Testing Relativised Uniform Equivalence under Answer-Set Projection in the System cc$\top$[⋆]

Johannes Oetsch[1], Martina Seidl[2], Hans Tompits[1], and Stefan Woltran[1]

[1] Institut für Informationssysteme, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
{oetsch,tompits}@kr.tuwien.ac.at
woltran@dbai.tuwien.ac.at

[2] Institut für Softwaretechnik, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
seidl@big.tuwien.ac.at

Abstract. The system cc$\top$ is a tool for testing correspondence between logic programs under the answer-set semantics with respect to different refined notions of program correspondence. The underlying methodology of cc$\top$ is to reduce a given correspondence problem to the satisfiability problem of quantified propositional logic and to employ extant solvers for the latter language as back-end inference engines. In a previous version of cc$\top$, the system was designed to test correspondence between programs based on *relativised strong equivalence under answer-set projection*. Such a setting generalises the standard notion of strong equivalence by taking the alphabet of the context programs as well as the projection of the compared answer sets to a set of designated output atoms into account. This paper outlines a newly added component of cc$\top$ for testing similarly parameterised correspondence problems based on *uniform equivalence*.

1 General Information

An important issue in software development is to determine whether two encodings of a given problem are equivalent, i.e., whether they yield the same result on a given problem instance. Depending on the context of problem representations, different definitions of "equivalence" are useful and desirable. The system cc$\top$ [1] (short for "correspondence-checking tool") is devised as a checker for a broad range of different such comparison relations defined between *disjunctive logic programs* (DLPs) *under the answer-set semantics* [2]. In a previous version of cc$\top$, the system was designed to test correspondence between logic programs based on *relativised strong equivalence under answer-set projection*. Such a setting generalises the standard notion of strong equivalence [3] by taking the alphabet of the context programs as well as the projection of the compared answer sets to a set of designated output atoms into account [4]. The latter feature

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

reflects the common use of local (hidden) variables which may be used in submodules but which are ignored in the final computation.

In this paper, we outline a newly added component of $\mathtt{cc}\top$ for testing similarly parameterised correspondence problems but generalising *uniform equivalence* [5]—that is, we deal with a component of $\mathtt{cc}\top$ for testing *relativised uniform equivalence under answer-set projection*. This notion, recently introduced in previous work [6], is less restrained, along with a slightly lower complexity than its strong pendant. However, in general, it is still outside a feasible means to be computed by answer-set solvers (provided that the polynomial hierarchy does not collapse). Yet, like relativised strong equivalence with projection, it can be efficiently reduced to the satisfiability problem of quantified propositional logic, an extension of classical propositional logic characterised by the condition that its sentences, generally referred to as *quantified Boolean formulas* (QBFs), are permitted to contain quantifications over atomic formulas. The architecture of $\mathtt{cc}\top$ takes advantage of this and uses existing solvers for quantified propositional logic as back-end reasoning engines.

2 System Specifics

The equivalence notions under consideration are defined for ground disjunctive logic programs with default negation under the answer-set semantics [2]. Let $AS(P)$ be the collection of the answer sets of a program P. Two programs, P and Q, are *strongly equivalent* iff, for any program R, $AS(P \cup R) = AS(Q \cup R)$; they are *uniformly equivalent* iff, for any set F of facts, $AS(P \cup F) = AS(Q \cup F)$. While strong equivalence is relevant for program optimisation and modular programming in general [7–9], uniform equivalence is useful in the context of hierarchically structured program components, where lower-layered components provide input for higher-layered ones. In abstracting from strong and uniform equivalence, Eiter *et al.* [4] introduced the notion of a *correspondence problem* which allows to specify (i) a *context*, i.e., a class of programs used to be added to the programs under consideration, and (ii) the relation that has to hold between the answer sets of the extended programs. The concrete formal realisation of relativised uniform equivalence with projection is as follows [6]: Consider a quadruple $\Pi = (P, Q, 2^A, \odot_B)$, where P, Q are programs, A, B are sets of atoms, $\odot \in \{\subseteq, =\}$, and $\mathcal{S} \;\odot_B\; \mathcal{S}'$ stands for $\{I \cap B \mid I \in \mathcal{S}\} \odot \{J \cap B \mid J \in \mathcal{S}'\}$. Then, *$\Pi$ holds* iff, for each $F \in 2^A$, $AS(P \cup F) \odot_B AS(Q \cup F)$. Furthermore, Π is called a *propositional query equivalence problem* (PQEP) if $\odot_B$ is given by $=_B$, and a *propositional query inclusion problem* (PQIP) if $\odot_B$ is given by $\subseteq_B$. Note that $(P, Q, 2^A, =_B)$ holds iff $(P, Q, 2^A, \subseteq_B)$ and $(Q, P, 2^A, \subseteq_B)$ jointly hold.

