SELP - A System for Studying Strong Equivalence between Logic Programs *

Yin Chen[1,2], Fangzhen Lin[3] and Lei Li[2]

[1] Department of Computer Science, South China Normal University, China
[2] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
[3] Software Institute, Sun Yat-sen University, China

Abstract. This paper describes a system called **SELP** for studying strong equivalence in answer set logic programming. The basic function of the system is to check if two given ground disjunctive logic programs are equivalent, and if not, return a counter-example. This allows us to investigate some interesting properties of strong equivalence, such as a complete characterization for a rule to be strongly equivalent to another one, and checking whether a given set of rules is strongly equivalent to another, perhaps simpler set of rules.

1 Introduction

The notion of strongly equivalent logic programs was proposed by [Lifschitz *et al.*, 2001]. It has been found useful for tasks such as program simplification. In this paper, we describe a system called **SELP** that can help us answer questions regarding this notion, from simple ones such as "are P and Q strongly equivalent" to more involved ones such as "is there another, preferably simpler logic program that is strongly equivalent to a given one".

The core of the system is checking whether two disjunctive logic programs are strongly equivalent. This is based on [Lin, 2002], which provides a simple mapping from logic programs to propositional theories that reduces strong equivalence to entailment in classical propositional logic. Thus, the problem of strong equivalence checking can be translated into a satisfiability problem in propositional logic, and solved using SAT solvers like *zChaff*. In addition, when two programs are not strongly equivalent, we may want to find some witnesses (counter-example). It is often hard to find a witness manually, but **SELP** can do this automatically: when two programs P_1 and P_2 are found not to be strongly equivalent, it will return a program P, such that $P_1 \cup P$ and $P_2 \cup P$ have different answer sets.

The most interesting part of **SELP** is that it allows us to study some properties of the notion of strong equivalence. In [Lin and Chen, 2005], we described some results on classes of strongly equivalent logic programs discovered using the system. In this paper, we shall show how the system can help us answer

* This work was supported in part by HK RGC CERG HKUST6170/04E.

questions of the following form: Given a set P of rules, is there another set of rules of certain property that is strongly equivalent to P?

In [Janhunen and Oikarinen, 2004], a system call LPEQ was developed for equivalence checking between two programs. It can check if two normal programs are strongly equivalent, and its implementation was based on answer set solver *smodel*. Our approach can deal with normal program as well as disjunctive program. Furthermore, our system can construct a counter-example when two programs are found not strongly equivalent.

We introduce some preliminaries here. Let L be a propositional language, i.e. a set of atoms. In this paper we shall consider logic programs with rules of the following form:

$$h_1; \cdots; h_k \leftarrow p_1, \cdots, p_m, not\, p_{m+1}, \cdots, not\, p_n \tag{1}$$

where h_i's and p_i's are atoms in L. So a logic program here can have default negation (*not*), constraints (when $k = 0$), and disjunctions in the head of its rules. In the following, if r is a rule of the above form, we write Hd_r to denote the set $\{h_1, ..., h_k\}$, Ps_r the set $\{p_1, ..., p_m\}$, and Ng_r the set $\{p_{m+1}, ..., p_n\}$. Thus a rule r can also be written as $Hd_r \leftarrow Ps_r, not\, Ng_r$.

The semantics of program is given by answer sets as defined in [Gelfond and Lifschitz, 1991]. Two logic programs P_1 and P_2 in L are said to be *equivalent* if they have the same answer sets, and *strongly equivalent* [Lifschitz *et al.*, 2001] (in the language L) if for any logic program P in L, $P \cup P_1$ and $P \cup P_2$ are equivalent.

The remainder of this paper is organized as follows. In section 2, we describe the core of the system, i.e. how to check if two programs are strongly equivalent. In section 3, we show how to find all programs that are strongly equivalent to a given one. Finally, we conclude the paper in section 4.

