atom in $I(\sigma)$ is intentional in $grnd(P, C)$, for any set C of constant symbols. This is no restriction, however. If a program P does not satisfy this assumption, we can always rewrite P into a program that meets this assumption: Let P be a program and σ a specification. Moreover, let P' be the program that results from P by replacing each occurrence of a predicate symbol p that also occurs in $I(\sigma)$ by $l(p)$, where l is an injective function from X into a set of globally new predicate symbols. Define $P[\sigma]$ as $P' \cup \{l(a) \leftarrow a \mid a \in L\}$. Then, $I(\sigma)$ clearly cannot contain any atom that is intensional in $grnd(P[\sigma], C)$, for any set C of constant symbols. Moreover, this transformation preserves the semantics of P. More specifically, for any set $X \subseteq I(\sigma)$, $I \in AS(P \cup X)$ iff $I' \in AS(P[\sigma] \cup X)$, where

$$I' = I \setminus I(\sigma) \cup \{l^{-1}(a)((c_1, \ldots, c_n)) \mid a(c_1, \ldots, c_n) \in I(\sigma) \cap I\} .$$

The following result establishes the connection between program correctness and PQEPs:

Theorem 22 *A program P is correct with respect to a specification σ iff $(M, N, \mathcal{F}_A, =_B)$ holds, where $A = I(\sigma)$, $B = O(\sigma)$, and*

$$M = grnd(P \cup C(\sigma), D) ,$$
$$N = grnd(Ref(\sigma) \cup C(\sigma), D),$$

where D is a finite set that contains all constants in $P \cup Ref(\sigma) \cup C(\sigma) \cup I(\sigma)$.

Proof. Program P is correct with respect to σ iff

($*$) for each set $A \subseteq I(\sigma)$ that is admissible with respect to σ,

$$AS(P \cup A) =_{O(\sigma)} AS(Ref(\sigma) \cup A).$$

Recall that $C(\sigma)$ contains, besides atoms from $I(\sigma)$, only atoms over predicates that are globally new. Moreover, for each set $S \subseteq I(\sigma)$, S is admissible with respect to σ iff $S \in AS(C(\sigma) \cup S)|_{I(\sigma)}$ by definition. Therefore, if S is admissible with respect to σ, then, for any program P, $S \in AS(P \cup C(\sigma) \cup S)|_{O(\sigma)}$ iff $S \in AS(P \cup S)|_{O(\sigma)}$. On the other hand, if S is not admissible with respect to σ, then, for any program P, $S \notin AS(P \cup C(\sigma) \cup S)$. It follows that Condition $(*)$ is equivalent to the following condition:

($\diamond$) for each set $A \subseteq I(\sigma)$, $AS(P \cup C(\sigma) \cup A) =_{O(\sigma)} AS(Ref(\sigma) \cup C(\sigma) \cup A)$.

Now,

$$AS(P \cup C(\sigma) \cup A) = AS(grnd(P \cup C(\sigma) \cup A, HU_{P \cup C(\sigma) \cup A})) \text{ and}$$
$$AS(Ref(\sigma) \cup C(\sigma) \cup A) = AS(grnd(Ref(\sigma) \cup C(\sigma) \cup A, HU_{Ref(\sigma) \cup C(\sigma) \cup A}))\,.$$

Observe that $HU_{P \cup C(\sigma) \cup A} \subseteq D$ and $HU_{Ref(\sigma) \cup C(\sigma) \cup A} \subseteq D$. By virtue of Lemma 13,

$$AS(grnd(P \cup C(\sigma) \cup A, HU_{P \cup C(\sigma) \cup A})) \;=\; AS(grnd(P \cup C(\sigma) \cup A, D)) \text{ and}$$
$$AS(grnd(Ref(\sigma) \cup C(\sigma) \cup A, HU_{Ref(\sigma) \cup C(\sigma) \cup A})) \;=\; AS(grnd(Ref(\sigma) \cup C(\sigma) \cup A, D))\,.$$

Hence, ($\diamond$) is equivalent to the following condition: for each set $A \subseteq I(\sigma)$,

$$AS(grnd(P \cup C(\sigma) \cup A, D)) =_{O(\sigma)} AS(grnd(Ref(\sigma) \cup C(\sigma) \cup A, D))$$

which, in turn, holds iff the PQEP $(M, N, \mathcal{F}_A =_B)$ holds. □

Using cc⊤ for Program Verification

In order to apply cc⊤ for verifying program correctness, we follow Theorem 22. The reason why a reduction to standard uniform equivalence or ordinary equivalence (with additional guessing rules) is not feasible, is the necessity of answer-set projection which has two sources: first, programmers usually employ auxiliary atoms which are not considered as output predicates, and second, new atoms are sometimes added by the grounding procedure (we return to this point in a moment). Recall that in terms of complexity theory, projection is the reason why deciding PQEPs is exponentially harder than to decide problems of ordinary equivalence or uniform equivalence (cf. Section 3.4).

