How Modern Deductive Database Systems Can Enhance Data Integration

Francesco Calimeri[1,2], Simona Perri[1], Giorgio Terracina[1], and Jessica Zangari[1]

[1] Department of Mathematics and Computer Science, University of Calabria, Rende, Italy
{calimeri,perri,terracina,zangari}@mat.unical.it
[2] DLVSystem Srl, Rende, Italy
calimeri@dlvsystem.com

Abstract. Data integration systems provide a transparent access to heterogeneous, possibly distributed, sources; deductive database and their extensions allow to easily address complex issues arising in data integration. However, the gap between state-of-the-art deductive databases and data integration systems is still to be closed. In this paper we focus on some recent advancements implemented in the $\mathcal{I}$-DLV system, and point out how these can facilitate the development of advanced data integration systems.

Keywords: Deductive DataBase, Data Integration, Instantiation, Answer Set Programming

1 Introduction

The task of an *information integration system* is to combine data residing at different sources, providing the user with a unified view called *global schema*. Users can formulate queries in a transparent and declarative way over the global schema: they do not need to be aware of details about the sources: the information integration system automatically retrieves relevant data from the sources, and suitably combines them to provide answers to user queries [16]. The global schema may also contain integrity constraints (such as key dependencies, inclusion dependencies, etc.).

Recent developments in IT, have made available a huge number of information sources, typically autonomous, heterogeneous and widely distributed. As a consequence, information integration has emerged as a crucial issue in several application domains, e.g., distributed databases, cooperative information systems, data warehousing, ontology-based data access, or on-demand computing. Deductive database systems in general, and Answer Set Programming (ASP) in particular, are powerful tools in this context, as demonstrated, for instance,

SEBD 2018, June 24-27, 2018, Castellaneta Marina, Italy.

by the approaches formalized in [3, 4, 17]. More generally, the adoption of logic-based systems allows to easily address complex problems like Consistent Query Answering (CQA) [22] and querying ontologies under inconsistencies [15]. The database community has spent many efforts in this area, and relevant research results have been obtained to clarify semantics, decidability and complexity of data-integration systems under different assumptions. However, filling the gap between deductive database systems and database integration tools is still an open challenge, and continuous improvements and extensions in ASP systems [9, 14] are certainly important contributions to reach this goal.

In this paper we discuss some of the most recent database oriented innovations in ASP as implemented in the $\mathcal{I}$-DLV system, and we point out how such improvements may enhance advanced data integration systems.

2 The $\mathcal{I}$-DLV System

The $\mathcal{I}$-DLV system [7] is a stand-alone modern ASP instantiator and deductive database engine, that has been also integrated as the grounding module of the renewed version of the popular system DLV [1]. The description of all the features of $\mathcal{I}$-DLV is out of the scope of this paper. In the following, we outline the major features having an important impact on $\mathcal{I}$-DLV as deductive database engine. For a comprehensive list of customizations and options, along with further details, we refer the reader to [7, 6] and to the online documentation [10].

2.1 Overview of evaluation features

$\mathcal{I}$-DLV supports the ASP-Core-2 [5] standard language; its high flexibility and extensible design ease the incorporation of optimization techniques, language updates and customizability. We provide next a brief overview of its instantiation process, focusing on peculiar optimizations whose synergic work, that can be driven at a fine-grained level from the user, is the key of $\mathcal{I}$-DLV efficiency.

Optimizations. The system adopts a bottom-up evaluation strategy based on a semi-naïve approach [27]. One of the most crucial and computationally expensive tasks is the grounding of each rule; it resembles the evaluation of the *relational joins* among positive body literals, and $\mathcal{I}$-DLV adopts a bunch of techniques to optimize it, many of which inspired by the database field and properly enhanced and readapted to $\mathcal{I}$-DLV purposes. Here we mention *body-reordering* criteria, *indexing* strategies and *decomposition rewritings* [8], along with additional fine-tuning optimizations acting to different extents on the evaluation process, with the general common aim of reducing the search space and improving overall performances [7].

- *Body-reordering* techniques aim at finding an optimal execution ordering for the join operations, by varying the order of literals in the rule bodies. Different orderings have been defined for $\mathcal{I}$-DLV; the one adopted by default has been specifically designed by considering the effects of each literal on the binding of variables [7].

