LPEQ and DLPEQ — Translators for Automated Equivalence Testing of Logic Programs*

Tomi Janhunen and Emilia Oikarinen

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
Tomi.Janhunen@hut.fi, Emilia.Oikarinen@hut.fi

1 Introduction

Answer set programming (ASP) is a logic programming paradigm where a problem at hand is solved in a declarative fashion by (i) writing a logic program whose *answer sets* correspond to the solutions of the problem; and by (ii) computing the answer sets of the program using a special purpose search engine. Despite the declarative nature of ASP, the development of logic programs resembles that in conventional programming languages. That is, a programmer often develops a series of improved programs for a particular problem, e.g., when optimizing the execution time and space. As a consequence, there is a need to ensure that subsequent programs, which differ in performance, yield the same output. In ASP, this leads us to the problem of checking whether given two logic programs P and Q give rise to the same answer sets. This notion corresponds to the *weak equivalence* of the programs P and Q, denoted by $P \equiv Q$ in the sequel.

A brute-force method to verify $P \equiv Q$ is to enumerate all answer sets of P and to check that each of them is also an answer set of Q, and vice versa. This approach is called NAIVE in [6, 9]. Cross-checking answer sets in this way is time-consuming if the number of answer sets is high. To address this problem, we have developed translation-based methods for the automated verification of weak equivalence [6, 9]. The rough idea is to translate programs P and Q into another logic program whose answer sets capture potential counter-examples to $P \equiv Q$. Then existing ASP implementations can be used to check the existence of counter-examples and special-purpose search engines need not be developed.

There are also other notions of equivalence that have been proposed for logic programs. Lifschitz el al. [7] consider P and Q to be *strongly equivalent*, denoted by $P \equiv_{\mathrm{s}} Q$, if and only if $P \cup R \equiv Q \cup R$ for all other programs R. Because of the arbitrary context R involved in the definition, checking $P \equiv_{\mathrm{s}} Q$ appears to be a more complicated task than checking $P \equiv Q$. Fortunately, the problem of covering all possible contexts can be circumvented using a characterization

* This research is affiliated with the project "Applications of Rule-Based Constraint Programming" (#53695) funded by the Academy of Finland.

of strong equivalence based on *SE-models* [11]. Inspired by this result, we have implemented a two-phased method for the automated verification of $P \equiv_{\mathrm{s}} Q$.

Our implementations, translators LPEQ [3] and DLPEQ [8], cover equivalence testing as follows: (i) weak equivalence of weight constraint programs [6] under a generalization of the stable model semantics [10]; (ii) classical equivalence of normal logic programs; (iii) strong equivalence [7, 11] of normal logic programs under the stable model semantics [1]; and (iv) weak equivalence of disjunctive logic programs [9, 8] supported by the GNT system [5] under the stable model semantics [2]. Many other syntactic extensions can be covered via translations and there are also further equivalence relations; see [9] for a summary.

We proceed as follows. Section 2 discusses some details of the translation-based method. In Section 3, we describe the current command line interfaces of LPEQ and DLPEQ and give some examples how the tools can be used for equivalence testing in practice. Section 4 concludes our experiences obtained from experiments with the two translator programs and the respective solvers.

2 Theoretical Background

The basic idea in the translation-based approach is to *translate* any two logic programs P and Q under consideration[1] into a single logic program $\mathrm{EQT}(P, Q)$ which has a stable model if and only if P has a stable model that is not a stable model of Q. Such stable models, if found, yield *counter-examples* to $P \equiv Q$. Consequently, the equivalence of P and Q can be established by showing that $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ have no stable models. The details of $\mathrm{EQT}(P, Q)$ can be found in [6] and the current implementation called LPEQ [3] covers the verification of weak equivalence for weight constraint programs [10].

We have also implemented other transformations that allow verifying the strong equivalence of normal logic programs. Turner's characterization of strong equivalence [11] implies that $P \equiv_{\mathrm{s}} Q$ if and only if P and Q are *classically equivalent*, denoted by $P \equiv_{\mathrm{c}} Q$, and for all interpretations M, the models $N \subseteq M$ of the Gelfond-Lifschitz reducts P_M and Q_M [1] coincide. The translator LPEQ includes the implementations of the respective translations $\mathrm{EQT}_{\mathrm{c}}(P, Q)$ and $\mathrm{EQT}_{\mathrm{s}}(P, Q)$ that are used analogously to $\mathrm{EQT}(P, Q)$, but for verifying $P \equiv_{\mathrm{s}} Q$. This has to be done phase-wise, i.e. the classical equivalence must be checked first. The reason is that the translation $\mathrm{EQT}_{\mathrm{s}}(P, Q)$ relies that $P \equiv_{\mathrm{c}} Q$ holds.[2]

Our translation-based approach is generalized to the disjunctive case in [9]. Counter-examples are defined as stable models M of P satisfying *either* (i) $M \not\models Q_M$; *or* (ii) $M \models Q_M$ and $M' \models Q_M$ for some $M' \subset M$. The respective translation $\mathrm{EQT}_{\mathrm{d}}(P, Q)$, which is used analogously to $\mathrm{EQT}(P, Q)$, captures both kinds of counter-examples at once. Yet another approach is based on two translations $\mathrm{EQT}_{\mathrm{d1}}(P, Q)$ and $\mathrm{EQT}_{\mathrm{d2}}(P, Q)$ which capture the two kinds of counter-examples separately. Then $\mathrm{EQT}_{\mathrm{d2}}(P, Q)$ is simplified given that counter-examples of the first kind do not exist. Thus $\mathrm{EQT}_{\mathrm{d1}}(P, Q)$ is to be used first.