Verifying students' solutions by following Theorem 22 and then applying cc⊤ is in principle possible but the resulting programs would be prohibitively large. So, instead of applying a naive grounding by strictly following the definition, we make use of the *intelligent grounding component* of DLV. In fact, DLV implements several optimisations when grounding a program, e.g., input rewriting, deletion of rules whose body is always false, and semi-naive evaluation. The choice of enabled options has significant impact on the running times of the subsequently employed QBF solver, however. We also remark that some optimisations, e.g., input rewriting, introduce new atoms. Thus, not only auxiliary atoms used by a programmer but also such new atoms stemming from the grounding request the use of projection in equivalence tests. Note that by using DLV's intelligent grounder, we can use strong negation as well as integer arithmetics and comparison

predicates in the programs. The grounder translates rules containing these constructs into rules of form (2.1).

However, the optimisations of the intelligent grounder may be too excessive. For any program P, the intelligent grounder only guarantees that $grnd(P, HU_P)$ and the result of the grounder are ordinarily equivalent with projection on $At(grnd(P, HU_P))$. For example, the grounding of the program P for the heater in Figure 7.2 would result in the empty program since it contains no facts. Indeed, P and $\emptyset$ are ordinarily equivalent. We need a method that guarantees that the semantics of the ground program, possibly joined with atoms from $I(\sigma)$, is correctly preserved under the conservative assumption that the grounder only preserves ordinary equivalence (with projection to the language of the considered program).

Our concrete method to ground programs is as follows:

Definition 18 *Let P be a program and σ a specification. Then, $igrnd(P, \sigma)$ is the program that results from P by performing the following steps:*

1. *Augment P by rules $a \leftarrow a'$ and $a' \vee a''$ for any $a \in I(\sigma)$, where a' and a'' are globally new atoms.*
2. *Use the intelligent grounder of* `DLV` *to ground the augmented version of P.*
3. *Finally, delete all rules containing primed or double-primed atoms from the resulting ground program.*

Theorem 23 *Let P be program and σ a specification. Define $M = grnd(P, D)$, where D is a finite set containing all constant symbols that occur in $P \cup I(\sigma)$, and $N = igrnd(P, \sigma)$. Then, the PQEP $(M, N, \mathcal{F}_{I(\sigma)}, =_{HB_P})$ holds.*

Proof. Let P' be the program that results from P after the first step of the transformation described in Definition 18, and define P'' as the output of the intelligent grounder in the second step in Definition 18. Recall that the intelligent grounder preserves ordinary equivalence with projection on $V = At(grnd(P', HU_{P'}))$. Thus, the PQEP $(grnd(P', D), P'', \{\emptyset\}, =_V)$ holds. This implies, by Theorem 8, that the PQEP $(M, N, \mathcal{F}_{I(\sigma)}, =_D)$ holds as well. □

In order to apply $\mathtt{cc}\top$ for verifying correctness of a program P with respect to a specification σ, we follow Theorem 22 but make use of $igrnd(\cdot, \cdot)$ instead of $grnd(\cdot, \cdot)$:

1. we compute $M = igrnd(P \cup C(\sigma), D)$ and $N = igrnd(Ref(\sigma) \cup C(\sigma), D)$, where D is defined as in Theorem 22, and
2. we use $\mathtt{cc}\top$ (along with a QBF solver) to decide the PQEP $(M, N, \mathcal{F}_{I(\sigma)}, =_{O\sigma})$.

It follows directly from Theorems 23, 6, and 7 that P is correct with respect to σ iff the PQEP $(M, N, \mathcal{F}_{I(\sigma)}, =_{O\sigma})$ holds. However, programs resulting from this procedure are still very large in general. We thus restrict the sets $I(\sigma)$ to contain only certain relevant predicates in our verification application. For the heater specification σ_H from above for example, we restrict $I(\sigma_H)$ in such a way that not all temperature values from 0 to 60 are considered but only an interval around 45 since it is very likely that if a student program is not correct, then it will diverge from the specification on input from this interval.

Table 7.1: Results of the program verification.