– *Indexing* techniques, instead, are intended to optimize the retrieval of matching instances from the predicate extensions. $\mathcal{I}$-DLV defines a flexible indexing schema: any predicate argument can be indexed, allowing both single- and multiple-argument indices, and for each predicate different indexing data structures can be built "on-demand", only if needed, while instantiating a rule containing that predicate in its body.
– $\mathcal{I}$-DLV exploits also a heuristic-guided *decomposition rewriting* technique relying on hyper-tree decompositions that replaces long rules with sets of smaller ones, with the aim of transforming the input program into an equivalent one possibly evaluated more efficiently.
– Eventually, we cite a series of techniques falling into the category of *join optimizations*, such as "pushing down selections" and other join rewritings; they have diverse aims, such as decreasing the number of matches considered during rule instantiation, early recovering inconsistencies in the input program, or syntactically rewriting the input program with the twofold intent of easing the instantiation and improving performance.

Query answering in $\mathcal{I}$-DLV is empowered with the *magic-sets* technique [2]: when the input program features a query, it simulates a top-down computation by rewriting the input program for identifying the relevant subset of the instantiation which is sufficient for answering the query. Restrictions on the instantiation is obtained by means of additional "magic" predicates, whose extensions represent relevant atoms with respect to the query.

Customizability. $\mathcal{I}$-DLV provides a fine-grained control over the whole computational process, allowing for enabling/disabling each one of the many optimization techniques both via command-line options and inline annotations. More in detail, $\mathcal{I}$-DLV programs can be enriched by global and local annotations (i.e., on a per-rule basis), for customizing some machineries such as *body ordering* and *indexing*. For instance, the indexing schema of a specific atom in a rule can be constrained to satisfy some specific conditions, annotating the rule as follows: `%@rule_atom_indexed(@atom=a(X,Y,Z), @arguments={0,2}).` when instantiating the annotated rule, the atom $a(X, Y, Z)$ will be indexed, if possible, with a double-index on the first and third arguments.

Since its release, $\mathcal{I}$-DLV proved its reliability and efficiency as both ASP grounder and deductive database engine. Recently, in the latest ASP Competition [14] $\mathcal{I}$-DLV ranked both as first and second combined with an automatic solver selector [12] that inductively chooses the best solver depending on some inherent features of the instantiation produced, and with the state-of-the-art solver *clasp* [13], respectively. Moreover, $\mathcal{I}$-DLV performance results are promising also as deductive database system [7]. The system has been tested on the query-based set of problems from OpenRuleBench [20], an open set of resources featuring a suite of benchmarks for analyzing performance and scalability of different rule engines, and compared with the former DLV version and XSB [24], which was among the clear winners of the official OpenRuleBench runs [20] and is currently one of the most widespread logic programming and deductive

database systems. Results show that not only $\mathcal{I}$-DLV behaves better than DLV, but it is definitely competitive against XSB. For a detailed description on such experiments we refer the reader to [7].

2.2 Interoperability Features

In this section we briefly illustrate mechanisms and tools of $\mathcal{I}$-DLV for *(i)* interoperability with external systems, *(ii)* accommodation of external sources of computation, and *(iii)* value invention/modification within logic programs. In particular, $\mathcal{I}$-DLV supports direct connection with relational databases and SPARQL enabled ontologies via explicit import/export *directives*, and access to external data via calls to Python scripts with *external atoms*.

RDBMS Data Access. $\mathcal{I}$-DLV can import relations from an RDBMS by means of an `#import_sql` directive. For instance, `#import_sql(DB, "user", "pass", "SELECT * FROM t", p)` can access database `DB` and import all tuples from table `t` into facts with predicate name `p`. Similarly, `#export_sql` directives are used to populate specific tables with the extension of a predicate.

Ontology-Based Data Access. Data can also be imported from *local* RDF/XML files and from *remote EndPoints* via SPARQL queries by means of directives of form: `#import_local_sparql("file","query",pred_name,pred_arity[,types]).` or `#import_remote_sparql("endpoint_url","query",pred_name,pred_arity[,types]).` where `query` is a SPARQL statement defining data to be imported and the optional `types` specifies the conversion for mapping data types to ASP-Core-2 terms. For the local import, `file` can be either a local or remote URL pointing to an RDF/XML file: in the latter case, the file is downloaded and treated as a local RDF/XML file; in any case, the ontology graph is built in memory. As for the remote import, the `endpoint_url` refers to a remote endpoint and building the graph is up to the remote server; this second option might be very convenient in case of large datasets.

