Data Integration and Answer Set Programming

Thomas Eiter

Knowledge Based Systems Group, Institute of Information Systems,
Vienna University of Technology, A-1040 Vienna, Austria
eiter@kr.tuwien.ac.at

Abstract. The rapid expansion of the Internet and World Wide Web led to growing interest in data and information integration, which should be capable to deal with inconsistent and incomplete data. Answer Set solvers have been considered as a tool for data integration systems by different authors. We discuss why data integration can be an interesting model application of Answer Set programming, reviewing valuable features of non-monotonic logic programs in this respect, and emphasizing the role of the application for driving research.

1 Introduction

Triggered by the rapid expansion of the Internet and the World Wide Web, the integration of data and information from different sources has emerged as a crucial issue in many application domains, including distributed databases, cooperative information systems, data warehousing, or on-demand computing.

However, the problem is complex, and no canonical solution exists. Commercial software solutions such as IBM's Information Integrator [1] and academic systems (see e.g. [2]) fulfill only partially the ambitious goal of integrating information in complex application scenarios. In particular, handling inconsistent and/or incomplete data is, both semantically and computationally, a difficult issue, and is still an active area of research; for a survey of query answering on inconsistent databases, see [3].

In recent years, there has been growing interest in using non-monotonic logic programs, most prominently answer set solvers like DLV [4], Smodels [5], or Cmodels-2 [6] as a tool for data integration, and in particular to reconcile data inconsistency and incompleteness, e.g. [7,8,9,10,11,12,13,14,15]. In our opinion, data integration can in fact be viewed as an interesting model application of Answer Set Programming (ASP), for a number of different reasons:

1. The problem is important. There is rapidly growing interest in data and information integration, and this was estimated to be a $10 Billion market by 2006 [16].
2. Some of the key features of non-monotonic logic programming and ASP in particular, namely declarativity, expressiveness, and capability of nondeterminism can be fruitfully exploited.
3. Interest in ASP engines as a tool for solving data integration tasks emerged with people outside the ASP community, and in fact by different groups [7,8,11,12,13].
4. The application has raised new research problems and challenges, which have driven research to enhance and improve current ASP technology.

C. Baral et al (Eds.): LPNMR 2005, LNAI 3662, pp. 13–25, 2005.
© Springer-Verlag Berlin Heidelberg 2005

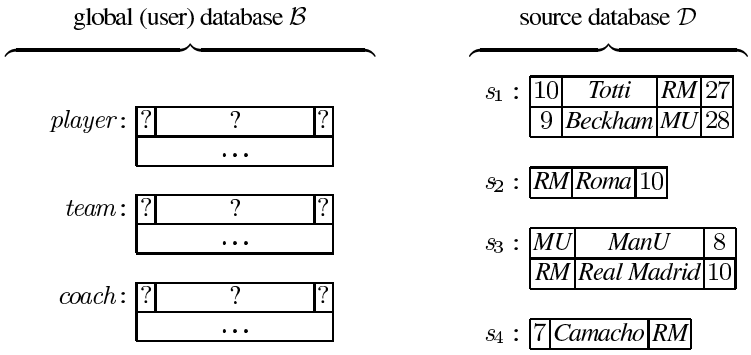

Fig. 1. Simple soccer data integration scenario – global and source relations

Example 1. To illustrate the problem, we consider a simple scenario of a data integration system which provides information about soccer teams. At the global (user) level, there are three relations

$player(Pcode, Pname, Pteam)$, $team(Tcode, Tname, Tleader)$, and
$coach(Ccode, Cname, Cteam)$,

which are interrelated with source relations

$s_1(A1, A2, A3, A4)$, $s_2(B1, B2, B3)$, $s_3(C1, C2, C3)$, and $s_4(D1, D2, D3)$

in the following way:

- $player$ correlates with the projection of s_1 to first three attributes;
- $team$ correlates with the union of s_2 and s_3; and
- $coach$ correlates with s_4.

(The precise form of correlation will be detailed later.) Now given the instance of the source database shown in Figure 1, how should a corresponding global database instance look like? In particular, if there are key constraints for the user relations, and further constraints like that a coach can neither be a player nor a team leader. Furthermore, if we want to pose a query which retrieves all players from the global relations (in logic programming terms, evaluate the rules

$$q(X) \leftarrow player(X, Y, Z)$$
$$q(X) \leftarrow team(V, W, X)$$

where q is a query predicate), how do we semantically determine the answer? □

Different approaches to data integration have been considered; see [17,2] for discussion. The most prominent ones are the *Global As View approach (GAV)*, in which the relations at the user level are amount to database views over the sources, and *Local As View approach (GAV)*, in which conversely the relations of the source database are

database views on the global relations. For both approaches, the usage of ASP has been explored, cf. [7,8,10,11,12,13,15].