Semester	Component	Number of instances	Classified as correct		Running-times (sec.)	
			Previous approach	cc⊤ approach	Average	Median
ws2006	c	50	44	38	0.9	1.0
	h	50	39	32	1.0	1.1
	s	50	29	22	0.4	0.1
	v	50	40	34	5.1	5.6
	all	50	42	32	70.2	103.0
ws2007	c	78	67	56	0.8	0.8
	h	78	69	59	0.6	0.6
	s	78	52	0	1.4	1.5
	v	78	48	8	4.5	2.9
	all	78	60	39	491.4	894.0
ws2008	c	100	54	40	1.3	2.3
	h	100	70	13	0.2	0.2
	s	100	59	28	1.4	3.0
	v	100	53	25	0.6	1.1
	all	100	52	19	132.3	72.5

7.4 Results

As already mentioned, we considered student data from three semesters. All experiments were carried out on a 3.0 GHz Quad Core Intel Xeon workstation with 33 GB of RAM and SuSE Linux version 10.3. We used the QBF solver `qpro` with encoding $\mathsf{T}[\cdot]$ as it turned out that all other solvers mentioned in the previous chapter showed a running-time behaviour several orders of magnitude worse than `qpro`'s. Concerning the setting for the intelligent grounder, we achieved best performance when the option for input rewriting was disabled. The reason is that this optimisation introduces new atoms which seems to be disadvantageous for `qpro`.

We also compared the outcomes of the equivalence tests with results from a test approach previously used in the course. In this test, sets of test cases are individually specified for each component and the whole system, where a test case for a component with specification σ is a subset of $I(\sigma)$ that is admissible with respect to σ. Then, we test whether a student's attempt to model the component by program P and our reference program $Ref(\sigma)$ yield the same answer sets—projected to $O(\sigma)$—when joined with such a test case. If this is the case, we say that a P *passes* a test case, otherwise P *fails* the test case. For each component, we specify a set of test cases that usually comprises 10 to 20 instances. As it turned out, many errors were undetected by the previous approach, thus it is rather prone to false-positives with respect to the verification task, i.e., P passes all its test cases, but still P is not correct with respect to σ.

Table 7.1 summarises the results of our experiments. We provide the year of the semester a course took place, the name of the component we considered, the number of instances of that

component, the number of instances classified as correct by the current approach, the number of instances classified as correct by our reduction approach, the average running times in seconds, as well as the median running time for solving the QBFs. A program is classified as correct by the current approach if it passes all its test cases. Components c, h, s, v denote the cooler, heater, switch, and valve as before, while "all" refers to the overall program consisting of all components, the encoding of the connections between them, and additional constraints required for diagnosing.

The ground programs for the component tests contain up to 985 rules. The number of atoms in the resulting QBFs ranges from 229 to 623. For the overall tests, programs contain up to 4818 rules, and the QBFs contain 949 to 3143 atoms. Table 7.1 shows that running times for the solved QBFs keep in reasonable bounds.

Note that whenever a program is classified as not correct by the current approach, then it is classified as not correct by the $\mathtt{cc}\top$ approach as well. Hence, the difference between the numbers of programs classified as correct by the two approaches is the number of false positives for the current approach. Coming as no surprise, the $\mathtt{cc}\top$ approach reveals significantly more incorrect solutions than the current approach. The reason that the number of correct overall programs is not smaller than the minimum number of its correct components is mainly due to different restrictions on what admissible input means. The two significantly small numbers of correct solutions for the switch and the valve component in the 'ws2007' test set is because of subtle differences between the reference and the student solutions in case some input values are missing which is informative for diagnosing. If this is considered to be too strict, one could simply exclude such cases by changing the admissibility constraints accordingly.

CHAPTER 8

Related Work

A related refined equivalence notion in the context of answer-set programming is *visible equivalence* between disjunctive logic programs, introduced by Janhunen and Oikarinen [41]. Visible equivalence is a form of ordinary equivalence with projection on what Janhunen and Oikarinen [41] call *visible atoms*. A major difference to the projection used in this paper is that, for visible equivalence, there has to be a bijection between the visible parts of the programs' answer sets. An equivalence problem of this form can be decided by a reduction to a logic program such that the latter has no answer sets iff the equivalence problem holds.

The system `dlpeq` [64] is capable of comparing disjunctive logic programs under ordinary equivalence. Similar to `cc⊤`, `dlpeq` is based on a translational approach. However, `dlpeq` reduces problems of ordinary equivalence to logic programs which is feasible since deciding problems of ordinary equivalence lies on the second level of the polynomial hierarchy [29]. Here, the reduction of a correspondence problem results in further logic programs such that the latter have no answer sets iff the encoded problem holds. In particular, `dlpeq` uses `GnT` [38] as answer-set solver. If the problem does not hold, the answer sets correspond to counterexamples. Hence, this system uses answer-set solvers themselves in order to check for equivalence.

We also mention the system `selp` [13] which checks strong equivalence by means of a reduction approach, very much in the spirit of our QBF approach. In particular, the problem of checking strong equivalence between disjunctive logic programs is reduced to propositional logic (recall that deciding strong equivalence is coNP-complete). Hence, the system makes use of SAT solvers as back-end inference engines instead of QBF solvers. Our system generalises `selp` in the sense that `cc⊤` handles a correspondence problem which coincides with a test for strong equivalence by the same reduction as used in SELP [55].

