An Implementation for Recognizing Rule Replacements in Non-Ground Answer-Set Programs*

Thomas Eiter, Patrick Traxler, and Stefan Woltran

Institut für Informationssysteme 184/3, Technische Universität Wien,
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,traxler,stefan}@kr.tuwien.ac.at

1 Introduction

Answer-set programming (ASP) has emerged as an important paradigm for declarative problem solving, and provides a host for many different application domains on the basis of nonmonotonic logic programs. The increasing popularity in ASP has raised also the interest in semantic comparisons of programs in ASP [3, 4], which are nowadays recognized as a theoretical basis for program optimization, where equivalence-preserving modifications are of primary interest; in particular, rewriting rules which allow to perform a local change in a program are important. Many such rules have been considered in the propositional setting (cf., e.g., [1, 6]) but just recently have been extended to the practicably important case of non-ground programs [2].

For illustration, consider rules from an encoding of the 3-colorability problem:

$$b(X) \vee b(a) \leftarrow edge(X, a), node(X), \text{not } r(X), \text{not } g(a), \text{not } g(X) \quad (1)$$

$$r(Y) \vee b(Y) \vee g(Y) \leftarrow node(Y). \quad (2)$$

Results from [2] show that (i) the first rule is redundant and can be deleted in any program which contains the second rule; (ii) the entire program fragment can be rewritten into a program without disjunctions, which is equivalent for any graph specification.

In this paper, we present theoretical foundations and a practical realization for recognizing these two particular replacements, which are *rule subsumption* and *local shifting*. We describe a tool which *scans* an input program and indicates which rules can be deleted (via subsumption) and which rules apply to local shifting. As a back-end inference engine for these recognition tasks, we make use of ASP-solvers, themselves. We report first experimental evaluations, showing that our approach is feasible.

2 Replacements in Answer-Set Programming

Our objects of interest are disjunctive logic programs formulated in a language over a set $\mathcal{A}$ of *predicate symbols*, a set $\mathcal{V}$ of *variables*, and a set $\mathcal{C}$ of *constants* (also called the *domain*). Atoms, rules, and programs are defined as usual and we use, for a rule r, $H(r)$ to denote the set of atoms in the *head* of r, $B(r)$ to denote the set of literals in the *body* of r, and $B^+(r)$ (resp., $B^-(r)$) to refer to the set of positive (resp., negative)

* This work was partially supported by the Austrian Science Fund (FWF) under project P18019.

atoms in $B(r)$. Let e be an atom, rule, or a program. The set of variables occurring in e is denoted by $\mathcal{V}_e$, and e is called *ground* iff $\mathcal{V}_e = \emptyset$. Similarly, we use $\mathcal{C}_e$ to refer to the set of constants occurring in e. Given a rule r and a set of constants $C \subseteq \mathcal{C}$, we define $grd(r, C)$ as the set of all rules $r\vartheta$ obtained from r by all possible substitutions $\vartheta : \mathcal{V}_r \to C$. The semantics of logic programs is given in terms of answer sets as usual. $\mathcal{AS}(P)$ denotes the set of all answer sets of a program P. Programs P_1, P_2 are called *strongly* (resp., *uniformly*) *equivalent* iff, for each set S of rules (resp., facts), $\mathcal{AS}(P_1 \cup S) = \mathcal{AS}(P_2 \cup S)$. For further details we refer to [3].

We consider here two forms of replacements, *rule subsumption* and *local shifting*, cf. [2]. The former is based on the following observation, generalizing a result in [5].

Proposition 1. *Let P be a program, and r, s be different rules in P, such that there exists a substitution $\vartheta : \mathcal{V}_s \to \mathcal{V}_r \cup \mathcal{C}_r$ satisfying $H(s\vartheta) \subseteq H(r) \cup B^-(r)$ and $B(s\vartheta) \subseteq B(r)$. Then, P is strongly equivalent to $P \setminus \{r\}$.*

The aim of local shifting is motivated by the elimination of disjunctions. We need the following notion: A ground rule r is *head-cycle free* (HCF) in a ground program P iff no distinct atoms $a, b \in H(r)$ are on a common cycle in the positive dependency graph of P. For an arbitrary program P (not necessarily ground), $r \in P$ is HCF in P iff, for each finite $C \subseteq \mathcal{C}$ and each $r' \in grd(r, C)$, r' is HCF in $grd(P, C)$.