In this remainder of this paper, we shall first briefly present a framework for data integration from the literature [17,18] which accommodates both GAV and LAV, along with proposals for semantics to deal with data inconsistencies. We then discuss why employing non-monotonic logic programs for this application is attractive, but also what shortcomings of ASP technology have been recognized, which have been driving (and still do so) research to improve ASP technology in order to meet the needs of this application. This is further detailed on the example of the INFOMIX information integration project, in which ASP has been adopted as the core computational technology to deal with data inconsistencies. We conclude with some remarks and issues for future research.

2 Data Integration Systems

While semi-structured data formats and in particular XML are gaining more and more importance in the database world, most of the theoretical work on advanced data integration has considered traditional relational databases, in which a database schema is modeled as a pair $\langle \Psi, \Sigma \rangle$ of a set Ψ of database relations and a set Σ of integrity constraints on them. The latter are first-order sentences on Ψ and the underlying (finite or infinite) set of constants (elementary values) *Dom*. A database instance can be viewed as a finite set of ground facts on Ψ and *Dom*, and is legal if it satisfies Σ.

A commonly adopted high-level structure of a data integration system $\mathcal{I}$ in a relational setting is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ with the following components [17,18]:

1. $\mathcal{G} = \langle \Psi, \Sigma \rangle$ is a relational schema called the *global schema*, which represents the user's view of the integrated data. The integrity constraints Σ are usually from particular constraint classes, since the interaction of constraints can make semantic integration of data undecidable. Important classes of constraints are key constraints and functional dependencies (as well-known from any database course), inclusion dependencies (which enforce presence of certain tuples across relations), and exclusion dependencies (which forbid joint presence of certain tuples).
2. $\mathcal{S}$ is the *source schema*, which is given by the schemas of the various sources that are part of the data integration system. Assuming that they have been standardized apart, we can view $\mathcal{S}$ as a relational schema of the form $\mathcal{S} = \langle \Psi', \Sigma' \rangle$. The common setting is that Σ' is assumed to be empty, since the sources are autonomous and schema information may be not disclosed to the integration system.
3. $\mathcal{M}$ is the *mapping*, which establishes the relationship between $\mathcal{G}$ and $\mathcal{S}$ in a semantic way. The mapping consists of a collection of *mapping assertions* of the forms

$$(1) \quad q_{\mathcal{G}}(\boldsymbol{x}) \sqsubseteq q_{\mathcal{S}}(\boldsymbol{x}) \quad \text{and} \; (2) \quad q_{\mathcal{S}}(\boldsymbol{x}) \sqsubseteq q_{\mathcal{G}}(\boldsymbol{x}),$$

   where $q_{\mathcal{G}}(\boldsymbol{x})$ and $q_{\mathcal{S}}(\boldsymbol{x})$ are database queries (typically, expressible in first-order logic) with the same free variables $\boldsymbol{x}$, on $\mathcal{G}$ respectively $\mathcal{S}$.

In the above framework, the GAV and LAV approach result as special cases by restricting the queries $q_{\mathcal{G}}(\boldsymbol{x})$ respectively $q_{\mathcal{S}}(\boldsymbol{x})$ to atoms $r(\boldsymbol{x})$.

Example 2. (cont'd) Our running example scenario is represented as a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ consists of the relations $player$, $team$, and $coach$. The associated constraints Σ are that the keys of $player$, $team$, and $coach$ are the attributes $\{Pcode, Pteam\}$, $\{Tcode\}$, and $\{Ccode, Cteam\}$, respectively, and that a coach can neither be a player nor a team leader. The source schema $\mathcal{S}$ comprises the relations s_1, s_2, s_3 and s_4. Finally, the GAV mapping $\mathcal{M}$ is defined in logic programming terms as follows (where $q_S \sqsubseteq q_G$ amounts to $q_G \leftarrow q_S$):

$$\begin{aligned} player(X, Y, Z) &\leftarrow s_1(X, Y, Z, W) \\ team(X, Y, Z) &\leftarrow s_2(X, Y, Z) \\ team(X, Y, Z) &\leftarrow s_3(X, Y, Z) \\ coach(X, Y, Z) &\leftarrow s_4(X, Y, Z) \end{aligned}$$

□

The formal semantics of a data integration system $\mathcal{I}$ is defined with respect to a given instance $\mathcal{D}$ of the source schema, $\mathcal{S}$, in terms of the set $sem(\mathcal{I}, D)$ of all instances $\mathcal{B}$ of the global schema, $\mathcal{G}$, which satisfy all constraints in $\mathcal{G}$ and, moreover,

- $\{\boldsymbol{c} \mid \mathcal{B} \models q_G(\boldsymbol{c})\} \subseteq \{\boldsymbol{c} \mid \mathcal{D} \models q_S(\boldsymbol{c})\}$ for each mapping assertion of form (1), and
- $\{\boldsymbol{c} \mid \mathcal{D} \models q_S(\boldsymbol{c})\} \subseteq \{\boldsymbol{c} \mid \mathcal{B} \models q_G(\boldsymbol{c})\}$ for each mapping assertion of form (2),

where for any constants $\boldsymbol{c}$ on *Dom*, $\mathcal{DB} \models q(\boldsymbol{c})$ denotes that $q(\boldsymbol{c})$ evaluates to true on the database $\mathcal{DB}$.