Generic Data Access via Python scripts. Input programs can be enriched by external atoms of the form: $\&\mathtt{p}(\mathtt{i}_1, \ldots, \mathtt{i}_n; \mathtt{o}_1, \ldots, \mathtt{o}_m)$, where `p` is the name of a Python function, $\mathtt{i}_1, \ldots,$ $\mathtt{i}_n$ and $\mathtt{o}_1, \ldots, \mathtt{o}_m$ $(n, m \geq 0)$ are input and output terms, respectively. For each instantiation $\mathtt{i}'_1, \ldots, \mathtt{i}'_n$ of the input terms, function `p` is called with arguments $\mathtt{i}'_1, \ldots, \mathtt{i}'_n$, and returns a set of instantiations for $\mathtt{o}_1, \ldots, \mathtt{o}_m$. For instance, a single line of Python: `def rev(s): s[::-1]` is sufficient to define a function `rev` that reverses strings, and which can be used within a rule of the following form: $revWord(Y) \text{:-} word(X), \&\mathtt{rev}(\mathtt{X}; \mathtt{Y})$. External atoms give the user a powerful tool for significantly extending interoperability, granting access to virtually unlimited external data sources. Hence, additional import/-export features to specific semistructured or unstructured data sources can be externally defined by suitable Python scripts. Obviously, "native" support for interoperability should be preferred whenever available. In fact, it is intuitive to understand that native support allows much better performance; experiments

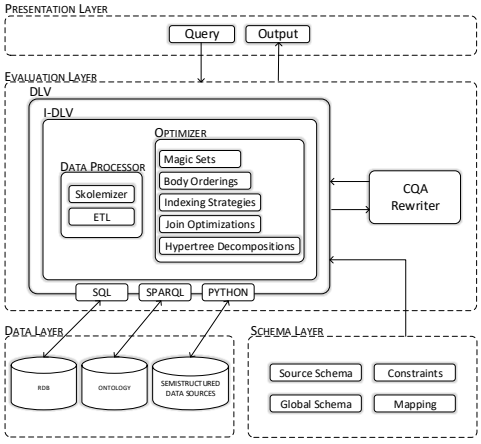

Fig. 1. Architecture of a data integration system based on $\mathcal{I}$-DLV.

reported in [6] give an idea of the effective gain on performance obtainable with a native support of SQL/SPARQL local import directives against the same directives implemented via Python scripts.

Value invention/modification. The availability of both external atoms and function symbols, included in the ASP-Core-2 compliance, allows to address very interesting issues from a database perspective. First of all, it is well known that function symbols allow to implement value invention by skolemization. This turns out to be a very useful feature when dealing with ontologies. Moreover, the generality of external atoms allows to include in logic rules data modification processes, typical of ETL workflows. In [26] we already described how external atoms may help data cleaning processes in a logic-based scenario.

3 Application of $\mathcal{I}$-DLV Features for Data Integration

The adoption of deductive database technology for data integration solutions is not new [16, 17, 25, 19, 23]; however, the recent developments on ASP described in this paper, allow a more concrete application of deductive systems in real-world applications requiring integration of heterogeneous data such as RDBMS, Ontologies and Semi-structured information sources. A general architecture for a modern integration system based on $\mathcal{I}$-DLV is presented in Figure 1, where both main architectural elements and specific $\mathcal{I}$-DLV functionalities oriented to data integration are highlighted; these will be described next by *layers*.

The *Data Layer*, which comprises the set of input information sources, can handle several kind of data types: *(i)* standard databases can be directly accessed through the `import_sql` directives included in $\mathcal{I}$-DLV; *(ii)* graph databases, RDF ontologies, and more generally SPARQL-enabled ontologies, can be accessed by

the `import_local_sparql` and `import_remote_sparql` directives; *(iii)* interoperability with any other kind of input format can be granted by external atoms relying on suitable Python scripts.

