New DLV Features for Data Integration*

Francesco Calimeri[1], Manuela Citrigno[1], Chiara Cumbo[1], Wolfgang Faber[2], Nicola Leone[1], Simona Perri[1], and Gerald Pfeifer[2]

[1] Department of Mathematics, University of Calabria - 87030 Rende (CS), Italy
{calimeri,citrigno,cumbo,leone,perri}@mat.unical.it
[2] Institut für Informationssysteme, TU Wien - A-1040 Wien, Austria
faber@kr.tuwien.ac.at gerald@pfeifer.com

Abstract. The DLV system is currently employed in projects on data integration – a challenging application area for Answer Set Programming. The present system description illustrates some new optimization techniques, which significantly enhance the general performance of DLV, and especially in the context of data integration.

1 Introduction

The integration of (possibly inconsistent) data from different sources is one of the most promising applications of Answer Set Programming [5, 1, 3, 14]. Recent studies [2, 10, 4, 7, 5] showed that query answering in data integration is often complete for the complexity class co-NP, or even for Π_2^P. This high computational complexity is mainly due to the necessity of handling inconsistencies (e.g., violations of key constraints) in the "integrated" database DB, which are due to conflicting data coming from different sources.

The (consistent) answer to a query $q(X)$ over a (possibly inconsistent) Data Integration System DB can be obtained by generating a logic program LP such that the stable models of LP yield the repairs[1] of DB. Answering the query $q(X)$ over the data integration system then amounts to cautious reasoning over the logic program LP [5, 14, 3], which can be performed by an ASP engine.

At the time being, the DLV system [12] is the most "database oriented" ASP engine, and it seems to be the best-suited ASP system for data-integration tasks. The suitability of DLV for data integration is backed by its use in relevant works in data integration, like [5, 14, 3] and by an IST project funded by the European Commission focusing on the exploitation of DLV for information integration (INFOMIX project, IST-2001-33570). Furthermore, in the experiments reported in [3], which also considered QBF, SAT, and CLP systems, DLV outperforms the compared systems on database repairing in the majority of cases.

However, as observed also in [5], DLV would be much more effective in database applications if it could take advantage of suitable optimization methods that speed up the evaluation of queries expressed as logic programs.

* This works was supported by the European Commission, under projects INFOMIX (IST-2001-33570), and WASP (IST-2001-37004).

[1] Intuitively, a repair is a maximal *consistent* subset of the database.

The present system description illustrates some recent enhancements in this direction [8, 11], which have not been demonstrated to a broader audience yet. In particular, we focus on the following optimization techniques.
(*i*) Query oriented techniques for binding propagation (Magic Sets) [8]. Intuitively, the adoption of binding propagation techniques in Disjunctive Logic Programming (DLP) allows us to exploit constants appearing in the query and the program, reducing the size of the instantiation by avoiding "a priori" the generation of many ground instances of the rules which cannot contribute to the derivation of the query goal.
(*ii*) Backjumping techniques for the program instantiation [11]. The application of a new, structure-based backjumping method to the instantiation process of `DLV` significantly reduces the instantiation time and, very importantly, limits the size of the instantiation by generating, for each program rule, only a relevant subset of all its possible ground instances.

The design and implementation of these optimizations is an important step towards the concrete use of ASP systems in real-life applications.

2 Improving DLP instantiators for larger data manipulations

The kernel modules of most ASP systems operate on a ground instantiation of the input program [9]. Any input program P is first submitted to an instantiation process, which may be computationally very expensive. Thus, having a good instantiator is a key feature. It should produce a ground program P' having the same answer sets as P such that: (i) P' can be computed efficiently from P, and (ii) P' is as small as possible.

The main reason for large ground programs even for small input programs is that each atom of a rule in a program $\mathcal{P}$ may be instantiated to many atoms in the Herbrand base, which leads to a combinatorial explosion. However, most of these atoms may not be derivable whatsoever, and hence such instantiations do not render applicable rules. A good instantiator thus generates ground instances of rules containing only atoms which *might* be derivable from $\mathcal{P}$.