Proposition 2. *Let P be a program and $r \in P$, such that for each $\vartheta : \mathcal{V}_r \to \mathcal{C}$, $|H(r\vartheta)| = |H(r)|$, and r is HCF in P. Then, P is uniformly equivalent to $P \setminus \{r\} \cup r^{\to}$, where*[1] *$r^{\to} = \{h \leftarrow B(r), \mathrm{not}(H(r) \setminus \{h\}) \mid h \in H(r)\}$.*

In our example from the introduction, it can be shown that (1) is subsumed by (2) by setting $\vartheta(X) = Y$. As well, both rules are HCF within the fragment, but (1) does not apply for local shifting since under $\vartheta(X) = a$, its head reduces to a single element. Next, we recapitulate the complexity of detecting rules for subsumption or shifting.

Proposition 3. *Given a program P and $r \in P$. Deciding whether (i) r is subsumed by some other rule $s \in P$ is* NP*-complete; (ii) r is HCF in P is* PSPACE*-complete.*

3 The Implemented System

For our implementation we use reductions to ASP itself to decide the problems under consideration (for details, see [7]). In particular, we provide a linear reduction for subsumption into the conjunctive query problem which is NP-complete, matching the intrinsic complexity of the encoded task. In the case of local shifting, we map (in polynomial time) this problem to that of querying a definite Horn program, which is EXPTIME-complete and thus just mildly harder than the encoded problem.

Encodings. We reduce the test whether a rule r is subsumed by a rule s as a (Boolean) conjunctive query problem (F, q), i.e., deciding, given a rule $q = b \leftarrow a_1, \ldots, a_m$ together with a set F of ground facts, whether the unique answer set of $F \cup \{q\}$ contains

[1] For a set $S = \{s_1, \ldots, s_n\}$ of atoms, not S abbreviates not $s_1, \ldots,$ not s_n.

b. Given two rules r, s, we construct a set F_r of facts and a query q_s as follows, where for any rule r, r' (resp., r'') denotes the result of replacing each predicate symbol p in r by a new symbol p' (resp., p''), and the substitution $\gamma : \mathcal{V} \rightarrow \mathcal{C}$ maps each variable V to a corresponding constant c_V, which does not occur in r, s:

$$\begin{aligned} F_r &= H(r\gamma) \cup B^-(r\gamma) \cup B^+(r'\gamma) \cup B^-(r''\gamma); \\ q_s &= b \leftarrow H(s), B^+(s'), B^-(s''). \end{aligned}$$

Theorem 1. *Rule r is subsumed by rule s iff the query problem (F_r, q_s) holds.*

Concerning local shifting, let, for a program P, $C^* \supseteq \mathcal{C}_P$ be a domain of size $|C^*| = 4 \cdot |\mathcal{C}_P|$, k be the maximal predicate-arity in P, let h, d, p and q be new predicate symbols with arities $\alpha(d) = 1$, $\alpha(h) = 2$, and $\alpha(p) = \alpha(q) = 2k + 2$, and let $\_$ be a new constant symbol. Define for two atoms $a = a'(t_1, \ldots, t_m)$ and $b = b'(s_1, \ldots, s_l)$ with $m, l \leq k$, and $\pi \in \{p, q\}$, the rule

$$\pi[a, b] := \pi(a', t_1, \ldots, t_m, \_, \ldots, \_, b', s_1, \ldots, s_l, \_, \ldots, \_) \leftarrow D, \tag{3}$$

such that b' appears as the $(k + 2)$nd argument in π, the $\_$'s fill up π properly, and D denotes a sequence of $d(X)'s$, for all variables X occurring in a or b. For a program P and a set of rules $Q \subseteq P$, we define the definite Horn program

$$\begin{aligned} P^*_Q = \{&d(c) \mid c \in C^*\} \cup \\ &\{q[a, b] \mid a, b \in H(r), a \neq b, r \in Q\} \cup \\ &\{p[a, b] \mid a \in H(s), b \in B^+(s), s \in P, H(s) \neq \emptyset, B^+(s) \neq \emptyset\} \cup \\ &\{p(\boldsymbol{x}, \boldsymbol{z}) \leftarrow p(\boldsymbol{x}, \boldsymbol{y}), p(\boldsymbol{y}, \boldsymbol{z}); \\ &\;\; h(\boldsymbol{x}, \boldsymbol{y}) \leftarrow p(\boldsymbol{x}, \boldsymbol{y}), p(\boldsymbol{y}, \boldsymbol{x}), q(\boldsymbol{x}, \boldsymbol{y}); \quad h(\boldsymbol{x}, \boldsymbol{x}) \leftarrow q(\boldsymbol{x}, \boldsymbol{x})\}; \end{aligned}$$

where $\boldsymbol{x}$ (resp., $\boldsymbol{y}$, $\boldsymbol{z}$) denotes a sequence of $k + 1$ distinct variables X_i (resp., Y_i, Z_i).