The *Schema Layer* includes everything that describes the data integration context from a conceptual point of view, namely source and global schemas, mappings and constraints, in a way similar to what has been widely studied in the literature [16]. The support to this design phase could be provided by already available external graphical tools, such as the one presented in [11].

The *Evaluation Layer* includes everything that allows to transform input data, schemas, and queries into answers in an effective way. The core role is played by $\mathcal{I}$-DLV which, as previously pointed out, has been incorporated as the grounding module of the DLV system. Here we concentrate our attention on three main logical portions: the *Data Processor*, the *Optimizer*, and the *CQA Rewriter*.

The *Data Processor* highlights some of the advanced functionalities included in $\mathcal{I}$-DLV; in more detail, the general capabilities of Python-based external atoms put into play the possibility to include ETL processes inside the ASP engine. This is a particularly interesting innovation, since reasoning on deductive databases usually excluded ETL processes that were confined to external workflows. Moreover, ASP-Core-2 compliance of the $\mathcal{I}$-DLV language implies the possibility to exploit function symbols as predicate arguments; in a database oriented setting, this allows to easily simulate skolemization. This is a particularly interesting feature when ontologies are among the inputs; in fact, it is well known that, in particular cases, value invention in ontologies can be handled via skolemization. This opportunity significantly expands data integration potentialities of the system w.r.t. existing proposals. It is worth observing that, in a parallel project involving DLV, a more general extension of ASP supporting existentially quantified rule heads, and consequently more complex axioms in ontologies, named DLV$^{\exists}$, has been proposed [18]; however this language extension and the corresponding evaluation engine is not included in $\mathcal{I}$-DLV yet.

The *Optimizer* applies to the resulting ASP program all database oriented optimizations previously outlined, and included in $\mathcal{I}$-DLV. In more detail, magic sets, join optimizations, hypertree decompositions, body orderings and indexing strategies may altogether provide crucial speedup in query answering processes, thus allowing the adoption of the system in real application scenarios.

In order to complete the picture relative to the *Evaluation Layer*, it is worth observing that, if the global schema is equipped with constraints that must be satisfied during data integration, *Consistent Query Answering* techniques and optimizations such as the ones presented in [21, 22] can be applied. In Figure 1, this is represented as a functionality external to DLV since it is not included inside the engine yet. However, it would be straightforward to incorporate them inside the system since they are based on rewritings of ASP programs.

Finally, the *Presentation Layer* is devoted to allow users to compose queries and get the corresponding results. Again available external graphical tools [11] can support this phases.

4 Future Work

In this paper we briefly reported on the most recent advancements on the deductive system $\mathcal{I}$-DLV for database oriented features, and we have shown their application to a data integration setting. In particular, reported features clearly show that data integration is still a very active and promising research area, which is kept strongly alive by new challenges arising from ontologies, semi-structured and unstructured information sources. Given the positive results in terms of efficiency and extensibility we obtained for the $\mathcal{I}$-DLV system, first of all we plan to incorporate in $\mathcal{I}$-DLV the features already developed in parallel projects, such as the CQA rewriting and optimizations techniques and the support to existential rules introduced in DLV$^{\exists}$ for ontology querying. Moreover, we plan to explicitly implement connectors to different data formats. As a matter of facts, reasoning on top of big data is also part of ongoing projects in the research group.

