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 propositional 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 Motivation and 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 $\texttt{cc}\top$ for testing similarly parameterised correspondence problems but generalising *uniform equivalence* [5]—that is, we deal with a component of $\texttt{cc}\top$ 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 counterpart (provided that the polynomial hierarchy does not collapse). However, in general, it is still outside a feasible means to be computed by propositional answer-set solvers (again under the proviso 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 $\texttt{cc}\top$ takes advantage of this and uses extant solvers for quantified propositional logic as back-end reasoning engines.

2 Background

Propositional disjunctive logic programs (DLPs) are finite sets of rules of the form

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

$n \geq m \geq l \geq 0$, where all a_i are propositional atoms from some fixed universe $\mathcal{U}$ and not denotes default negation. An *interpretation* I is a *model* of a program P, denoted by $I \models P$, iff for every rule from P (as defined above), it holds that, whenever $\{a_{l+1}, \ldots, a_m\} \subseteq I$ and $\{a_{m+1}, \ldots, a_n\} \cap I = \emptyset$, then $\{a_1, \ldots, a_l\} \cap I \neq \emptyset$.

Following Gelfond and Lifschitz [2], an interpretation I is an *answer set* of a program P iff it is a minimal model of the *reduct* P^I, resulting from P by (i) deleting all rules containing default negated atoms $\mathrm{not}\ a$ such that $a \in I$, and (ii) deleting all default negated atoms in the remaining rules. The collection of all answer sets of a program P is denoted by $\mathcal{AS}(P)$.

In order to semantically compare programs, different notions of equivalence have been introduced in the context of the answer-set semantics. 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 comparison relation that has to hold between the answer sets of the extended programs. Following Eiter *et al.* [4], we focus here on correspondence problems where the comparison relation is a projection (to a given set of atoms) of the standard subset or set-equality

relation. The context, on the other hand, contains all programs which are sets of facts over some set A of atoms, identified with the power set 2^A over A.

Thus, 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 $\{I \cap B \mid I \in \mathcal{S}\} \odot \{J \cap B \mid J \in \mathcal{S}'\}$. 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 $\subseteq_B$. We say that Π *holds* iff, for each $F \in 2^A$, $AS(P \cup F) \odot_B AS(Q \cup F)$. Note that $(P, Q, 2^A, =_B)$ holds iff $(P, Q, 2^A, \subseteq_B)$ and $(Q, P, 2^A, \subseteq_B)$ jointly hold. We also refer to A as the *context set* and to B as the *projection set*.

For illustration, consider the programs

$$P = \{sad \vee happy \leftarrow;\ sappy \leftarrow sad, happy;\ confused \leftarrow sappy\} \quad \text{and}$$
$$Q = \{sad \leftarrow \text{not } happy;\ happy \leftarrow \text{not } sad;\ confused \leftarrow sad, happy\},$$

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., $\{happy, sad\}$. For the output, it would be natural to consider the common intensional atom $confused$. We thus consider $\Pi = (P, Q, 2^A, =_B)$ 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.

3 System Specifics

As pointed out in Section 1, the overall approach of cc$\top$ is to reduce PQEPs and PQIPs to the satisfiability problem of quantified propositional logic and to use extant solvers [10] for the latter language as back-end inference engines for evaluating the resulting formulas. The reductions required for this approach are described by Oetsch *et al.* [6] but cc$\top$ employs additional optimisations [11]. The overall architecture of cc$\top$ is depicted in Fig. 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`.

Next, let us turn our attention to the concrete usage of cc$\top$. The syntax of the programs is the basic `DLV` syntax.[3] In this syntax, the two programs P and Q from the above example look as follows:

$$P = \begin{cases} \texttt{sad v happy.} \\ \texttt{sappy :- sad, happy.} \\ \texttt{confused :- sappy.} \end{cases} \qquad Q = \begin{cases} \texttt{sad :- not happy.} \\ \texttt{happy :- not sad.} \\ \texttt{confused :- sad, happy.} \end{cases}$$

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

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

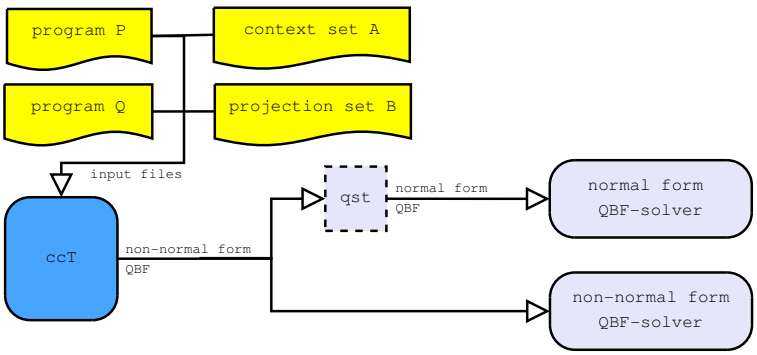

Fig. 1. Overall architecture of ccT.