The notions of *sound*, *complete*, and *exact mapping* between query expressions $q_G(\boldsymbol{x})$ and $q_S(\boldsymbol{x})$ [17], reflecting assumptions on the source contents, are then elegantly captured as follows:

- sound mapping: $q_S(\boldsymbol{x}) \sqsubseteq q_G(\boldsymbol{x})$ (intuitively, some data in the sources is missing),
- complete mapping: $q_G(\boldsymbol{x}) \sqsubseteq q_S(\boldsymbol{x})$ (intuitively, the sources contain excess data),
- exact mapping: $q_S(\boldsymbol{x}) \sqsubseteq q_G(\boldsymbol{x}) \wedge q_G(\boldsymbol{x}) \sqsubseteq q_S(\boldsymbol{x})$

The answer to a query Q with out query predicate $q(\boldsymbol{x})$ against a data integration system $\mathcal{I}$ with respect to source data $\mathcal{D}$, is given by the set of tuples $ans(Q, \mathcal{I}, \mathcal{D}) = \{\boldsymbol{c} \mid \mathcal{B} \models q(\boldsymbol{c}),$ for each $\mathcal{B} \in sem(\mathcal{I}, \mathcal{D})\}$; that is, $ans(Q, \mathcal{I}, \mathcal{D})$ collects all tuples $\boldsymbol{c}$ on *Dom* such that $q(\boldsymbol{c})$ is a skeptical consequence with respect to all "legal" global databases.

In a GAV setting under sound mappings, the smallest candidate database $\mathcal{B}$ for $sem(\mathcal{I}, \mathcal{D})$, is given by the *retrieved global database*, $ret(\mathcal{I}, \mathcal{D})$, which is the materialization of all the views on the sources. Under exact mappings, $ret(\mathcal{I}, \mathcal{D})$ is in fact the only candidate database for $sem(\mathcal{I}, \mathcal{D})$.

Example 3. (cont'd) Note that the mapping $\mathcal{M}$ in our running example is a sound mapping. The global database $\mathcal{B}_0 = ret(\mathcal{I}, \mathcal{D})$ for $\mathcal{D}$ as in Figure 1 is shown in Figure 2. It violates the key constraint on $team$, witnessed by the two facts $team(\textit{RM}, \textit{Roma}, 10)$ and $team(\textit{RM}, \textit{Real Madrid}, 10)$, which coincide on $Tcode$ but differ on $Tname$. Since this key constraint is violated in every database $\mathcal{B}'$ which contains $\mathcal{B}_0$, it follows that $sem(\mathcal{I}, \mathcal{D})$ is empty, i.e., the global relations can not be consistently populated. □

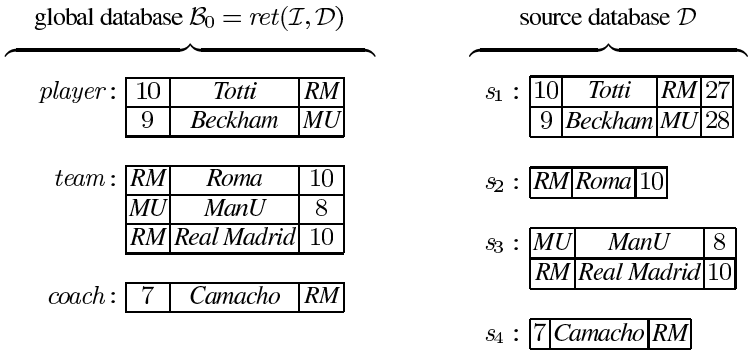

Fig. 2. Global database $\mathcal{B}_0 = ret(\mathcal{I}, \mathcal{D})$ for the soccer scenario as retrieved from the sources

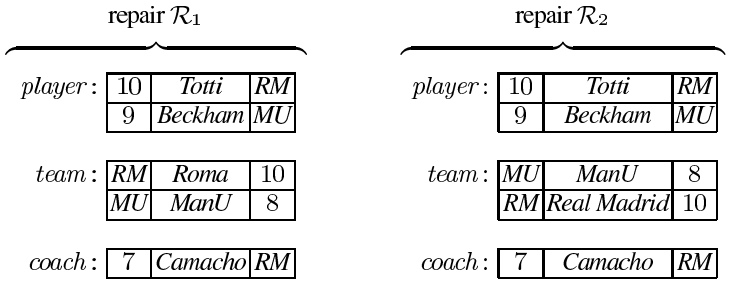

Fig. 3. Repairs in the example data integration scenario $\mathcal{I}$ w.r.t. $\mathcal{D}$