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 `cc⊤` provides a wide range of more fine-grained equivalence notions that allow practical comparisons useful for program verification, optimisation, debugging, and modular programming. The correspondence problems addressed by `dlpeq` and `selp` can be specified with `cc⊤` as special cases. For the special case of ordinary equivalence, we compared `cc⊤` against the system `dlpeq` in previous

work [55]. There, the used benchmarks rely on randomly generated $(2, \exists)$-QBFs. Each QBF is reduced to a program such that the latter possesses an answer set iff the original QBF is valid. The idea is to compare the program with itself having a randomly selected rule dropped, thus simulating a "sloppy" programmer. For details on the considered benchmark set, cf. Oikarinen and Janhunen [64]. The `dlpeq` approach turns out to be faster than the reduction approach to QBFs. Interestingly, among the tested QBF solvers, `qpro` showed to be the most competitive one while the PCNF-QBF solvers perform comparably bad even for small instances.

A further related equivalence notion is *update equivalence* [37]. For this notion, equivalence problems are defined in terms of two sets $\mathcal{Q}$ and $\mathcal{R}$ of programs: $\mathcal{Q}$ defines the sets of rules that might be deleted from the programs under consideration, and $\mathcal{Q}$ defines the set of rules that might be added to the programs. Hence, two programs P_1 and P_2 are update equivalent iff, for any $Q \subseteq \mathcal{Q}$ and any $R \subseteq \mathcal{R}$, $(P_1 \setminus R) \cup Q$ and $(P_2 \setminus R) \cup Q$ have the same answer sets. This notion captures also the notion of relativised uniform equivalence and thus uniform equivalence but does not incorporate projection.

Woltran [88] considered parameterised correspondence problems that allow to express relativised versions of both strong and uniform equivalence. In particular, this notion of equivalence, called *hyperequivalence*, relativises strong equivalence by two parameters: a head alphabet H and a body alphabet B. Two programs P and Q are equivalent under this notion iff $AS(P \cup R) = AS(Q \cup R)$, for each program R consisting of rules whose heads contain only atoms from H and whose bodies contain only atoms from B. It was shown that uniform and relativised uniform equivalence is indeed a special case of hyperequivalence, namely if $B = \emptyset$ [88]. However, projection on the answer sets was not considered which limits the relevance of hyperequivalence in practice. Truszczynski [82] investigated a generalisation of strong and uniform equivalence based on operators on complete lattices. As well, Pührer, Tompits, and Woltran [73] discuss necessary and sufficient conditions under which the elimination of disjunction, negation, or both, in programs is possible, preserving hyperequivalence.

Our work focuses on propositional programs only. Nevertheless, an important issue is the more general non-ground setting where programs are defined over a first-order language and are thus allowed to use variables ranging over a, in general, infinite domain. However, thereby we have to face undecidability which holds already for unrelativised uniform equivalence [27]. Note, however, that results for the propositional case are relevant for the non-ground case as well: Whenever the underlying domain can be finitely fixed—a reasonable assumption for many practical applications—non-ground programs can be considered as concise representation for their propositional counterparts that are obtained by grounding them over the finite domain. Recall that we illustrated an application of `cc`$\top$ involving non-ground programs in Chapter 7.

The more general case involving also infinite domains was considered by Oetsch and Tompits [61]. There, correspondence problems generalising PQIPs and PQEPs to the first-oder level are translated to formulas of second-order logic. It is also shown that referring to second-order logic is inescapable due to recursion-theoretic reasons. In this context, we also mention the work by Fink [32] who extended the notion of uniform equivalence to non-ground theories. There, characterisations using a first-order version of the logic of here-and-there were introduced. Note that strong equivalence between non-ground programs can be decided by a dedicated system that is based on a reduction to a decidable fragment of first-order logic [24].

Our approach to decide correspondence problems relies on efficient reductions to QBFs and to subsequently employ efficient QBF solvers to compute solutions. Similar reductions to the language of quantified propositional logic have been considered in the context of many problems whose complexity is beyond NP. Such problems in particular come from the realm of nonmonotonic reasoning, including default reasoning, disjunctive logic programs, autoepistemic logic, and propositional circumscription [18]. Furthermore, QBF reductions have been considered for argumentation [7], equilibrium logic and programs with nested expressions [69], belief revision [14, 15], paraconsistent reasoning [1, 2, 8], and planning [76, 77].

Regarding Chapter 7 on testing and verifying logic programs, we mention that structure-based testing and different notions of test-input coverage were considered for propositional programs in related work [39]. These testing methods were evaluated in a subsequent paper for testing non-ground programs [40]. To this end, the considered non-ground programs were grounded over some finite domain, similar to the verification approach outlined in this thesis.