References

1. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: LPNMR. Lecture Notes in Computer Science, vol. 10377, pp. 215–221. Springer (2017)
2. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Artif. Intell. **187**, 156–192 (2012)
3. Arenas, M., Bertossi, L.E., Chomicki, J.: Specifying and Querying Database Repairs using Logic Programs with Exceptions. In: Larsen, H.L., Kacprzyk, J., Zadrozny, S., Andreasen, T., Christiansen, H. (eds.) Proceedings of the Fourth International Conference on Flexible Query Answering Systems (FQAS 2000) (2000)
4. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: IJCAI. pp. 16–21. Morgan Kaufmann (2003)
5. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Ricca, F., Schaub, T.: Asp-core-2: Input language format. ASP Standardization Working Group, Tech. Rep (2012)
6. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: External computations and interoperability in the new DLV grounder. In: AI*IA. Lecture Notes in Computer Science, vol. 10640, pp. 172–185. Springer (2017)
7. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: I-DLV: the new intelligent grounder of DLV. Intelligenza Artificiale **11**(1), 5–20 (2017). https://doi.org/10.3233/IA-170104, http://dx.doi.org/10.3233/IA-170104
8. Calimeri, F., Fuscà, D., Perri, S., Zangari, J.: Optimizing answer set computation via heuristic-based decomposition. In: PADL. Lecture Notes in Computer Science, vol. 10702, pp. 135–151. Springer (2018)
9. Calimeri, F., Gebser, M., Maratea, M., Ricca, F.: Design and results of the fifth answer set programming competition. Artif. Intell. **231**, 151–181 (2016)
10. Calimeri, F., Perri, S., Fuscà, D., Zangari, J.: $\mathcal{I}$-DLV homepage (since 2016), https://github.com/DeMaCS-UNICAL/I-DLV/wiki
11. Febbraro, O., Grasso, G., Leone, N., Reale, K., Ricca, F.: Datalog development tools - (extended abstract). In: Datalog. Lecture Notes in Computer Science, vol. 7494, pp. 81–85. Springer (2012)

12. Fuscà, D., Calimeri, F., Zangari, J., Perri, S.: I-DLV+MS: preliminary report on an automatic ASP solver selector. In: RCRA@AI*IA. CEUR Workshop Proceedings, vol. 2011, pp. 26–32. CEUR-WS.org (2017)
13. Gebser, M., Kaminski, R., Kaufmann, B., Romero, J., Schaub, T.: Progress in clasp series 3. In: LPNMR. Lecture Notes in Computer Science, vol. 9345, pp. 368–383. Springer (2015)
14. Gebser, M., Maratea, M., Ricca, F.: The sixth answer set programming competition. J. Artif. Intell. Res. **60**, 41–95 (2017)
15. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-tolerant query answering in ontology-based data access. J. Web Sem. **33**, 3–29 (2015)
16. Lenzerini, M.: Data integration: A theoretical perspective. In: PODS. pp. 233–246. ACM (2002)
17. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005). pp. 915–917. ACM Press, Baltimore, Maryland, USA (Jun 2005)
18. Leone, N., Manna, M., Terracina, G., Veltri, P.: Efficiently computable datalog$\exists$ programs. In: KR. AAAI Press (2012)
19. Leone, N., Ricca, F., Rubino, L.A., Terracina, G.: Efficient application of answer set programming for advanced data integration. In: PADL. Lecture Notes in Computer Science, vol. 5937, pp. 10–24. Springer (2010)
20. Liang, S., Fodor, P., Wan, H., Kifer, M.: OpenRuleBench: An Analysis of the Performance of Rule Engines. In: Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009. pp. 601–610. ACM (2009). https://doi.org/10.1145/1526709.1526790, http://doi.acm.org/10.1145/1526709.1526790
21. Manna, M., Ricca, F., Terracina, G.: Consistent query answering via ASP from different perspectives: Theory and practice. Theory and Practice of Logic Programming **13**(2), 277–252 (2013)
22. Manna, M., Ricca, F., Terracina, G.: Taming primary key violations to query large inconsistent data via ASP. TPLP **15**(4-5), 696–710 (2015). https://doi.org/10.1017/S1471068415000320, http://dx.doi.org/10.1017/S1471068415000320
23. Nardi, B., Reale, K., Ricca, F., Terracina, G.: An integrated environment for reasoning over ontologies via logic programming. In: RR. Lecture Notes in Computer Science, vol. 7994, pp. 253–258. Springer (2013)
24. Swift, T., Warren, D.S.: XSB: Extending Prolog with Tabled Logic Programming. Theory and Practice of Logic Programming **12**(1-2), 157–187 (2012). https://doi.org/10.1017/S1471068411000500, http://dx.doi.org/10.1017/S1471068411000500
25. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory and Practice of Logic Programming **8**, 129–165 (2008)
26. Terracina, G., Martello, A., Leone, N.: Logic-based techniques for data cleaning: An application to the italian national healthcare system. In: LPNMR. Lecture Notes in Computer Science, vol. 8148, pp. 524–529. Springer (2013)
27. Ullman, J.D.: Principles of Database and Knowledge-Base Systems, Volume I. Computer Science Press (1988)