Optimized encodings for Consistent Query Answering via ASP from different perspectives*

Marco Manna, Francesco Ricca, and Giorgio Terracina

Department of Mathematics, University of Calabria, Italy
{manna,ricca,terracina}@mat.unical.it

Abstract. A data integration system provides transparent access to different data sources by suitably combining their data, and providing the user with a unified view of them, called *global schema*. However, source data are generally not under the control of the data integration process, thus integrated data may violate global integrity constraints even in presence of locally-consistent data sources. In this scenario, it may be anyway interesting to retrieve as much consistent information as possible. The process of answering user queries under global constraint violation is called *consistent query answering* (CQA). Several notions of CQA have been proposed, e.g. depending on whether the information in the database is assumed to be *sound* or *complete*. This paper provides a contribution in this setting; it uniforms solutions coming from different perspectives under a common core and provides some optimizations designed to identify and isolate the inefficient part of the computation of consistent answers. The effectiveness of the approach is evidenced by experimental results reported in the paper.

1 Introduction

The enormous amount of information dispersed over many data sources, often stored in different heterogeneous databases, has boosted in recent years the interest for data integration systems [11]. Roughly speaking, a data integration system provides transparent access to different data sources by suitably combining their data, and providing the user with a unified view of them, called *global schema*.

In many cases the application domain imposes some consistency requirements on integrated data. For instance, it may be at least desirable to impose some integrity constraints (ICs), e.g. primary/foreign keys, on global relations.

However, data stored at the sources may violate global ICs when integrated, since in general source data are not under the control of the data integration process. The standard approach to this problem basically consists of explicitly modifying the data in order to eliminate violation of ICs (data cleaning). However, the explicit repair of data is not always convenient or possible. Therefore, when answering a user query, the system should be able to "virtually repair" relevant data in a declarative fashion (in the line of [11, 2, 4, 7]), in order to provide consistent answers (this task is also called consistent query answering, or CQA).

* This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009.

The database community has spent many efforts in this area, relevant research results have been obtained to clarify semantics, decidability, and complexity of data-integration under constraints and, specifically, for CQA. In particular, several notions of CQA have been proposed (see [4] for a survey), e.g. depending on whether the information in the database is assumed to be *sound* or *complete*. Basically, the completeness assumption coincides with the *closed world assumption*, where facts missing from the database are considered to be false, whereas the soundness assumption coincides with the *open world assumption*, where information in the database can be enriched with missing information needed to satisfy the global constraints. Other notions weaken above requirements allowing some forms of exceptions [6].

However, while efficient systems are already available for simple data integration scenarios, scalable solutions have not been implemented yet for CQA, mainly due to the fact that handling inconsistencies arising from constraints violation is inherently hard.

This paper provides a contribution in this setting. Specifically, it starts from state-of-the-art approaches on different semantic perspectives [2, 4, 7, 12] and revisits them in the light of the experience we gained in the INFOMIX [12] project in order to overcome the limitations experienced in real-world scenarios. In particular, we provide a purely declarative, logic-based approach to the problem, featuring several optimizations designed to "localize" and limit the inefficient part of the computation of consistent answers to small fragments of the input, yet allowing interesting classes of queries and constraints on the global schema.

The main characteristics of the proposed approach are the following:

- Well known semantics assumptions on consistent query answering are supported.
- Declarative programming is exploited to uniformly express known semantics via Answer Set Programming (ASP) and to define a common "core" of optimizations.
- The problem of consistent query answering is reduced to cautious reasoning on (possibly) disjunctive Datalog programs; this allows to effectively compute the query results precisely, by using a state-of-the-art disjunctive Datalog system.
- Large amounts of data, typical of real-world integration scenarios, are handled using as internal query evaluator the DLV^{DB} [15, 14] system, which allows for mass-memory database evaluations and distributed data management features.

In order to assess the effectiveness of the proposed optimizations, we carried out experimental activities on a real world scenario comparing their behavior on different semantics.