Since "ex-falso-quodlibet" is not a desirable principle for database query answers (e.g., in our scenario $(Roma)$ would be a query answer), relaxations aim at adopting global databases $\mathcal{B}$ which (1) satisfy all constraints and (2) satisfy the mapping assertion $\mathcal{M}$ with respect to $\mathcal{D}$ *as much as possible*. The latter may be defined in terms of a preference ordering (i.e., a reflexive and transitive relation) $\succeq$ over global databases $\mathcal{B}$, such that those $\mathcal{B}$ are accepted which are most preferred. Different possibilities for instantiating $\succeq$ exist and have been considered in a number of papers [7,8,19,10,11,12,20,13,21,15]. A popular one with GAV mappings is the one which prefers databases $\mathcal{B}$ which are as close as possible, under symmetric set difference, to the retrieved global databases $ret(\mathcal{I}, \mathcal{D})$; it amounts to the smallest set of tuples to be added and/or deleted. Other orderings are based on giving preferences to retaining tuples in $ret(\mathcal{I}, \mathcal{D})$ over adding new ones (as in loosely sound semantics [20,13]), or even demand that only deletion of tuples is allowed (guided by some completeness assumption [21]). These databases are commonly referred to as "repairs" of the global database $ret(\mathcal{I}, \mathcal{D})$.

Example 4. (cont'd) Let us adopt the preference relation $\succeq_{\mathcal{B}_0}$ which prefers $\mathcal{B}_1$ over $\mathcal{B}_2$ if the symmetric difference $\mathcal{B}_1 \triangle \mathcal{B}_0$ is a subset of $\mathcal{B}_2 \triangle \mathcal{B}_0$. Then the retrieved global database $\mathcal{B}_0$ for the running example has two repairs $\mathcal{R}_1$ and $\mathcal{R}_2$ shown in Figure 3.

Accordingly, the example query Q evaluates to $ans(Q, \mathcal{I}, \mathcal{D}) = \{(8), (9), (10)\}$, since the respective facts $q(c)$ are derived over both repairs. □

For a more rigorous discussion of the above model, in particular from the perspective of logic, we refer to [18], where it is also shown that a number of different orderings $\succeq$ used by different authors can be generically captured.

3 Employing Non-monotonic Logic Programs

Recently, several approaches to formalize repair semantics by using non-monotonic logic programs have been proposed, cf. [7,8,9,19,10,11,12,13,14,15]. The idea common to most of these works is to encode the constraints Σ of the global schema $\mathcal{G}$ into a function-free logic program, Π, using unstratified negation and/or disjunction, such that the answer sets of this program [22] yield the repairs of the global database. Answering a user query, Q, then amounts to cautious reasoning over the logic program Π augmented with the query, cast into rules, and the retrieved database $\mathcal{B}$. For an elaborated discussion of these proposals, we refer to [23]. In [19], the authors consider data integration via abductive logic programming, where, roughly speaking, the repairs of a database are computed as abductive explanations.

In the following, we discuss some key features of answer set programs which can be fruitfully exploited for doing data integration via ASP.

High expressiveness. ASP is a host for complex database queries. ASP with disjunction has Π_2^P-expressiveness, which means that each database query with complexity in Π_2^P can be expressed in this formalism [24]. Such expressiveness is strictly required for query answering in some settings [13,21,25]; for example, under loosely-exact semantics in the presence of inclusion dependencies for fixed queries, but also for ad hoc queries under absence of such dependencies. And, ASP with disjunction has co-NEXP$^{\text{NP}}$ program complexity [24], and thus very complex problems can be polynomially reduced to it (as for query answering, however, not always in a data-independent manner). For more on complexity and expressiveness of ASP, see [26,27].

Declarative language. ASP supports a fully declarative, rule-based approach to information integration. While languages such as Prolog also support declarative integration, the algorithm-oriented semantics makes it more difficult to design and understand integrations policies for the non-expert.

Nondeterminism. ASP engines have been originally geared towards model generation (i.e., computation of one, multiple, or all answer sets) rather than towards theorem proving. This is particularly useful for solving certain AI problems including model-based diagnosis, where given some observations and a background theory, a model is sought which reconciles the actual and predicted observations in terms of assumptions about faulty components applying Occam's Razor.

Repair semantics for query answering from inconsistent data bases and integration systems [7,8,10,11,12,13,15] is closely related to this diagnostic problem. Using an ASP engine, a particular repair for a corrupted database might be computed and installed in its place.

Proximity to database query languages. ASP programs are close to logic-based database query languages such as conjunctive queries, union of conjunctive queries, (plain) SQL, and datalog. This in particular facilitates a seamless integration of various components of a data integration system – query evaluation, database "repair", and database mapping – into a uniform language.