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

We assume them to be stored in files `A` and `B`. The concrete invocation syntax for translating the problem $\Pi = (P, Q, 2^A, =_B)$ into a corresponding QBF is

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

where the command-line options '`-u`' and '`-e`' evince that we want to check for a notion of *u*niform *e*quivalence. To check for uniform inclusion, replace '`-e`' by '`-i`' or omit the parameter.

The output will be written directly to the standard-output device from where it can serve as input for QBF-solvers. Since ccT does not output QBFs in a specific normal form, for using solvers requiring normal-form QBFs, the additional normaliser `qst` [12] is employed. Finally, ccT 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 ccT is available at

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

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

4 ccT on Stage

In this section, we give a brief and, for space reasons, rather informal discussion on an application of ccT for verification and debugging needs in the context of a logic programming course at our university. This is not only to make the concept of correspondence checking within a refined framework more tangible but also to show a concrete application field. As a subtask in this course, the students have to model an air-conditioning system consisting of components for cooling and heating, as well as a valve and a switch element. More specifically, they are given a detailed description of the desired input/output behaviour of the components and the system as a whole, and

they have to develop logic programs that comply with that specification. Without going into the details, such a specification could demand that the input of, e.g., a heating component consists of an airstream which can be 0 (air does not float) or 1 (air floats) and has an associated temperature (an integer from a certain range) as well as a control parameter (also an integer) to control the heating power. Analogously to the input airstream, a heater has an output airstream. Now, the specification determines the behaviour of the component with respect to the output airstream conditioned by the input airstream and the control parameter.

A straightforward strategy to verify the student's solution is the following: (i) write a sample solution that correctly implements the specification, (ii) define test cases, i.e., sets of facts representing the input for a component, and (iii) compare the output of our sample solution against the output of the student's component. This method, used in previous years and implemented by a more or less simple script, is obviously sound but not complete with respect to detecting potential flaws. Here comes cc$\top$ into the play: this verification problem can be stated as a PQEP[4], where the context set consist of the atoms that constitute the input and the projection set contains the atoms that represent the output of a component (thus allowing the students an unrestricted use of additional atoms in their programs). Hence, we have a sound and complete method for verification at hand. We employed this approach last winter semester for evaluating the submitted exercises and it compared favourably to the old method.

Two things were necessary to obtain reasonable run-times for evaluating the QBFs, however: First, we had to restrict the context class, and second, we added additional constraints to the programs to impose some restrictions on the input of the components, like specifying not more than one input value for an airstream temperature. The later point is also to make the test more fair. Albeit we loose completeness in the sense from above this way, we are able to verify thousands of test cases implicitly with the cc$\top$-approach compared to only 10 to 20 test cases with the old script. Also, a direct comparison of the results between the two test approaches is very encouraging: all errors detected by the script were also detected by the cc$\top$-approach, while 26 (out of 200) components were classified as correct by the script but as non-equivalent to our sample solution by cc$\top$. It is worth mentioning that on these problem instances the solver `qpro` [13] showed, with a median run-time of 2.54 seconds, excellent performance.

5 Conclusion

In this paper, we presented the architecture and system specifics of a new component of the tool `cc`$\top$ for testing parameterised correspondence problems based on *uniform equivalence* for disjunctive logic programs under the answer-set semantics. The correspondence problems are efficiently compiled to quantified Boolean formulas for which many solvers have been implemented. As related work, we mention the system DLPEQ [14] for deciding ordinary equivalence, which is based on a reduction to logic programs, and the system SELP [15] for checking strong equivalence, which is based

[4] The programs under consideration are not propositional, i.e., they contain variables. Nevertheless, the domain, i.e., the set of constants that can occur in the programs is finite and such programs can always be treated as a shorthand for the respective propositional programs.

on a reduction to classical logic quite in the spirit of our implementation approach. We successfully applied $\mathtt{cc}\top$ for the verification of students' programs obtained from a laboratory course about logic programming at our university. Future work includes extending our methods to non-ground answer set programs, which are important in practical applications.

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
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 TOCL **2**(4) (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 of LNCS, 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 Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7). Volume 2923 of LNCS, Springer (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 LNCS, 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). Volume 3542 of LNCS, Springer (2005) 376–392
11. Oetsch, J., Seidl, M., Tompits, H., Woltran, S.: An Extension of the System $\mathtt{cc}\top$ 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.
13. Egly, U., Seidl, M., Woltran, S.: A Solver for QBFs in Nonprenex Form. In Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006). (2006) 477–481
14. Oikarinen, E., Janhunen, T.: Verifying the Equivalence of Logic Programs in the Disjunctive Case. In Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7). Volume 2923 of LNCS, Springer (2004) 180–193
15. Chen, Y., Lin, F., Li, L.: SELP - A System for Studying Strong Equivalence Between Logic Programs. In Proceedings of the 8th International Conference on Logic Programming and Non Monotonic Reasoning (LPNMR 2005). Volume 3552 of LNCS, Springer (2005) 442–446