CHAPTER 9

Conclusion and Future Work

In this thesis, we studied refined versions of uniform equivalence, called PQEPs, for disjunctive logic programs under the answer-set semantics. We also considered corresponding implication problems, which we call PQIPs, in which projective set inclusion is taken as basic comparison relation instead of projective set equality. Such correspondence problems allow to restrict the alphabet of the context class and facilitate the removal of auxiliary atoms in the comparison of programs—two important concepts for program comparisons in practice.

In particular, we provided novel model-theoretic characterisations in terms of wedges which, at the same time, yield new characterisation for (relativised) uniform equivalence (vis-a-vis UE-models). We also analysed the computational complexity of correspondence checking, revealing that deciding PQEPs and PQIPs is hard, viz. laying on the third level of the polynomial hierarchy. Finally, we described efficient reductions of PQIPs and PQEPs to QBFs and simplifications for certain special cases of correspondence problems. These reduction serve as basis for an extension of the system ccT for deciding PQIPs and PQEPs and have been incorporated into the system accordingly. This reduction approach to QBFs is motivated, on the one hand, by the high complexity of the correspondence problems at hand. On the other hand, we can use efficient QBF solvers as back-end engines for computing QBFs. We presented the architecture and system specifics of ccT for deciding PQEPs and PQIPs. In fact, the implemented translations include optimised versions in the sense that adequacy with respect to the structure of a resulting QBF and the intrinsic complexity of a correspondence problem is retained also for (relativised) uniform equivalence.

We complemented our more theoretical discussions 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. This analysis shows that discriminating among different back-end solvers for quantified propositional logic is a crucial issue towards optimal performance. Moreover, our encodings also provide a challenging benchmark set for QBF solvers for which there are only a few structured problems with more than one quantifier alternation available. We also showed how ccT can be used in a concrete application scenario that is concerned with the verification of programs. Although ccT processes propositional programs only, it can still be employed for program comparisons of non-ground programs by finitely fixing

a domain of interest.

There remain many issues for future work. For instance, we used PQEPs to verify programs that realise diagnostic reasoning in Chapter 7. For model-based diagnosis, however, native concepts of equivalence, directly defined in terms of a diagnosis problem, would be useful.

In case programs are not equivalent, a counterexample that gives information why the programs are not equivalent is often of great value. Although $\mathtt{cc}\top$ can be used to generate QBFs such that assignments for the open variables correspond to such counterexamples, few QBF solvers can directly compute such assignments. A different option to handle this issue in future work would be to make use of *certificates* for QBFs, where a certificate can be seen as a compact representation of a model of a QBF [5]. As some QBF solvers are able to produce certificates [6,51], these could be used for generating counterexamples from respective certificates if an equivalence problem does not hold.

A further open topic for future work is the extension of our results to more general classes of programs like, e.g., nested programs or arbitrary theories. Also, to deal with other language extensions like aggregates or optimisation statements seems to be a worthwhile effort since they are needed in many problem encodings. Often, non-ground programs are formulated over a language with an infinite domain. An important topic is to single out at least sufficient conditions when we can restrict this domain to a finite subdomain such that program equivalence over this subdomain implies equivalence over the unrestricted domain. Also, we plan to conduct experiments with more real-world oriented benchmarks, like ones stemming from planning or scheduling domains.

Bibliography

[1] Ofer Arieli. Paraconsistent preferential reasoning by signed quantified Boolean formulae. In *Proceedings of the 16th European Conference on Artificial Intelligence* (*ECAI 2004*), pages 773–777. IOS Press, 2004.

[2] Ofer Arieli and Marc Denecker. Reducing preferential paraconsistent reasoning to classical entailment. *Journal of Logic and Computation*, 13(4):557–580, 2003.

[3] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, Cambridge, England, 2003.

[4] Chitta Baral and Michael L. Fredman. Reasoning agents in dynamic domains. In Jack Minker, editor, *Logic-based Artificial Intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[5] Marco Benedetti. Extracting certificates from quantified boolean formulas. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI 2005*), pages 47–53. Morgan Kaufmann Publishers Inc., 2005.

[6] Marco Benedetti. sKizzo: A suite to evaluate and certify QBFs. In *Proceedings of the 20th International Conference on Automated Deduction* (*CADE 2005*), volume 3632 of *Lecture Notes in Computer Science*, pages 369–376. Springer, 2005.

[7] Philippe Besnard, Anthony Hunter, and Stefan Woltran. Encoding deductive argumentation in quantified Boolean formulae. *Artificial Intelligence*, 173(15):1406–1423, 2009.

[8] Philippe Besnard, Torsten Schaub, Hans Tompits, and Stefan Woltran. Representing paraconsistent reasoning via quantified propositional logic. In *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 84–118. Springer, 2005.

[9] Armin Biere. Resolve and expand. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing* (*SAT 2004*), volume 3542 of *Lecture Notes in Computer Science*, pages 59–70. Springer, 2005.