2 Checking strong equivalence between two logic programs

Lifschitz, Pearce, and Valverde [2001] showed that checking for strong equivalence between two logic programs can be done in the logic of here-and-there, a three-valued non-classical logic somewhere between classical logic and intuitionistic logic. Lin [2002] provided a mapping from logic programs to propositional theories and showed that two logic programs are strongly equivalent iff their corresponding theories in propositional logic are equivalent. This result will be the basis that we are using in this paper for checking if two logic programs are strongly equivalent, and we repeat it here.

Let P_1 and P_2 be two finite logic programs, and L the set of atoms in them.

Theorem 1. *[Lin, 2002] P_1 and P_2 are strongly equivalent iff in the propositional logic, the following two entailments hold:*

$$\{p \supset p' | p \in L\} \cup \Delta(P_1) \models \Delta(P_2), \tag{2}$$

$$\{p \supset p' | p \in L\} \cup \Delta(P_2) \models \Delta(P_1). \tag{3}$$

where for each $p \in L$, p' is a new atom, and for each program P, $\Delta(P) = \{\Delta(r) \mid r \in P\}$, where for each rule r of the form (1), $\Delta(r)$ is the conjunction of the following two sentences:

$$p_1 \wedge \cdots \wedge p_m \wedge \neg p'_{m+1} \wedge \cdots \wedge \neg p'_n \supset h_1 \vee \cdots \vee h_k, \tag{4}$$
$$p'_1 \wedge \cdots \wedge p'_m \wedge \neg p'_{m+1} \wedge \cdots \wedge \neg p'_n \supset h'_1 \vee \cdots \vee h'_k. \tag{5}$$

Notice that if $m = n = 0$, then the left sides of the implications in (4) and (5) are considered to be $true$, and if $k = 0$, then the right sides of the implications in (4) and (5) are considered to be $false$.

Theorem 1 makes it possible to check the strong equivalence between two logic programs using a SAT solver. For instance, to verify (2), it is sufficient to check that both of the following two formulas are satisfied for all $r \in P_2$:

$$\{p \supset p' \mid p \in L\} \cup \Delta(P_1) \cup \{p \mid p \in Ps_r\} \cup \{\neg p' \mid p \in Ng_r\} \cup \{\neg p \mid p \in Hd_r\},$$
$$\{p \supset p' \mid p \in L\} \cup \Delta(P_1) \cup \{p' \mid p \in Ps_r\} \cup \{\neg p' \mid p \in Ng_r \cup Hd_r\}.$$

We implement this idea by using the SAT solver *zChaff*.

If P_1 and P_2 are not strongly equivalent, then *zChaff* will return an assignment that is a counter-example to either (2) or (3), and from this assignment, we can construct a program P such that $P \cup P_1$ and $P \cup P_2$ are not equivalent, i.e. P is a witness of the fact that P_1 and P_2 are not strongly equivalent. The next theorem shows how to do this.

Theorem 2. *Let P_1 and P_2 be two programs, M a model of $\{p \supset p' | p \in L\} \cup \Delta(P_1)$, and not $\Delta(P_2)$. Let M_L and $M_{L'}$ be the two sets of atoms defined as follows:*

$$M_L = \{p \mid p \in L \text{ and } M \models p\}, \tag{6}$$
$$M_{L'} = \{p \mid p \in L \text{ and } M \models p'\}. \tag{7}$$

Then we have

(1) If $M_{L'}$ is not closed under P_2, then $P_1 \cup P$ and $P_2 \cup P$ is not equivalent, where $P = \{p \leftarrow \mid p \in M_{L'}\}$.
(2) If $M_{L'}$ is closed under P_2, then $P_1 \cup P$ and $P_2 \cup P$ is not equivalent, where $P = \{p \leftarrow \mid p \in M_L\} \cup \{p \leftarrow q \mid p, q \in M_{L'} \setminus M_L, p \neq q\}$.