[1] It is assumed that the Herbrand bases $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$ coincide.

[2] Note that $\mathrm{EQT}_{\mathrm{s}}(P, Q)$ need not be used if $\mathrm{EQT}_{\mathrm{c}}(P, Q)$ has a stable model.

`lparse p1.lp > p1.sm`	`lparse --dlp p1.lp > p1.sm`
`lparse p2.lp > p2.sm`	`lparse --dlp p2.lp > p2.sm`
`lpeq p1.sm p2.sm | smodels 1`	`dlpeq p1.sm p2.sm | gnt 1`
`lpeq p2.sm p1.sm | smodels 1`	`dlpeq p2.sm p1.sm | gnt 1`

Fig. 1. Examples on running LPEQ and DLPEQ

3 Examples

The translators LPEQ [3] and DLPEQ [8] have been implemented in the C programming language under the Linux operating system. Both translators take two logic programs and command line options as their input and produce a translation for equivalence testing as their output. The input files are assumed to be in an internal format, as produced by the front-end LPARSE of the SMODELS system. Rules with variables can also be used, but LPARSE performs an instantiation for the rules. As an example, the weak equivalence of two *weight constraint programs*, once stored in the input files `p1.lp` and `p2.lp`, can be verified using LPEQ and SMODELS by giving the commands shown in Fig. 1 on the left-hand side. Classical and strong equivalence of *normal programs* can be verified using SMODELS in a similar fashion, but additional command line options `-c` and `-s`, respectively, have to be supplied for LPEQ. Furthermore, LPARSE (preferably version 1.0.13 or later) has to invoked using the command line option `-d all`. Otherwise LPARSE might apply program reductions that are sound for weak equivalence, but incorrect for classical/strong equivalence.

The other translator is used very similarly and it has been designed for disjunctive programs and the solver GNT [4]. In addition to disjunctive rules, DLPEQ supports *compute statements* which are extra constraints on stable models. In order to check the weak equivalence of two *disjunctive programs* `p1.lp` and `p2.lp`, the translator DLPEQ can be run using the commands listed in Fig. 1 on the right. Note that LPARSE has to be invoked with the command line option `--dlp` for disjunctive programs. By default, DLPEQ produces the one-phased translation $\mathrm{EQT_d}(P, Q)$ as its output. Translations for the two-phased approach are produced by the command line options `-p [1|2]`.

A further command line option `-v` triggers a textual output for both translators, which we hope to be useful with solvers that are not compatible with the internal file format. In particular, to enhance interoperability with the DLV system, the output of DLPEQ can be adjusted by the command line option `--dlv`.

4 Experiments

We have carried out several experiments in order to compare the performance obtained with translations produced by LPEQ and DLPEQ with that of NAIVE. In Figure 2, we give the results for testing the weak equivalence of a normal logic program P, which corresponds to a random 3-sat instance, and a variant P' which is obtained from P by dropping one of its rules at random.

Our experiments suggest that the translation-based approach is superior to the naive one when the programs being tested possess many stable models, and in particular when the programs turn out to be not equivalent. But if the number of stable models is low or the programs under consideration do not possess stable models at all, then the naive cross-checking approach is likely to be faster. This is mainly because the translation $\mathrm{EQT}(P, Q)$ employs P as a subprogram to generate stable models of P. Furthermore, our preliminary experiments suggest that for normal programs, verifying strong equivalence is computationally more demanding than verifying weak equivalence. We believe that this is due to a greater search space for counter-examples, as the number of SE-models tends to be much higher than that of stable models.

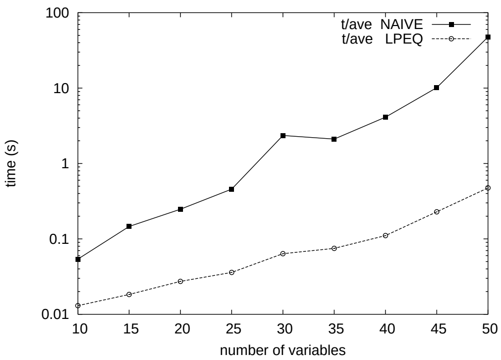

Fig. 2: Testing the equivalence of random 3-sat instances with clauses-to-variables ratio 4.

References

1. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. MIT Press.
2. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
3. T. Janhunen. LPEQ 1.13 — a tool for testing the equivalence of logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2002. Computer Program.
4. T. Janhunen. GnT (version 2) — a tool for computing disjunctive stable models. `http://www.tcs.hut.fi/Software/gnt/`, 2003. Computer Program.
5. T. Janhunen, I. Niemelä, D. Seipel, P. Simons, and J.-H. You. Unfolding partiality and disjunctions in stable model semantics. CoRR: cs.AI/0303009, March 2003.
6. T. Janhunen and E. Oikarinen. Testing the equivalence of logic programs under stable model semantics. In S. Flesca et al., editors, *Logics in Artificial Intelligence, Proceedings of the 8th European Conference*, pages 493–504, Cosenza, Italy, September 2002. Springer-Verlag. LNAI 2424.
7. V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.
8. E. Oikarinen. DLPEQ 1.9 — a tool for testing the equivalence of disjunctive logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2003. Computer Program.
9. E. Oikarinen and T. Janhunen. Verifying the equivalence of logic programs in the disjunctive case. In *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, 2004. This volume.
10. P. Simons, I. Niemelä, and T. Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.
11. H. Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4–5):609–622, 2003.