[10] Martin Brain, Tom Crick, Marina De Vos, and John Fitch. TOAST: Applying answer set programming to superoptimisation. In *Proceedings of the 22nd International Conference on Logic Programming* (*ICLP 2006*), volume 4079 of *Lecture Notes in Computer Science*, pages 270–284. Springer, 2006.

[11] Daniel R. Brooks, Esra Erdem, Selim T. Erdogan, James W. Minett, and Donald Ringe. Inferring phylogenetic trees using answer set programming. *Journal of Automated Reasoning*, 39(4):471–511, 2007.

[12] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The Third Answer Set Programming Competition: Preliminary report of the system competition track. In *Proceedings of the 11th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2011*), volume 6645 of *Lecture Notes in Computer Science*, pages 388–403. Springer, 2011.

[13] Yin Chen, Fangzhen Lin, and Lei 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 *Lecture Notes in Artificial Intelligence*, pages 442–446. Springer, 2005.

[14] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran. On computing belief change operations using quantified Boolean formulas. *Journal of Logic and Computation*, 14(6):801–826, 2004.

[15] James P. Delgrande, Torsten Schaub, Hans Tompits, and Stefan Woltran. Belief revision of logic programs under answer set semantics. *Transactions on Computational Logic*, to appear.

[16] Marc Denecker, Joost Vennekens, Stephen Bond, Martin Gebser, and Miroslaw Truszczynski. The Second Answer Set Programming Competition. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2009*), volume 5753 of *Lecture Notes in Computer Science*, pages 637–654. Springer, 2009.

[17] DLV system. `www.dlvsystem.com`, last visited: July 20, 2011.

[18] Uwe Egly, Thomas Eiter, Hans Tompits, and Stefan Woltran. Solving advanced reasoning tasks using quantified Boolean formulas. In *Proceedings of the 7th Conference on Artificial Intelligence* (*AAAI 2000*) *and of the 12th Conference on Innovative Applications of Artificial Intelligence* (*IAAI 2000*), pages 417–422. AAAI Press, 2000.

[19] Uwe Egly, Martina Seidl, Hans Tompits, Stefan Woltran, and Michael 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 *Lecture Notes in Computer Science*, pages 214–228. Springer, 2004.

[20] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in nonprenex form. In *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI 2006*), pages 477–481. IOS Press, 2006.

[21] Uwe Egly, Martina Seidl, and Stefan Woltran. A solver for QBFs in negation normal form. *Constraints*, 14(1):38–79, 2009.

[22] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. The diagnosis frontend of the DLV system. *AI Communications*, 12(1-2):99–111, 1999.

[23] Thomas Eiter, Wolfgang Faber, Nicola Leone, Gerald Pfeifer, and Axel Polleres. Planning under incomplete knowledge. In *Proceedings of the 1st International Conference on Computational Logic* (*CL 2000*), volume 1861 of *Lecture Notes in Computer Science*, pages 807–821. Springer, 2000.

[24] Thomas Eiter, Wolfgang Faber, and Patrick Traxler. Testing strong equivalence of Datalog programs - Implementation and examples. In *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2005*), volume 3662 of *Lecture Notes in Computer Science*, pages 437–441. Springer, 2005.

[25] Thomas Eiter and Michael 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 *Lecture Notes in Computer Science*, pages 224–238. Springer, 2003.

[26] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying logic programs under uniform and strong equivalence. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2004*), volume 2923 of *Lecture Notes in Computer Science*, pages 87–99. Springer, 2004.

[27] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In *Proceedings of the 20th National Conference on Artificial Intelligence* (*AAAI 2005*), pages 695–700. AAAI Press, 2005.

[28] Thomas Eiter, Michael Fink, and Stefan Woltran. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Transactions on Computational Logic*, 8(3), 2007.

[29] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3/4):289–323, 1995.

[30] Thomas Eiter, Hans Tompits, and Stefan Woltran. On solution correspondences in answer set programming. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (*IJCAI 2005*), pages 97–102. Professional Book Center, 2005.

[31] Esra Erdem, Vladimir Lifschitz, and Donald Ringe. Temporal phylogenetic networks and logic programming. *Theory and Practice of Logic Programming*, 6(5):539–558, 2006.

[32] Michael Fink. Equivalences in answer-set programming by countermodels in the logic of here-and-there. In *Proceedings of the 24th International Conference on Logic Programming* (*ICLP 2008*), volume 5366 of *Lecture Notes in Computer Science*, pages 99–113. Springer, 2008.

[33] Martin Gebser, Benjamin Kaufmann, André Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (*IJCAI 2007*), pages 386–392. AAAI Press/MIT Press, 2007.

[34] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[35] Enrico Giunchiglia, Massimo Narizzano, and Armando Tacchella. Backjumping for quantified Boolean logic satisfiability. *Artificial Intelligence*, 145:99–120, 2003.