While the basic function of our system **SELP** is to check whether two given logic programs are strongly equivalent, and if not provides a witness, we do not envision its use this way. Rather, we consider it a tool to systematically study the notion of strong equivalence. We have used this system to discover theorems about strongly equivalent logic programs in [Lin and Chen, 2005]. Here we describe how our system can be used to find whether there is a simpler set of rules that is strongly equivalent to a given set of rules.

3 Finding strongly equivalent logic programs

One application of **SELP** is about finding out whether there is another, preferably simpler logic program that is strongly equivalent to a given one. For instance, we have seen that the self-loops (loops of length one) like $p \leftarrow p, q$ are strongly equivalent to $\emptyset$. A natural follow-up question is then: what about loops of length two, like $\{(a \leftarrow b), (b \leftarrow a)\}$?

Given a program P in the language L, an obvious way to look for another program in L that is strongly equivalent to P would be to generate all possible programs in L, and call **SELP** on them one by one. This is clearly infeasible even for a program with only three or four atoms. Fortunately, there is a much better way of doing it. Instead of considering all possible sets of rules, we can first find all possible rules that are redundant in the presence of P, i.e. all rules r such that $P \cup \{r\}$ is strongly equivalent to P, and consider sets of these rules only, as the following theorem says.

Theorem 3. *Let P be a logic program in L, and S the set of rules defined as follows:*

$$S = \{r \mid r \text{ is in } L, \text{ and } P \cup \{r\} \text{ and } P \text{ are strongly equivalent } \}.$$

For any program Q in L, if P and Q are strongly equivalent, then $Q \subseteq S$.

Notice that the set S in the theorem includes "trivial" rules like $p \leftarrow p$. As we mentioned in [Lin and Chen, 2005], we need to consider only rules where each atom occurs at most once, i.e. non-redundant rules.

Corollary 1. *Let P be a logic program in L, and S_P the set of rules defined as follows:*

$S_P = \{r | r$ *is a non-redundant rule in L, and $P \cup \{r\}$ and P are strongly equivalent* $\}$.

For any program Q in L, if P and Q are strongly equivalent, then $Q' \subseteq S_P$, where Q' is obtained from Q by deletion rules that are strongly equivalent to $\emptyset$, and replace each remaining rule r by $Hd_r \setminus Ng_r \leftarrow Ps_r, not\, Ng_r$.

Using this corollary, our system **SELP** finds all programs that are strongly equivalent to a given logic program in two steps:

- generate all possible non-redundant rule r, and check if P is strongly equivalent to $P \cup \{r\}$, thus computing the set S_P,
- for each subset of S_P, check if it is strongly equivalent to P.

For our example loop with length two, $P_1 = \{(a \leftarrow b), (b \leftarrow a)\}$, S_{P_1} consists of the following rules:

$$a \leftarrow b \qquad b \leftarrow a \qquad \leftarrow a, not\, b \qquad \leftarrow b, not\, a$$

As it turned out, there is no subset of S_{P_1} that is strongly equivalent to P_1 yet does not contain P_1, i.e. P_1 cannot be simplified using strong equivalence.

4 Concluding remarks and future work

We develop a tool **SELP** for studying strong equivalence between logic programs. It can check if two programs are strongly equivalent, as well as, it is helpful for us to find some general theorems on strong equivalence. We will try to find some more theorems by **SELP**. It is also an interesting work to find some general theorems on uniform equivalence, and to find the difference between two kinds of equivalence.

References

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

[Janhunen and Oikarinen, 2004] Tomi Janhunen and Emilia Oikarinen. Lpeq and dlpeq - translators for automated equivalence testing of logic programs. In Vladimir Lifschitz and Ilkka Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 336–340. Springer, 2004.

[Lifschitz *et al.*, 2001] V. Lifschitz, D. Pearce, and A. Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[Lin and Chen, 2005] Fangzhen Lin and Yin Chen. Discovering classes of strongly equivalent logic programs. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI–05)*, 2005. To appear.

[Lin, 2002] Fangzhen Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In *Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002)*, pages 170–176, 2002.