At each step of an instantiation by `DLV`, there is a number of predicates, called *solved*, such that the (total) truth values of all their ground instances have already been determined by the instantiator. For instance, all predicates defined only by facts are solved. Occurrences of these predicates can be safely omitted from the resulting ground program; it suffices to include asserting facts for each true ground instances of solved predicates.

In other words, we are not interested in all "consistent" substitutions for all variables, but rather in their restrictions to the variables that occur in literals over unsolved predicates. To this end, we designed and implemented a new backjumping-based instantiation method. In particular, given a rule r to be instantiated, our algorithm exploits both the semantical and the structural information about r in order to compute only a relevant subset of all its possible ground instances (see [11] for a detailed illustration).

Example 1. Consider the rule r_1 below, where predicates q_3, q_4 and q_5 are solved. The following are all applicable ground instances of r_1.

$$r_1 : a(X,Z) \texttt{ :- } q_1(X,Z,Y), q_2(W,T,S), q_3(V,T,H), q_4(Z,H), q_5(T,S,V).$$
$$a(x_1,z_1) \texttt{ :- } q_1(x_1,z_1,y_1), q_2(w_1,t_1,s_1), q_3(v_1,t_1,h_1), q_4(z_1,h_1), q_5(t_1,s_1,v_1).$$
$$a(x_1,z_1) \texttt{ :- } q_1(x_1,z_1,y_1), q_2(w_1,t_1,s_1), q_3(v_2,t_1,h_1), q_4(z_1,h_1), q_5(t_1,s_1,v_2).$$
$$\vdots$$
$$a(x_1,z_1) \texttt{ :- } q_1(x_1,z_1,y_1), q_2(w_1,t_1,s_1), q_3(v_{100},t_1,h_{100}), q_4(z_1,h_{100}), q_5(t_1,s_1,v_{100}).$$

All these 10000 rules are semantically equivalent to a single instance, which is the only one computed by our algorithm: $a(x_1,z_1) \texttt{ :- } q_1(x_1,z_1,y_1), q_2(w_1,t_1,s_1)$.

3 Magic Sets for DLP

The Magic Sets method [13] is a strategy for simulating the top-down evaluation of a query using a bottom-up evaluation procedure. It modifies the original program by means of certain rules, that act as filters for the relevant information. Roughly, the goal is to use the constants appearing in the query and the program in order to reduce the size of the instantiation by eliminating "a priori" some ground instances of the rules which are irrelevant for the query goal.

In [8] the Magic Set algorithm has been extended to DLPs. While in non-disjunctive programs, bindings are propagated only head-to-body, any rewriting for DLPs has to propagate bindings also head-to-head in order to preserve soundness. For instance, consider the rule $p(X) \texttt{ v } q(Y) \texttt{ :- } a(X,Y), r(X)$ and the query $p(1)$? on a program Π. Even though the query propagates the binding for the predicate p, in order to correctly answer the query, we also need to evaluate the truth value of $q(Y)$, which indirectly receives the binding through the body predicate $a(X,Y)$. If the program contains facts $a(1,2)$, and $r(1)$, then atom $q(2)$ is relevant for the query, since the truth of $q(2)$ would invalidate the derivation of $p(1)$ from the above rule (because of minimality in the semantics). This shows that the bindings also have to be propagated head-to-head.

An extension of Magic Sets to DLP has been implemented in DLV, resulting in an algorithm called DMS [8].

Example 2 (Strategic Companies [6]). A collection C of companies produces some goods in a set G; each company $c_i \in C$ is controlled by a set of other companies $O_i \subseteq C$. $C' \subset C$ is a *strategic set* if it is a minimal set of companies producing all the goods in G, such that if $O_i \subseteq C'$ for some $i = 1, \ldots, m$ then $c_i \in C'$ must hold. This scenario can be modeled by means of a program $\mathcal{P}_{sc}$:

$$r_1 : sc(C_1) \texttt{ v } sc(C_2) \texttt{ :- } produced\_by(P, C_1, C_2).$$
$$r_2 : sc(C) \texttt{ :- } controlled\_by(C, C_1, C_2, C_3),\ sc(C_1),\ sc(C_2),\ sc(C_3).$$

Moreover, given a company $c \in C$, we consider a query $\mathcal{Q}_{sc} = sc(c)$ asking whether c belongs to some strategic set of C. The output of DMS is the following program, which in general is much more efficient for answering $\mathcal{Q}_{sc}$ [8].