[36] S. Grell, T. Schaub, and J. Selbig. Modelling biological networks by action languages via answer set programming. In *Proceedings of the 22nd International Conference on Logic Programming* (*ICLP 2006*), volume 4079 of *Lecture Notes in Computer Science*, pages 285–299. Springer, 2006.

[37] Katsumi Inoue and Chiaki Sakama. Equivalence of logic programs under updates. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence* (*JELIA 2004*), volume 3229 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2004.

[38] Tomi Janhunen, Ilka Niemelä, Dietmar Seipel, and Patrik Simons. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[39] Tomi Janhunen, Ilkka Niemelä, Johannes Oetsch, Jörg Pührer, and Hans Tompits. On testing answer-set programs. In *Proceedings of the 19th European Conference on Artificial Intelligence* (*ECAI 2010*), volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 951–956. IOS Press, 2010.

[40] Tomi Janhunen, Ilkka Niemelä, Johannes Oetsch, Jörg Pührer, and Hans Tompits. Random vs. structure-based testing of answer-set programs: An experimental comparison. In *Proceedings of the 11th International Conference Logic Programming and Nonmonotonic Reasoning* (*LPNMR 2011*), volume 6645 of *Lecture Notes in Computer Science*, pages 242–247. Springer, 2011.

[41] Tomi Janhunen and Emilia Oikarinen. Automated verification of weak equivalence within the SMODELS system. *Theory and Practice of Logic Programming*, 7(4):1–48, 2007.

[42] Volker Klotz. *The System QUIP: A Solver for Advanced Reasoning Tasks using Quantified Boolean Formulas*. Master's thesis, Vienna University of Technology, Vienna, Austria, 2001.

[43] Daniel Le Berre, Massimo Narizzano, Laurent Simon, and ArmandoTacchella. The Second QBF Solvers Comparative Evaluation. In *Proceedings of the 7th International Conference*

on Theory and Applications of Satisfiability Testing (*SAT 2004*). *Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 376–392. Springer, 2005.

[44] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[45] Reinhold Letz. Lemma and model caching in decision procedures for quantified Boolean formulas. In *Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods* (*TABLEAUX 2002*), volume 2381 of *Lecture Notes in Computer Science*, pages 160–175. Springer, 2002.

[46] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[47] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In *Proceedings of the 11th International Conference on Logic Programming* (*ICLP 1994*), pages 23–37. MIT Press, 1994.

[48] Fangzhen Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2002*), pages 170–176. Morgan Kaufmann, 2002.

[49] Fangzhen Lin and Yin Chen. Discovering classes of strongly equivalent logic programs. *Journal of Artificial Intelligence Research*, 28:431–451, 2007.

[50] Albert R. Meyer and Larry J. Stockmeyer. Word problems requiring exponential time. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing*, pages 1–9. ACM Press, 1973.

[51] Massimo Narizzano, Claudia Peschiera, Luca Pulina, and Armando Tacchella. Evaluating and certifying QBFs: A comparison of state-of-the-art tools. *AI Communications*, 22(4):191–210, 2009.

[52] Massimo Narizzano, Luca Pulina, and Armando Tacchella. Report of the Third QBF Solvers Evaluation. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):145–164, 2006.

[53] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog decision support system for the Space Shuttle. In *Answer Set Programming: Towards Efficient and Scalable Knowledge Represenation and Reasoning*, pages 139–145. AAAI Press, 2001.

[54] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. ccT: A correspondence-checking tool for logic programs under the answer-set semantics. In *Proceedings of the 10th European Conference on Logics in Artificial Intelligence* (*JELIA 2006*), volume 4160 of *Lecture Notes in Computer Science*, pages 502–505. Springer, 2006.

[55] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan 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.

[56] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. ccT: A tool for checking advanced correspondence problems in answer-set programming. In *Proceedings of the 1st International Workshop on Logic and Search* (*LaSh 2006*), 2006.

[57] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. A tool for advanced correspondence checking in answer-set programming. In *Proceedings of the 11th International Workshop on Nonmonotonic Reasoning* (*NMR 2006*), volume IfI-06-04 of *IfI Technical Report Series*, pages 20–28. Institut für Informatik, Technische Universität Clausthal, 2006.

[58] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. A tool for advanced correspondence checking in answer-set programming: Preliminary experimental results. In *Proceedings of the 20th Workshop on Logic Programming* (*WLP 2006*), volume 1843-06-02 of *INFSYS Research Report*, pages 200–205. Technische Universität Wien, Austria, 2006.

[59] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. Testing relativised uniform equivalence under answer-set projection in the system ccT. In *Proceedings of the 17th International Conference on Applications of Declarative Programming and Knowledge Management* (*INAP 2007*) *and the 21st Workshop on Logic Programming* (*WLP 2007*), volume 5437 of *Lecture Notes in Computer Science*, pages 241–246. Springer, 2007.