Executable specifications. Given the seamless integration of various components of a data integration system, an ASP program for information integration tasks can be regarded as an executable specification, which can be run on supplied input data. This has been elaborated in [14,23] where abstract logic specification for querying a GAV data integration $\mathcal{I}$ with a query Q has been considered in terms of a hierarchically composed disjunctive datalog program $\Pi_{\mathcal{I}}(Q) = \Pi_{\mathcal{M}} \cup \Pi_{\Sigma} \cup \Pi_{Q}$ such that (in simplified notation):

1. $ret(\mathcal{I}, \mathcal{D}) \rightleftharpoons \mathrm{AS}(\Pi_{\mathcal{M}} \cup \mathcal{D})$, where $\Pi_{\mathcal{M}}$ is a stratified normal datalog program, computing the mapping $\mathcal{M}$;
2. $rep_{\mathcal{I}}(\mathcal{D}) \rightleftharpoons \mathrm{AS}(\Pi_{\Sigma} \cup ret(\mathcal{I}, \mathcal{D}))$, where Π_{Σ} is an (unstratified resp. disjunctive) program computing the repairs, and
3. $ans(Q, \mathcal{I}, \mathcal{D}) = \{\boldsymbol{c} \mid q(\boldsymbol{c}) \in M$ for each $M \in \mathrm{AS}((\Pi_{\mathcal{M}} \cup \Pi_{\Sigma} \cup \Pi_{Q} \cup \mathcal{D})\}$, where Π_{Q} is a non-recursive safe datalog program with negation (defining the query output predicate q);

here, $\mathrm{AS}(\mathcal{P})$ are the answer sets of a program $\mathcal{P}$ and $\rightleftharpoons$ denotes a polynomial-time computable correspondence between two sets.

Language constructs. The original ASP language [22] has been enriched with a number of constructs, including different forms of constraints such as DLV's weak constraints, Smodels's choice rules and weight constraints, rule preferences (see e.g. [28] for a survey), and more recently aggregates (see [29,30,31] and references therein).

Aggregates, for instance, are very desirable for querying databases (SQL provides many features in this respect). Weak and weight constraints are convenient for specifying certain integration semantics, such as repairs based on cardinality (Hamming) distance of changes to the retrieved global database. Rule preferences might be exploited to expressing preferences in repair, and e.g. help to single out a canonical repair. Therefore, refinements and variants of standards proposals for integration semantics can be accomodated well.

Knowledge representation capability. ASP provides, thanks to the availability of facts and rules, different kinds of negation (strong and weak i.e. default negation), a rich language for representing knowledge. This makes ASP attractive for crafting special, domain dependent integration policies, in which intuitively a knowledge-base is run for determining the best integration result.

4 Emerging ASP Research Issues

While ASP is attractive as a logic specification formalism for data integration, and available ASP engines can be readily applied for rapid prototyping of experimental systems

that work well on small examples, it becomes quickly apparent that ASP technology needs to be seriously improved in order to meet the requirements of this application and make its usage feasible in practice. Among others, the following important issues emerge:

Scalability. In real data integration scenarios, one needs to deal with massive amounts of data (in the Gigabytes and beyond), rather than with a few tuples. A graceful scaling of an integration system's response time with respect to the amount of data processed is desired. A problem in this respect is that current ASP technology builds on program grounding, i.e., the reduction of non-ground programs to ground (propositional) programs which are then evaluated with special algorithms. Even though the grounding strategies of DLV and Smodels' grounder Lparse are highly sophisticated and avoid as much as possible the generation of "unnecessary rules," the grounding of a repair program over a large dataset will be ways too large to render efficient query answering. Therefore, optimization methods are needed which allow for handling large amounts of data. A call for this has been made e.g. in [11].

Nonground queries. Another issue is that as seen in the example scenario, queries to an integration system typically contain variables, all whose instances should be computed. ASP solvers, however, have been conceived for model computation rather than for query answering. Query answering, e.g. as originally supported in DLV, was limited to ground (variable-free) queries (which can be reduced to model computation resp. checking program consistency by simple transformations). The straightforward method of reducing a non-ground query by instantiation to a series of (separate) ground queries is not efficient, since roughly the system response time will be $O(\#gq * srt)$, where $\#gq$ is the number of ground queries and srt is the response time for a single ground query. Therefore, efficient methods for answering non-ground queries are needed.

Software interoperability. In data integration settings in practice, data is stored in multiple repositories, and often in heterogeneous formats. In a relational setting, such repositories will be managed by a commercial DBMS such as Oracle, DB2, SQLServer etc. Data exchange between a DBMS and an ASP engine via files or other operating systems facilities, as was the only possibility with early ASP systems, requires extra development effort and, moreover, is an obvious performance bottleneck. To facilitate efficient system interoperability, suitable interfaces from ASP solvers to DBMS must be provided, such as an ODBC interface as available in other languages. Furthermore, interfaces to other software for carrying out specific tasks in data integration (e.g., data cleaning, data presentation) are desirable.

These issues are non-trivial and require substantial foundational and software development work. Scalability and non-ground query answering are particularly challenging issues, which are not bound to the data integration application. Advances on them will be beneficial to a wide range of other applications as well.

The above issues have been driving some of the research on advancing and enhancing ASP technology in the last years, and in particular of the ASP groups at the University of Calabria and at TU Vienna. A number of results have been achieved, which are briefly summarized as follows.