Theorem 2. *For any program P and $Q \subseteq P$, each $r \in Q$ is HCF in P iff no atom $h(\cdot, \cdot)$ is contained in the (unique) answer set of P^*_Q.*

Hence, we are able to test whether a single rule r is HCF in P (via querying $P^*_{\{r\}}$) or whether P entirely is HCF (via querying P^*_P). Moreover, inspecting atoms $h(\cdot, \cdot)$ in the answer set of P^*_Q, indicates which pair of atoms prevent rules in Q from being shifted.

System Description. Both reductions and invocations of the ASP-solver `DLV`[2] are realized via perl scripts. A simple shell script runs both tests. All scripts and further information are available at `http://www.kr.tuwien.ac.at/research/eq/simpl/`. The input program is required to be in `DLV`-format. Consider the file `program.dl` contains our example program from above. Starting `simplify program.dl` then yields:

```
Scanning for Rule Subsumption...
b(X) v b(a) :- edge(X,Y),node(X),not r(X),not g(a),not g(X).
[subsumed by r(Y) v b(Y) v g(Y) :- node(Y).]

Scanning for Local Shifting...
r(Y) v b(Y) v g(Y) :- node(Y).
```

[2] Available under `http://www.dlvsystem.com`.

Experiments. For first results on our approach, we set up a test series available at

`http://www.kr.tuwien.ac.at/research/eq/simpl/`

The test for subsumption always involves a pair of rules, while the test for local shifting has to take an entire program into account. Thus, we encode for the former different NP-hards problems as pairs of rules such that subsumption holds iff the encoded problem holds. For the latter we used various application programs (some from the web) and tested whether disjunction can be eliminated in a uniform-equivalence preserving way.

Concerning subsumption, we encoded (i) graph 3-colorability and (ii) propositional satisfiability. For (i), consider a graph $G = (V, E)$ and let B_E denote the sequence of atoms $e(X_i, X_j)$, where $(v_i, v_j) \in E$. Then the rule $\leftarrow e(r, b)$, $e(b, r)$, $e(r, g)$, $e(g, r)$, $e(b, g)$, $e(g, b)$ is subsumed by the rule $\leftarrow B_E$ iff G is 3-colorable. Our system scales well showing reasonable response times for problems generated from graphs containing up to 40 nodes (depending on the number of nodes, but within a few seconds for 30 nodes). For (ii), consider a CNF ϕ over variables $X_1, \ldots, X_n$ and represent each clause c by a triple $p(L_1, L_2, L_3)$ where $L_i = X$ (resp., $L_i = \bar{X}$) if X (resp., $\neg X$) is the i-th literal in c. Let p_ϕ be the sequence representing ϕ in this way plus pairs $v(X, \bar{X})$ for all variables X occurring in ϕ. Then, $\leftarrow p(1, 1, 1)$, $p(1, 1, 0)$, $p(1, 0, 1)$, $p(1, 0, 0)$, $p(0, 1, 1)$, $p(0, 1, 0)$, $p(0, 0, 1)$, $v(1, 0)$, $v(0, 1)$ is subsumed by the rule $\leftarrow p_\phi$ iff ϕ is satisfiable. Also in this case, our implementation provides good response times (around a few seconds) for a suite of uniform random 3-sat formulas taken from SATLIB.

Concerning the test for local shifting, we set up a suite of disjunctive programs collected from different sources, including encodings for problems as *Hamiltonian cycle*, *strategic companies*, or *diagnosis*. For all programs, our tool recognized all rules applicable to local shifting rather fast (always within a second).

The presented work has to be seen as a starting point for a more general tool considered as support for programmers in terms of offline simplification of (possibly incomplete) programs. To the best of our knowledge, our implementation is the first realization of such simplification methods working directly on non-ground programs.

References

1. S. Brass and J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, 38(3):167–213, 1999.
2. T. Eiter, M. Fink, H. Tompits, P. Traxler, and S. Woltran. Replacements in Non-Ground Answer-Set Programming. Accepted for publication in *Proc. KR'06*, 2006.
3. T. Eiter, M. Fink, H. Tompits, and S. Woltran. Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case. In *Proc. AAAI'05*, pg. 695–700. AAAI Press, 2005.
4. V. Lifschitz, D. Pearce, and A. Valverde. Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.
5. F. Lin and Y. Chen. Discovering Classes of Strongly Equivalent Logic Programs. In *Proc. IJCAI'05*, pages 516–521, 2005.
6. M. Osorio, J. A. Navarro, and J. Arrazola. Equivalence in Answer Set Programming. In *Proc. LOPSTR'01, Selected Papers*, vol. 2372 of *LNCS*, pg. 57–75. Springer, 2001.
7. P. Traxler. Techniques for Simplifying Disjunctive Datalog Programs with Negation. Master's thesis, Technische Universität Wien, Institut für Informationssysteme, 2006.