$magic\_sc^b(c)$.
$magic\_sc^b(C_2)$:- $magic\_sc^b(C_1),\ produced\_by(P, C_1, C_2)$.
$magic\_sc^b(C_1)$:- $magic\_sc^b(C_2),\ produced\_by(P, C_1, C_2)$.
$magic\_sc^b(C_1)$:- $magic\_sc^b(C),\ controlled\_by(C, C_1, C_2, C_3)$.
$magic\_sc^b(C_2)$:- $magic\_sc^b(C),\ controlled\_by(C, C_1, C_2, C_3)$.
$magic\_sc^b(C_3)$:- $magic\_sc^b(C),\ controlled\_by(C, C_1, C_2, C_3)$.
r'_{1_m} : $sc(C_1)$ v $sc(C_2)$:- $magic\_sc^b(C_1),\ magic\_sc^b(C_2),\ produced\_by(P, C_1, C_2)$.
r''_{1_m} : $sc(C_2)$ v $sc(C_1)$:- $magic\_sc^b(C_2),\ magic\_sc^b(C_1),\ produced\_by(P, C_1, C_2)$.
r_{2_m} : $sc(C)$:- $magic\_sc^b(C),\ controlled\_by(C, C_1, C_2, C_3),\ sc(C_1),\ sc(C_2),\ sc(C_3)$.

4 Conclusions

We have described some of the most recent enhancements of DLV. These have been motivated by applications in data integration, where large amounts of data are to be processed and scalability is a very important issue. Their practical impact has been assessed by many experiments [8, 11], with very positive results.

References

1. INFOMIX project (IST-2001-33570). http://www.mat.unical.it/infomix/.
2. M. Arenas, L. E. Bertossi, and J. Chomicki. Specifying and querying database repairs using logic programs with exceptions. pp. 27–41, 2000.
3. O. Arieli, M. Denecker, B. Van Nuffelen, and M. Bruynooghe. Database repair by signed formulae. In *Proceedings of FoIKS 2004*, LNCS 2942, pp. 14–30, 2004.
4. P. Barceló and L. Bertossi. Repairing databases with annotated predicate logic. NMR 2002.
5. L. Bravo and L. E. Bertossi. Logic Programs for Consistently Querying Data Integration Systems. In *Proceedings of IJCAI'03*, pp. 10–15, 2003.
6. M. Cadoli, T. Eiter, and G. Gottlob. Default Logic as a Query Language. *IEEE TKDE*, 9(3):448–463, 1997.
7. A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proceedings of IJCAI 2003*, pp. 16–21.
8. C. Cumbo, W. Faber, G. Greco, and N. Leone. Enhancing the Magic Set Method for Disjunctive Datalog Programs. In *Proceedings of ICLP 2004*. To appear.
9. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A Deductive System for Nonmonotonic Reasoning. In *Proceedings of LPNMR 1997*, n.1265, LNAI, pp. 363–374.
10. G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. v.2237 of *LNAI*, pp. 348–364, 2001.
11. N. Leone, S. Perri, and F. Scarcello. Backjumping Techniques for Rules Instantiation in the DLV System. In *Proceedings of NMR2004*, pp. 258–266.
12. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL*, 2004. To appear. Available via http://www.arxiv.org/ps/cs.AI/0211004.
13. J. D. Ullman. *Principles of Database and Knowledge Base Systems*, 1989.
14. D. Lembo, M. Lenzerini, and R. Rosati. Source Inconsistency and Incompleteness in Data Integration. KRDB 2002.