- As for scalability and optimization, the magic set method (introduced in [32]) has been extended to disjunctive programs and tuned for data integration [33,34,35]; focusing techniques and optimization methods genuine to data integration are presented in [14,23]; different variants of repair programs have been examined for their suitability, in this context, recent notions and results on program equivalence [36,37,38,39] turned out to be a useful tool.
- Non-ground query answering is supported in the current releases of DLV.
- Interfacing of relational DBMS is supported by an ODBC interface in DLV [40], and a tight coupling between ASP engines and relational DBMSs has been conceived [41].

These results have been obtained in the course of the INFOMIX project, which is briefly presented in the next subsection, since to our knowledge it is the most comprehensive initiative to employ ASP in a data integration system.

4.1 The INFOMIX Project

INFOMIX [42] has been launched jointly by the ASP groups of the University of Calabria and TU Vienna, the data integration group at the University of Rome "La Sapienza," and Rodan Systems S.A., a Polish database software house, with the objective to provide powerful information integration for handling inconsistent and incomplete information, using computational logic tools as an implementation host for advanced reasoning tasks. The usage of an ASP engine like DLV or Smodels which is capable of handling non-ground programs and provides the expressiveness needed, appeared to be well-suited. However, the research issues mentioned above had to be addressed in order to make employment of ASP in realistic integration scenarios feasible beyond toy examples.

The INFOMIX prototype [43,44] is built on solid theoretical foundations, and implements the GAV approach under sound semantics. It offers the user a powerful query language which, as a byproduct of the usage of ASP, allows in certain cases also queries beyond recursion-free positive queries (which are those expressed by non-recursive ASP programs without negation), in particular stratified queries or queries with aggregates, depending on the setting of the integrity constraints of the global schema; the underlying complexity and undecidability frontier has been charted in [20]. Furthermore, INFOMIX provides the user with tools for specifying and managing a data integration scenario, as well as with a rich layer for accessing and transforming data from sources (possibly dispersed on the Internet) in heterogeneous formats (including relational format, XML, and HTML under constraints) into a homogenous format (conceptually, into a fragment of XML Schema) by data wappers.

At the INFOMIX core are repair logic programs for handling data inconsistencies, which are dynamically compiled by rewriting algorithms. Pruning and optimization methods are applied which aim at reducing the portion of data which need to be accessed for query answering. In particular, the usage of the ASP engine is constrained to the inconsistent data part which needs repair. Details about this can be found in paper and reports [42,13,14,35].

Compared to data integration systems with similar semantics, of which the most prominent are the Hippo [45] and ConQuer [46], INFOMIX is capable of handling a

much larger range of queries and constraint settings which are realistic in practice. This is exemplified by the INFOMIX Demo Scenario, in which data from various legacy databases and web pages of the University of Rome "La Sapienza" are integrated into a global view which has 14 relations and about 30 integrity constraints, including key constraints, inclusion and exclusion dependencies. Most of the 9 typical user queries in the Demo Scenario can't be handled by Hippo; ConQuer can only handle key constraints, and thus is not applicable to the scenario. On the other hand, Hippo and ConQuer are very efficient and faster than INFOMIX on the specific settings which they can handle.

Thanks to improved ASP technology and the optimization techniques, INFOMIX is able to handle the queries in the Demo Scenario reasonably efficient within a few seconds for core integration time, and tends to scale gracefully. Without these improvements and optimizations, the performance is much worse and the system response time barely acceptable.

5 Discussion and Conclusion

As argued above, data integration can be seen as an interesting model application of Answer Set Programming. The results which have been obtained so far are encouraging, and show clear benefits of using ASP. We remark that an experimental comparison of computing database repairs with different logic-based methods – QBF, CLP, SAT, and ASP solvers – in a propositional setting is reported in [9], which shows that ASP performs very well. We suspect that in the realm of a relational setting (in which QBF and SAT solvers can't be directly applied and require preliminary grounding), it behaves even more advantageous.

In spite of the advances that have been achieved on the issues in Section 4, research on them is by no means closed, and in fact a lot of more work is necessary.

Optimization of ASP programs is still at a rather early stage, and there is room for improvement. Currently, optimization is done at the level of ASP solvers, which employ internal optimization strategies that to some extent build on heuristics. Optimization at the "external" level, independent of a concrete ASP solver, is widely unexplored. Recent results on program equivalences (cf. [36,37,38,47,39] and references therein) might provide a useful basis for optimization methods. However, as follows from classic results in database theory, basic static optimization tasks for ASP programs are undecidable in very plain settings (cf. [48]), and thus a detailed study and exploration of decidable cases is needed.

Efficient non-ground query answering is an issue which is perhaps tied to a more fundamental issue concerning the architecture of current state-of-the-art answer set engines: it is unclear whether their grounding approach is well-suited as a computational strategy. Indeed, for programs in which predicate arities are bounded by a constant, non-ground query answering can be carried out in polynomial space (as follows from results in [49]) while current answer set engines use exponential space for such queries in general.

As for software interoperability, ASP engines need to interface a large range of other data formats besides relational data, including popular formats like XML and, more recently, also RDF. For XML data, ASP extensions are desired which allow to manipulate them conveniently. The Elog language [50] may be a guiding example in

this direction. In turn, integration of ASP solvers into more complex software systems needs also better support.