The plan of the paper is as follows. Section 2 formally introduces the data integration model and the meaning of consistent query answering under different semantics. Section 3 first introduces a unified (standard) solution to handle CQA via ASP which uniforms available approaches over different semantics, and then presents some optimizations. Section 4 describes the benchmark framework we adopted in the tests and presents obtained results. Finally, in Section 5 we draw some conclusions.

2 Data Integration Framework

In this paper, we use the following notation. We always denote, by Γ, a domain alphabet of values; by t, a tuple of values from Γ; by X, a variable; by $\bar{\mathbf{x}}$ a sequence $X_1, \ldots, X_n$

of variables, and by $|\bar{\mathbf{x}}| = n$ its length. Given $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$, we denote by $\bar{\mathbf{x}}, \bar{\mathbf{x}}'$ a possible sequence of length $|\bar{\mathbf{x}}| + |\bar{\mathbf{x}}'|$ obtained as the juxtaposition of all the variables in $\bar{\mathbf{x}}$ and $\bar{\mathbf{x}}'$. Also, we denote, by $\sigma(\bar{\mathbf{x}})$ a conjunction of comparison atoms of the form $X \odot X'$ with $\odot \in \{\leq, =, \neq\}$, and by $\ominus$, the symmetric difference operator between two sets. Finally, we adopt the *unique name assumption*.

A (relational) *schema* is a pair $R = \langle relSym(R), intCon(R) \rangle$ where $relSym(R)$ and $intCon(R)$ are the relational symbols and the integrity constraints (ICs) of R, respectively. The arity of a given relation $r \in relSym(R)$ is denoted by $arity(r)$. A *database* for R is any set of the form

$$\Delta = \{r(t) : r \in relSym(R) \ \wedge \ t \text{ is a tuple from } \Gamma \ \wedge \ |t| = arity(r)\}$$

Let $r_1, \ldots, r_m$ be relation symbols, the set $intCon(R)$ contains ICs of the form:

$$(c_1) \quad \forall \bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m \ \neg(r_1(\bar{\mathbf{x}}_1) \wedge \ldots \wedge r_m(\bar{\mathbf{x}}_m) \wedge \sigma(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m))$$
$$(c_2) \quad \forall \bar{\mathbf{x}}_1 \ \exists \bar{\mathbf{x}}_2 \ r_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \rightarrow r_2(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$$

Constraints of type c_1 (where $arity(r_i) = |\bar{\mathbf{x}}_i|$, for each i in $[1..m]$) are *denial constraints* (DCs) whereas those of type c_2 are *inclusion dependencies* (INDs). Note that, *functional dependencies* as well as *key dependencies* are special cases of denial constraints. A database Δ for R is said to be *consistent* w.r.t R iff all ICs are satisfied.

A *conjunctive query* $q(\bar{\mathbf{x}})$ over R is a formula of the form

$$\exists \bar{\mathbf{x}}_1^b, \ldots, \bar{\mathbf{x}}_m^b \ \ r_1(\bar{\mathbf{x}}_1^b, \bar{\mathbf{x}}_1^f) \wedge \ldots \wedge r_m(\bar{\mathbf{x}}_m^b, \bar{\mathbf{x}}_m^f) \wedge \sigma(\bar{\mathbf{x}}_1^b, \bar{\mathbf{x}}_1^f \ldots, \bar{\mathbf{x}}_m^b, \bar{\mathbf{x}}_m^f)$$

where $\bar{\mathbf{x}} = \bar{\mathbf{x}}_1^f, \ldots, \bar{\mathbf{x}}_m^f$ are its free variables. In particular, a query is *ground* if it does not contain any variable; it is *quantifier free* if all of its variables are free; it is *simple conjunctive* if it does not contain any repeated relation symbol. Given a database Δ for R, and a query $q(\bar{\mathbf{x}})$, the *answer* to q is the set of n-tuples of values

$$ans(q, \Delta) = \{t : \Delta \models q(t)\}.$$

2.1 The Data Integration Model

As usual [11], a *data integration system* is formalized as a triple $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ where

- $\mathcal{G}$ is the *global schema*. A *global database* for $\mathcal{I}$ is any database for $\mathcal{G}$;
- $\mathcal{S}$ is the *source schema*. A *source database* for $\mathcal{I}$ is any database consistent w.r.t $\mathcal{S}$;
- $\mathcal{M}$ is the *global-as-view* (GAV) *mapping*, that associates to each element g in $relSym(\mathcal{G})$ a (union of) conjunctive query over $\mathcal{S}$.