For illustration, consider the programs

$$P = \{\mathit{sad} \vee \mathit{happy} \leftarrow;\; \mathit{sappy} \leftarrow \mathit{sad}, \mathit{happy};\; \mathit{confused} \leftarrow \mathit{sappy}\},$$
$$Q = \{\mathit{sad} \leftarrow \mathrm{not}\ \mathit{happy};\; \mathit{happy} \leftarrow \mathrm{not}\ \mathit{sad};\; \mathit{confused} \leftarrow \mathit{sad}, \mathit{happy}\},$$

which express some knowledge about the "moods" of a person, where P uses an auxiliary atom sappy. The programs can be seen as queries over a propositional database which consists of facts from, e.g., $\{\mathit{happy}, \mathit{sad}\}$. For the output, it would be natural to consider the common intensional atom $\mathit{confused}$. We thus consider $\Pi = (P, Q,$

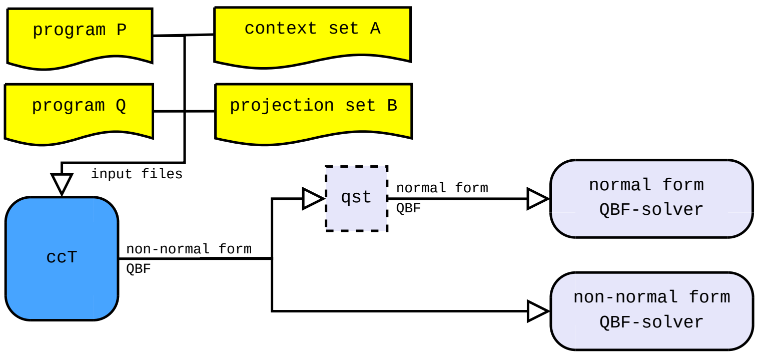

Fig. 1. Overall architecture of $\mathtt{cc}\top$.

$2^A, =_B)$ 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.

As pointed out in Section 1, the overall approach of $\mathtt{cc}\top$ is to reduce PQEPs and PQIPs to the satisfiability problem of quantified propositional logic and to use extant solvers for the latter language [10] as back-end inference engines for evaluating the resulting formulas. The reductions required for this approach are described by Oetsch *et al.* [6] but $\mathtt{cc}\top$ employs additional optimisations [11]. We note that quantified propositional logic is an extension of classical propositional logic in which sentences are permitted to contain quantifications over atomic formulas. It is standard custom to refer to the formulas of this language as *quantified Boolean formulas* (QBFs).

The overall architecture of $\mathtt{cc}\top$ is depicted in Figure 1. The system takes as input two programs, P and Q, and two sets of atoms, A and B. Command-line options select between two kinds of reductions, a direct one or an optimised one, and whether the programs are compared as a PQIP or a PQEP. Detailed invocation syntax can be requested with option `-h`. The syntax of the programs is the basic `DLV` syntax.[1] Since $\mathtt{cc}\top$ does not output QBFs in a specific normal form, for using solvers requiring normal-form QBFs, the additional normaliser `qst` [12] is employed. Finally, $\mathtt{cc}\top$ is developed entirely in ANSI C; hence, it is highly portable. The parser for the input data was written using LEX and YACC. Further information about $\mathtt{cc}\top$ is available at

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

Experimental evaluations using different QBF solvers are reported in a companion paper [11].

References

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

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

2. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing **9** (1991) 365–385
3. Lifschitz, V., Pearce, D., Valverde, A.: Strongly Equivalent Logic Programs. ACM Transactions on Computational Logic **2** (2001) 526–541
4. Eiter, T., Tompits, H., Woltran, S.: On Solution Correspondences in Answer Set Programming. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005). (2005) 97–102
5. Eiter, T., Fink, M.: Uniform Equivalence of Logic Programs under the Stable Model Semantics. In: Proceedings of the 19th International Conference on Logic Programming (ICLP 2003). Volume 2916 in Lecture Notes in Computer Science. Springer (2003) 224–238
6. Oetsch, J., Tompits, H., Woltran, S.: 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), AAAI Press (2007) 458–464
7. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying Logic Programs Under Uniform and Strong Equivalence. In Lifschitz, V., Niemelä, I., eds.: Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7). Volume 2923 of Lecture Notes in Computer Science. Springer Verlag (2004) 87–99
8. Pearce, D.: 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. Springer (2004) 210–224
9. Lin, F., Chen, Y.: Discovering Classes of Strongly Equivalent Logic Programs. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005). (2005) 516–521
10. Le Berre, D., Narizzano, M., Simon, L., Tacchella, L.A.: 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., Springer (2005) 376–392
11. Oetsch, J., Seidl, M., Tompits, H., Woltran, S.: An Extension of the System cc$\top$ for Testing Relativised Uniform Equivalence under Answer-Set Projection (2007). Submitted draft
12. Zolda, M.: Comparing Different Prenexing Strategies for Quantified Boolean Formulas (2004). Master's Thesis, Vienna University of Technology