[60] Johannes Oetsch, Martina Seidl, Hans Tompits, and Stefan Woltran. ccT on stage: Generalised uniform equivalence testing for verifying student assignment solutions. In *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning* (*ICLP 2009*), volume 5753 of *Lecture Notes in Computer Science*, pages 382–395. Springer, 2009.

[61] Johannes Oetsch and Hans Tompits. Program correspondence under the answer-set semantics: The non-ground case. In *Proceedings of the 24th International Conference on Logic Programming* (*ICLP 2008*), volume 5366 of *Lecture Notes in Computer Science*, pages 591–605. Springer, 2008.

[62] Johannes Oetsch, Hans Tompits, and Stefan Woltran. Facts do not cease to exist because they are ignored: Relativised uniform equivalence with answer-set projection. In *Proceedings of the Workshop on Correspondence and Equivalence for Nonmonotonic Theories* (*CENT 2007*), pages 25–36, 2007.

[63] Johannes Oetsch, Hans Tompits, and Stefan 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.

[64] Emilia Oikarinen and Tomi 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 *Lecture Notes in Computer Science*, pages 180–193. Springer, 2004.

[65] Emilia Oikarinen and Tomi Janhunen. Modular equivalence for normal logic programs. In *Proceedings of the 17th European Conference on Artificial Intelligence* (*ECAI 2006*), pages 412–416. IOS Press, 2006.

[66] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[67] David Pearce. Simplifying logic programs under answer set semantics. In *Proceedings of the 20th International Conference on Logic Programming* (*ICLP 2004*), volume 3132 of *Lecture Notes in Computer Science*, pages 210–224. Springer, 2004.

[68] David Pearce, Hans Tompits, and Stefan Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In *Proceedings of the 10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving* (*EPIA 2001*), volume 2258 of *Lecture Notes in Computer Science*, pages 306–320. Springer, 2001.

[69] David Pearce, Hans Tompits, and Stefan Woltran. Characterising equilibrium logic and nested logic programs: Reductions and complexity. *Theory and Practice of Logic Programming*, 9(5):565–616, 2009.

[70] David A. Plaisted and Steven Greenbaum. A structure preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[71] Axel Polleres. Semantic web languages and semantic web services as application areas for answer set programming. In *Nonmonotonic Reasoning, Answer Set Programming and Constraints*, number 05171 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[72] Potassco: The Potsdam Answer Set Solving Collection Homepage. `http://potassco.sourceforge.net`, last visited: July 18, 2011.

[73] Jörg Pührer, Hans Tompits, and Stefan Woltran. Elimination of disjunction and negation in answer-set programs under hyperequivalence. In *Proceedings of the 24th International Conference on Logic Programming* (*ICLP 2008*), volume 5366 of *Lecture Notes in Computer Science*, pages 561–575. Springer, 2008.

[74] QBFLIB Homepage. `www.qbflib.or`, last visited: July 27, 2011.

[75] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[76] Jussi Rintanen. Constructing conditional plans by a theorem prover. *Journal of Artificial Intelligence Research*, 10:323–352, 1999.

[77] Jussi Rintanen. Asymptotically optimal encodings of conformant planning in QBF. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence* (*AAAI 2007*), pages 1015–1050. AAAI Press, 2007.

[78] Yehoshua Sagiv. Optimizing Datalog programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 659–698. Morgan Kaufmann, 1988.

[79] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In *Proceedings of the 1st International Workshop on Practical Aspects of Declarative Languages* (*PADL 1999*), volume 1551 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 1999.

[80] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[81] Hans Tompits and Stefan 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 *Lecture Notes in Computer Science*, pages 189–203. Springer, 2005.

[82] Miroslaw Truszczynski. Strong and uniform equivalence of nonmonotonic theories - An algebraic approach. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning* (*KR 2006*), pages 389–399. AAAI Press, 2006.

[83] Grighori S. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part II*, pages 234–259. Seminars in Mathematics, V.A. Steklov Mathematical Institute, vol. 8, 1968. English translation: Consultants Bureau, New York, 1970, pp. 115–125.

[84] Alan Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936.

[85] Hudson Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4-5):602–622, 2003.

[86] Stefan Woltran. *Quantified Boolean Formulas—From Theory to Practice*. Dissertation, Vienna University of Technology, Vienna, Austria, 2003.

[87] Stefan 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 *Lecture Notes in Computer Science*, pages 161–173. Springer, 2004.

[88] Stefan Woltran. A common view on strong, uniform, and other notions of equivalence in answer-set programming. *Theory and Practice of Logic Programming*, 8(2):217–234, 2008.

[89] Michael Zolda. Comparing different prenexing strategies for quantified boolean formulas, 2004. Master's Thesis, Vienna University of Technology.