Let $\mathcal{D}$ be a source database for $\mathcal{I}$. The *retrieved global database* is

$$ret(\mathcal{I}, \mathcal{D}) = \{g(t) : g \in relSym(\mathcal{G}) \ \wedge \ t \in ans(q, \mathcal{D}) \ \wedge \ q \in \mathcal{M}(g)\}$$

for $\mathcal{G}$ satisfying the mapping. Notice that, when source data are combined in a unified schema with its own ICs, the retrieved global database might be inconsistent.

Remark 1. In our setting, w.l.o.g. [1], queries are Datalog rules and a database consists of a set of Datalog facts. □

Example 1. Consider a bank association that desires to unify the databases of two branches. The first (source) database models managers by using a table $man(code, name)$ and employees by a table $emp(code, name)$, where $code$ is a primary key for both tables. The second database stores the same data in table $employee(code, name, role)$. Suppose that the data has to be integrated under a global schema with two tables $m(code)$ and $e(code, name)$, where the global ICs are:

- $\forall x_1, x_2, x_3, x_4 \;\; \neg(e(x_1, x_2) \wedge e(x_3, x_4) \wedge x_1 = x_3 \wedge x_2 \neq x_4)$ i.e., $code$ is the primary key of e;
- $\forall x_1 \exists x_2 \; m(x_1) \rightarrow e(x_1, x_2)$ i.e., an IND imposing that each manager code must be an employee code as well.

The mapping is defined as follows:

$$\begin{array}{ll} e(X_c, X_n) :- \, emp(X_c, X_n). & m(X_c) :- \, man(X_c, \_). \\ e(X_c, X_n) :- \, employee(X_c, X_n, \_). & m(X_c) :- \, employee(X_c, \_, \text{'}manager\text{'}). \end{array}$$

Assume that, emp stores tuples *('e1','john')*, *('e2','mary')*, *('e3','willy')*, man stores *('e1','john')*, and $employee$ stores *('e1','ann','manager')*, *('e2','mary','manager')*, *('e3','rose','emp')*. It is easy to verify that, although the source databases are consistent w.r.t. local constraints, the global database, obtained by evaluating the mapping, violates the key constraint on e as both *john* and *ann* have the same code *e1* in table e. □

2.2 Consistent Query Answering under different semantics

In case that $ret(\mathcal{I}, \mathcal{D})$ violates ICs, one can still be interested in querying the "consistent" information originating from $\mathcal{D}$. One possibility is to "repair" $ret(\mathcal{I}, \mathcal{D})$ (by inserting or deleting tuples) in such a way that all the ICs are satisfied. But there are several ways to "repair" $ret(\mathcal{I}, \mathcal{D})$ e.g., to satisfy an IND of the form $r_1 \rightarrow r_2$ one might either remove violating tuples from r_1 or insert new tuples in r_2. Moreover, the repairing strategy depends on the particular semantic assumption made on the data integration system. Semantic assumptions may range from (strict) soundness to (strict) completeness. Roughly speaking, completeness complies with the *closed world assumption* where missing facts are assumed to be false; on the contrary, soundness complies with the *open world assumption* where $ret(\mathcal{I}, \mathcal{D})$ may be incomplete. We consider the following *semantics sound, complete, CM-complete, exact, loosely-sound, loosely-complete, loosely-exact* [2, 4, 6, 7]. More formally, let Σ denote a *semantics*, and $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ a data integration system. A global database $\mathcal{B}$ is said to be a Σ*-repair* for $ret(\mathcal{I}, \mathcal{D})$ if it is consistent w.r.t. $\mathcal{G}$ and one of the following conditions holds:

1. $\Sigma =$ *sound* and $\mathcal{B} \supseteq ret(\mathcal{I}, \mathcal{D})$;
2. $\Sigma =$ *complete* and $\mathcal{B} \subseteq ret(\mathcal{I}, \mathcal{D})$;
3. $\Sigma =$ *CM-complete*, $\mathcal{B} \subseteq ret(\mathcal{I}, \mathcal{D})$, and there is no other consistent global database $\mathcal{B}' \subseteq ret(\mathcal{I}, \mathcal{D})$ such that $\mathcal{B}' \supset \mathcal{B}$;
4. $\Sigma =$ *exact* and $\mathcal{B} = ret(\mathcal{I}, \mathcal{D})$;

5. Σ = *loosely-sound* and there is no other consistent global database $\mathcal{B}'$ such that $\mathcal{B}' \cap ret(\mathcal{I}, \mathcal{D}) \supset \mathcal{B} \cap ret(\mathcal{I}, \mathcal{D})$;
6. Σ = *loosely-complete* and there is no other consistent global database $\mathcal{B}'$ such that $\mathcal{B}' - ret(\mathcal{I}, \mathcal{D}) \subset \mathcal{B} - ret(\mathcal{I}, \mathcal{D})$;
7. Σ = *loosely-exact*, and there is no other consistent global database $\mathcal{B}'$ such that $\mathcal{B}' \ominus ret(\mathcal{I}, \mathcal{D}) \subset \mathcal{B} \ominus ret(\mathcal{I}, \mathcal{D})$.

Now, given a source database $\mathcal{D}$ for $\mathcal{I}$ and a semantics Σ, the *consistent answer* to a query q of arity n w.r.t. $\mathcal{D}$ and $\mathcal{I}$, is the set:

$$ans_{\Sigma}(q, \mathcal{I}, \mathcal{D}) = \{t : t \in ans(q, \mathcal{B}),\ \textit{for each } \Sigma\textit{-repair } \mathcal{B}\}$$

Consistent Query Answering (CQA) is the problem of computing $ans_{\Sigma}(q, \mathcal{I}, \mathcal{D})$.

Observe that *exact* semantics is trivial, since here CQA makes sense only if no global constraint is violated by the retrieved database. The *complete* semantics always allows the empty database as a repair and, thus, any query will return a void answer. CQA under *loosely-complete* semantics actually coincides with CQA under the complete one [5]. The *CM-complete* [7] semantics allows a minimal number of deletions in each repair to avoid empty repairs, if possible, but does not allow any insertion. The *sound* semantics, allowing insertions only, fails when some denial constraint is violated, whereas the *loosely-sound* one overcomes this problem by allowing a minimal amount of deletions. Finally, the *loosely-exact* semantics combines the loosely-sound and the loosely-complete ones; in fact it allows both insertions and deletions but minimizes the symmetric difference between the retrieved database and the repairs.

Example 2. By following Example 1, the retrieved global database admits exactly the following repairs under the *CM-complete* semantics:

$$\mathcal{B}_1^r = \{e(\text{‘e2’},\text{‘mary’}),\ e(\text{‘e1’},\text{‘john’}),\ e(\text{‘e3’},\text{‘willy’}),\ m(\text{‘e1’}),\ m(\text{‘e2’})\}$$
$$\mathcal{B}_2^r = \{e(\text{‘e2’},\text{‘mary’}),\ e(\text{‘e1’},\text{‘john’}),\ e(\text{‘e3’},\text{‘rose’}),\ m(\text{‘e1’}),\ m(\text{‘e2’})\}$$
$$\mathcal{B}_3^r = \{e(\text{‘e2’},\text{‘mary’}),\ e(\text{‘e1’},\text{‘ann’}),\ e(\text{‘e3’},\text{‘willy’}),\ m(\text{‘e1’}),\ m(\text{‘e2’})\}$$
$$\mathcal{B}_4^r = \{e(\text{‘e2’},\text{‘mary’}),\ e(\text{‘e1’},\text{‘ann’}),\ e(\text{‘e3’},\text{‘rose’}),\ m(\text{‘e1’}),\ m(\text{‘e2’})\}$$