As for future developments of ASP and data integration, one interesting issue would be to realize an operational data integration system which is deployed in a concrete application. The results of INFOMIX are encouraging in this direction. Here, a hybrid system combining complementary approaches like those of Hippo, ConQuer, and INFOMIX would be an intriguing idea. As for the perspective of advanced data integration at the level of a full-fledged commercial DBMS, we feel that research is still at an early stage and industry seems not to be ready for immediate takeup.

There are several interesting directions for further research on the usage of ASP in data integration. Among them is powerful mediated data integration, as discussed e.g. in [19], and peer-to-peer data integration in a network of information systems [51]. Furthermore, applications of ASP in advanced integration of information sources which contain information beyond factual knowledge, and in data model and ontology management might be worthwhile to explore. Thanks to its rich knowledge representation capabilities, ASP might prove to be a valuable tool for developing declarative formalisms in these areas.

Acknowledgments. I am very grateful to Wolfgang Faber for comments on a draft of this paper. I would like to thank all the many colleagues with whom I had the pleasure to discuss about and work on issues on data integration using non-monotonic logic programming, and here in particular the INFOMIX project team. The views expressed here are, however, personal and not necessarily in line with those of INFOMIX folks. Furthermore, I am very grateful to the support of the European Commission of this work under contracts IST-2001-33570 INFOMIX, FET-2001-37004 WASP, and IST-2001-33123 CologNeT, and the Austrian Science Fund (FWF) project P18019-N04.

References

1. Hayes, H., Mattos, N.: Information on demand. DB2 Magazine **8** (2003)
2. Halevy, A.Y.: Data integration: A status report. In: Proc. 10. Conference on Database Systems for Business, Technology and Web (BTW 2003), LNI 26, GI (2003) 24–29
3. Bertossi, L., Chomicki, J.: Query answering in inconsistent databases. In Chomicki, J., van der Meyden, R., Saake, G., eds.: Logics for Emerging Applications of Databases. Springer (2003) 43–83
4. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Transactions on Computational Logic, to appear. Available via `http://www.arxiv.org/ps/cs.AI/0211004`.
5. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. Artificial Intelligence **138** (2002) 181–234
6. Lierler, Y., Maratea, M.: Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. In: Proc. 7th Int'l Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR-7). LNCS 2923, Springer (2004) 346–350
7. Arenas, M., Bertossi, L.E., Chomicki, J.: Specifying and querying database repairs using logic programs with exceptions. In: Proc. 4th Int. Conf. on Flexible Query Answering Systems (FQAS 2000), Springer (2000) 27–41
8. Arenas, M., Bertossi, L.E., Chomicki, J.: Answer sets for consistent query answering in inconsistent databases. Theory and Practice of Logic Programming **3** (2003) 393–424

9. Arieli, O., Denecker, M., Nuffelen, B.V., Bruynooghe, M.: Database repair by signed formulae. In: Proc. 3rd Int'l Symp. on Foundations of Information and Knowledge Systems (FoIKS 2004). LNCS 2942, Springer (2004) 14–30
10. Bertossi, L.E., Chomicki, J., Cortes, A., Gutierrez, C.: Consistent answers from integrated data sources. In: Proc. 6th Int'l Conf. on Flexible Query Answering Systems (FQAS 2002). (2002) 71–85
11. Bravo, L., Bertossi, L.: Logic programming for consistently querying data integration systems. In: Proc. 18th Int'l Joint Conf. on Artificial Intelligence (IJCAI 2003). (2003) 10–15
12. Bravo, L., Bertossi, L.: Deductive databases for computing certain and consistent answers to queries from mediated data integration systems. Journal of Applied Logic **3** (2005) 329–367
13. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. IJCAI 2003. (2003) 16–21
14. Eiter, T., Fink, M., Greco, G., Lembo, D.: Efficient evaluation of logic programs for querying data integration systems. In: Proc. 19th Int'l Conf. on Logic Programming (ICLP 2003). LNCS 2916, Springer (2003) 163–177
15. Greco, G., Greco, S., Zumpano, E.: A logic programming approach to the integration, repairing and querying of inconsistent databases. In: Proc. ICLP 2001. (2001) 348–364
16. Mattos, N.M.: Integrating information for on demand computing. In: Proc. 29th Int'l Conf. on Very Large Data Bases (VLDB 2003). (2003) 8–14
17. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. 21st ACM Symposium on Principles of Database Systems (PODS 2002). (2002) 233–246
18. Andrea Calí, D.L., Rosati, R.: A comprehensive semantic framework for data integration systems. Journal of Applied Logic **3** (2005) 308–328
19. Arieli, O., Denecker, M., Nuffelen, B.V., Bruynooghe, M.: Coherent integration of databases by abductive logic programming. J. Artificial Intelligence Research **21** (2004) 245–286
20. Calì, A., Lembo, D., Rosati, R.: On the decidability and complexity of query answering over inconsistent and incomplete databases. In: Proc. PODS 2003. (2003) 260–271
21. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information and Computation **197** (2005) 90–121
22. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
23. Eiter, T., Fink, M., Greco, G., Lembo, D.: Optimization methods for logic-based consistent query answering over data integration systems. Tech. Report INFSYS RR-1843-05-05, Institute of Information Systems, TU Vienna, Austria (2005). Extends [14]
24. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM Trans. on Database Systems **22** (1997) 364–417
25. Greco, G., Greco, S., Zumpano, E.: A logical framework for querying and repairing inconsistent databases. IEEE Trans. Knowl. Data Eng. **15** (2003) 1389–1408
26. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and Expressive Power of Logic Programming. ACM Computing Surveys **33** (2001) 374–425
27. Marek, V.W., Remmel, J.B.: On the expressibility of stable logic programming. Journal of the Theory and Practice of Logic Programming **3** (2003) 551–567
28. Delgrande, J.P., Wand, K., Schaub, T., Tompits, H.: A classification and survey of preference handling approaches in nonmonotonic reasoning. Computational Intelligence **20** (2004) 308–334
29. Gelfond, M.: Representing Knowledge in A-Prolog. In Kakas, A.C., Sadri, F., eds.: Computational Logic. Logic Programming and Beyond. LNCS 2408, Springer (2002) 413–451
30. Pelov, N.: Semantics of Logic Programs with Aggregates. PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium (2004)