Finally, the query $m(X)$, asking for the list of manager codes, has both *e1* and *e2* as consistent answers. □

2.3 Properties of CQA

We next recall some well known definitions as well as some important CQA properties.

Let r be a relation symbol with $n = arity(r)$, and $key(r)$ be a set of indexes from $I = \{1, \ldots, n\}$. A *key dependency* (KD) for r consists of a set of DCs, exactly one for each index k belonging to $I - key(r)$, of the form

$$\forall \bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2\ \neg(r(\bar{\mathbf{x}}_1) \wedge r(\bar{\mathbf{x}}_2) \wedge \sigma(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2) \wedge\ \bar{\mathbf{x}}_1^k \neq \bar{\mathbf{x}}_2^k)$$

where $|\bar{\mathbf{x}}_1| = |\bar{\mathbf{x}}_2| = n$, and $\sigma(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$ is a conjunction of comparison atoms of the form $\bar{\mathbf{x}}_1^i = \bar{\mathbf{x}}_2^i$, for each $i \in key(r)$. The set $key(r)$ is the *primary-key* of r.

Let d be an IND of the form $\forall \bar{\mathbf{x}}_1 \ \exists \bar{\mathbf{x}}_2 \ r_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \ \rightarrow \ r_2(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$. We denote by $univInd(d, r_i)$ the set of (projection) indexes from $\{1, \ldots, arity(r_i)\}$ induced by the positions of the variables $\bar{\mathbf{x}}_1$ in r_i, for each i in $[1..2]$. Moreover, d is said to be *non-key-conflicting* (NKC) [5] if and only if $univInd(d, r_2) \subseteq key(r_2)$.

In the following, we recall some known results about computability and complexity of CQA under different semantic assumptions (as usual in both ASP and Data Integration fields, we only refer to data complexity [16]).

Proposition 1. *[5] In general, CQA is undecidable if $\Sigma \in \{$sound, loosely-sound, loosely-exact$\}$.*

Proposition 2. *[5] CQA with KDs and NKC-INDs, is decidable. In detail it is coNP-complete under the sound and loosely-sound semantics and Π_2^p-complete under loosely-exact semantics.*

Proposition 3. *[7] CQA is decidable if $\Sigma =$ CM-complete. In detail, it is Π_2^p-complete, in general; and coNP-complete in case of acyclic INDs.*

In light of the above propositions and the considerations outlined in Section 2, we concentrate our analysis on the following semantics with the specified restrictions: (i) *CM-complete* with acyclic INDs; (ii) *loosely-sound* with only primary-key-dependencies as DCs, and NKC-INDs. In fact, these are the decidable cases having a complexity at most in coNP. Recall, moreover, that the sound semantics fails when some denial constraint is violated.

3 Computation of CQA via ASP

In this section, we show how to exploit *Answer Set Programming* (ASP) [9, 10] for efficiently computing consistent answers to user queries under different semantic assumptions. ASP is a powerful logic programming paradigm allowing (in its general form) for disjunction in rule heads [13] and nonmonotonic negation in rule bodies. In the following, we assume that the reader is familiar with ASP.

The suitability of ASP for implementing CQA has been already recognized in the literature [11, 2, 4, 7]. The general approaches are based on the following idea: produce an ASP program P having an answer set for each repair, so that the problem of computing CQA corresponds to cautious reasoning on P.

We next introduce two algorithms (for those cases having a complexity at most in coNP) that take as input a data integration system and a query, and produce an ASP program that can be exploited for computing CQA. After showing a general encoding, we propose an "optimized" method that is able to produce complexity-wise easier programs that are even optimal according to the complexity classification of constraints and queries in case of CM-complete semantics.

3.1 Standard Solution

Given a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a query q, we next give the general algorithm for building an ASP program, say Π_{cqa}, and a new query, say q_{cqa}, solving the

CQA problem via ASP. This program is created by “rewriting” each IC in $intCon(\mathcal{G})$. We report separately the rules created for each kind of IC, and we detail the creation of additional auxiliary rules.

Denial Constraints. Let $\Sigma \in \{$*CM-complete*, *loosely-sound*$\}$. For each DC of the form $\forall \bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m \;\; \neg(g_1(\bar{\mathbf{x}}_1) \wedge \ldots \wedge g_m(\bar{\mathbf{x}}_m) \wedge \sigma(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m))$ in $intCon(\mathcal{G})$, insert the following rule into Π_{cqa}:

$$g^c_1(\bar{\mathbf{x}}_1) \vee \cdots \vee g^c_m(\bar{\mathbf{x}}_m) \coloneq g_1(\bar{\mathbf{x}}_1), \cdots, g_m(\bar{\mathbf{x}}_m), \sigma(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m).$$

Inclusion dependencies. Let $\Sigma =$ *CM-complete*. For each IND in $intCon(\mathcal{G})$ of the form $\forall \bar{\mathbf{x}}_1 \, \exists \bar{\mathbf{x}}_2 \; g_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \rightarrow g_2(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$, with $\pi_2 = univInd(d, g_2)$, insert the following rules into Π_{cqa}:

$$g_2^{r\text{-}\pi_2}(\bar{\mathbf{x}}_1) \coloneq g^r_2(\bar{\mathbf{x}}_1, \_).$$
$$g^c_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \coloneq g_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3), \; not \;\; g_2^{r\text{-}\pi_2}(\bar{\mathbf{x}}_1).$$

Repaired Relations. Let $\Sigma \in \{$*CM-complete*, *loosely-sound*$\}$. For each relation symbol $g \in relSym(\mathcal{G})$, insert the following rule into Π_{cqa}:

$$g^r(\bar{\mathbf{x}}) \coloneq g(\bar{\mathbf{x}}), \; not \;\; g^c(\bar{\mathbf{x}}).$$

Cleaning Step. Remove useless body literals (i.e., negative literals whose complementary literals do not occur in any head) and possibly duplicated rules.

Query rewriting. Build q_{cqa} from q as follows by considering each atom of the form $g(\bar{\mathbf{x}})$ where $\bar{\mathbf{x}} = X_1, \ldots, X_n$:

1. Replace $g(\bar{\mathbf{x}})$ by $g^r(\bar{\mathbf{x}})$ only if there is an IC where g occurs.
2. If $\Sigma =$ *loosely-sound*, then apply the unfolding technique described in [6].

Theorem 1. *Let $\Sigma \in \{$CM-complete, loosely-sound$\}$. Given a retrieved global database $ret(\mathcal{I}, \mathcal{D})$ for a global schema $\mathcal{G}$ and a query q over $\mathcal{G}$, $t \in ans_{\Sigma}(q, \mathcal{I}, \mathcal{D})$ iff $q_{cqa}(t)$ is a cautious consequence of the ASP program $ret(\mathcal{I}, \mathcal{D}) \cup \Pi_{cqa}$.*

Proof (Sketch). If $\Sigma =$ *CM-complete*, then we claim that Π_{cqa} always allows to find only and all the repairs, exactly one per model. Let $\mathcal{B}^r$ be a repair. In the following, we describe how to obtain a model containing for each relation, say g, exactly only and all the tuples of g that do not appear in $\mathcal{B}^r$. We will collect such tuples in the new relation g^c, while we collect in g^r only and all the tuples of g appearing in $\mathcal{B}^r$. First of all, observe that, since we are considering only acyclic INDs, there must necessarily be some relation which is not involved in the left-hand side of any IND. Thus, for each relation, say g, exhibiting such a property:

1. By the disjunctive rules (if any) involving g (and containing in the right-hand side only relations of $\mathcal{G}$), we guess (exploiting the minimality of answer sets semantics) the minimal set of tuples of g, collected in g^c, that do not appear in $\mathcal{B}^r$.
2. Contrariwise, only and all the tuples of g that have to be in $\mathcal{B}^r$ are collected in the repaired relation g^r and computed by the rule $g^r(\bar{\mathbf{x}}) \coloneq g(\bar{\mathbf{x}}), \; not \;\; g^c(\bar{\mathbf{x}})$.

3. Next, for each IND of the form $\forall \bar{\mathbf{x}}_1 \; \exists \bar{\mathbf{x}}_2 \; g_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \rightarrow g(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$ (involving g in the right-hand side), we use the rule $g_1^c(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \coloneq g_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3),\; not \;\; g^{r\text{-}\pi_2}(\bar{\mathbf{x}}_1).$ for deciding which tuples of g_1 cannot (necessarily) appear in $\mathcal{B}^r$ ($g^{r\text{-}\pi_2}$ is a projection of g^r). Note that, these tuples are univocally identified after fixing g^r.
4. Finally, we can reiterate from step 1, by considering g_1 instead of g (the tuples of g_1^c may only be augmented by some disjunctive rule involving g_1).

It is clear that, by construction, Π_{cqa} has exactly one answer set per repair. Finally, the query is reorganized to exploit the repaired relations, and cautious reasoning does the rest.

Analogous considerations can be done when $\Sigma = $ *loosely-sound*. Still disjunctive rules guess a minimal set of tuples to be "canceled" whereas unfolding allows to deal with INDs (cfr. [6]). □

Remark 2. It is worth noting that our general encoding belongs to the head-cycle free class of ASP programs [3] for which cautious reasoning is coNP-complete. In fact, each predicate appearing in the body of any disjunctive rule is only defined by the mapping which, clearly, does not depend on any canceled predicates. □

In our ongoing example, the program (and the new query built from $q(X) \coloneq m(X).$) obtained by applying the above algorithm, under the *CM-complete* semantics, is:

$$
\begin{aligned}
& e^c(X_c, X_n) \vee e^c(X'_c, X'_n) \coloneq e(X_c, X_n),\; e(X'_c, X'_n),\; X_c = X'_c,\; X_n \neq X'_n. \\
& e^r(X_c, X_n) \coloneq e(X_c, X_n),\; not\; e^c(X_c, X_n). \\
& e^{r\text{-}\{1\}}(X_c) \coloneq e^r(X_c, \_). \\
& m^c(X_c) \coloneq m(X_c),\; not\;\; e^{r\text{-}\{1\}}(X_c). \\
& m^r(X_c) \coloneq m(X_c),\; not\;\; m^c(X_c). \\
& q_{cqa}(X_c) \coloneq m^r(X_c).
\end{aligned}
$$

When the obtained program is evaluated on the database we obtain four answer sets. It can be verified that, all the answer sets contain m^r(*'e1'*) and m^r(*'e2'*), (i.e., they are cautious consequences of Π_{cqa}) and, thus, *'e1'* and *'e2'* are the consistent answers to the original query.

3.2 Optimized Solution

The algorithm reported in the previous section is a general solution for solving the CQA problem, but, in several cases, more efficient ASP programs can be produced. First of all, note that the general algorithm blindly considers all the ICs on the global schema, including those that have no effect on the specific query. Consequently, redundant logic rules might be produced which slow down program evaluation. Note also that, there are a number of cases in which, according to [5, 7], the complexity of CQA stays in P; but disjunctive programs, for which cautious reasoning is a hard task [8], are generated in presence of denial constraints. This means that, the evaluation of the produced logic programs might be much more expensive than required in those "easy" cases. In more detail, depending on the types of both schema constraints and queries, CQA is tractable in the following cases:

- *Quantifier-free queries* and either:
  - denial constraints only, or
  - at most one key per relation if Σ = CM-complete;
- *Simple Conjunctive queries* and either:
  - at most one functional dependency per relation if Σ = CM-complete, or
  - at most one key per relation if Σ = CM-complete;
- *Conjunctive queries* and:
  - inclusion dependencies only;

In the following, we provide an optimized version of the standard algorithm that is capable of identifying tractable (sub-)cases for a generic input query and that produces ASP programs for CQA which are complexity-wise optimal.