31. Faber, W., Leone, N., Pfeifer, G.: Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity. In: Proc. 9th European Conference on Logics in Artificial Intelligence (JELIA 2004). LNCS 3229, Springer (2004) 200–212
32. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic Sets and Other Strange Ways to Implement Logic Programs. In: Proc. PODS 1986. (1986) 1–16
33. Greco, S.: Binding Propagation Techniques for the Optimization of Bound Disjunctive Queries. IEEE Trans. Knowledge and Data Engineering **15** (2003) 368–385
34. Cumbo, C., Faber, W., Greco, G.: Enhancing the magic-set method for disjunctive datalog programs. In: Proc. ICLP 2004 (2004) 371–385
35. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. In: Proc. 10th Int'l Conf. on Database Theory (ICDT 2005). LNCS 3363, Springer (2005) 306–320
36. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2** (2001) 526–541
37. Lin, F.: Reducing strong equivalence of logic programs to entailment in classical propositional logic. In: Proc. 8th Int'l Conf. on Principles of Knowledge Representation and Reasoning (KR 2002), (2002) 170–176
38. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: Proc. LPNMR-7. LNCS 2923, Springer (2004) 87–99
39. Eiter, T., Fink, M., Woltran, S.: Semantical characterizations and complexity of equivalences in answer set programming. ACM Transactions on Computational Logic, to appear. Tech. Rep. INFSYS RR-1843-05-01, TU Vienna, Austria (2005)
40. Calimeri, F., Citrigno, M., Cumbo, C., Faber, W., Leone, N., Perri, S., Pfeifer, G.: New DLV features for data integration. In: Proc. JELIA 2004. LNCS 3229, Springer (2004) 698–701
41. Leone, N., Lio, V., Terracina, G.: DLV DB: Adding efficient data management features to ASP. In: Proc. LPNMR-7. LNCS 2923, Springer (2004) 341–345
42. INFOMIX homepage (since 2001) `http://sv.mat.unical.it/infomix`.
43. Leone, N., et al.: The INFOMIX system for advanced integration of incomplete and inconsistent data. In: Proc. ACM SIGMOD 2005 Conference, ACM (2005) 915–917
44. Leone, N., et al.: Data integration: a challenging ASP application. In: Proc. LPNMR 2005. LNCS, Springer (2005). This volume
45. Chomicki, J., Marcinkowski, J., Staworko, S.: Computing consistent query answers using conflict hypergraphs. In: Proc. 13th ACM Conference on Information and Knowledge Management (CIKM-2004), ACM Press (2004) 417–426
46. Fuxman, A., Fazli, E., Miller, R.J.: ConQuer: Efficient management of inconsistent databases. In: Proc. ACM SIGMOD 2005 Conference, ACM (2005) 155-166
47. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Strong and uniform equivalence in answer-set programming: Characterizations and complexity results for the non-ground case. In: Proc. 20th National Conference on Artificial Intelligence (AAAI '05) (2005)
48. Halevy, A.Y., Mumick, I.S., Sagiv, Y., Shmueli, O.: Static analysis in datalog extensions. Journal of the ACM **48** (2001) 971–1012
49. Eiter, T., Faber, W., Fink, M., Pfeifer, G., Woltran, S.: Complexity of model checking and bounded predicate arities for non-ground answer set programming. In: Proc. KR 2004 (2004) 377–387
50. Baumgartner, R., Flesca, S., Gottlob, G.: The Elog web extraction language. In: Proc. 8th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2001). LNCS 2250, Springer (2001) 548–560
51. Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Logical foundations of peer-to-peer data integration. In: Proc. PODS 2004 (2004) 241–251