Given a global schema $\mathcal{G}$ and a query q, the optimized algorithm analyzes both $intCon(\mathcal{G})$ and q, and: (i) singles out only the ICs affecting query results, (ii) employs positive non-disjunctive rules for dealing with DCs in known tractable cases, and (iii) exploits a strategy that replaces unfolding in case of INDs if Σ = loosely-sound. Specifically, consider the directed labeled graph $G_c = \langle relSym(\mathcal{G}), E\rangle$, called *constraint graph*, built in such a way that arc $(g, g', c) \in E$ if and only if one of the following holds:

- c is a DC in $intCon(\mathcal{G})$ involving both g and g'; or
- c is an IND in $intCon(\mathcal{G})$ of the form $g \to g'$ if Σ = CM-complete.
- c is an IND in $intCon(\mathcal{G})$ of the form $g' \to g$ if Σ = loosely-sound.

After analyzing and classifying the query (to recognize whether it is either quantifier-free, or simple conjunctive, or conjunctive), the constraint graph G_c is visited several times starting from each relation in the query. The visited nodes of G_c correspond to the relations involved in the query process, whereas the arcs traversed during the visits correspond to the constraints that might influence the query results. Thus, the corresponding relations and constraints are marked to be considered for further processing; unmarked constraints will be discarded. At the same time, the algorithm tags each marked constraint to be either easy or hard, depending on whether the above-reported conditions on the complexity of CQA are satisfied or not. In particular, the tag associated to a given constraint is set (or updated) during each visit depending on query kind, number and type of encountered constraints. The tag of each constraint c corresponding to a traversed arc e is set to “easy” if both (i) c was not previously tagged as “hard”, and (ii) at least one of the following conditions holds (otherwise c is tagged as “hard”):

1. if the query is quantifier-free, and either
   a. all the arcs belonging to the connected component of G_c containing e are labeled by denial constraints, or
   b. Σ = CM-complete and all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labeled by a key constraint
2. if the query is simple-conjunctive, Σ = CM-complete, and either
   a. all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labeled by a functional dependency constraint, or
   b. all the nodes belonging to the connected component of G_c containing e have at most one outgoing arc labeled by a key constraint

3. if the query is conjunctive, and
   a. all the arcs belonging to the connected component of G_c containing e are labeled by inclusion dependencies

At this point, the ASP program Π_{cqa} (and q_{cqa}, accordingly) can be modified as follows:

Step 1. For each DC in $intCon(\mathcal{G})$ marked "easy", insert the following m rules (instead of the ones containing disjunction in the standard solution) into Π_{cqa}:

$$g_i^c(\bar{\mathbf{x}}_i) :- g_1(\bar{\mathbf{x}}_1), \cdots, g_m(\bar{\mathbf{x}}_m), \sigma(\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_m). \quad \forall i \in [1..m]$$

Step 2. If $\Sigma = $ *loosely-sound*, for each marked IND of the form $\forall \bar{\mathbf{x}}_1 \; \exists \bar{\mathbf{x}}_2 \; g_1(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_3) \rightarrow g_2(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$, with $\pi_2 = univInd(d, g_2)$, insert the following rules into Π_{cqa}.
- $g_2^{\pi_2}(\bar{\mathbf{x}}_1) :- g_2^r(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2)$.
- $g_2^{\pi_2}(\bar{\mathbf{x}}_1) :- g^{\pi}(\bar{\mathbf{x}}_1, \bar{\mathbf{x}}')$. $\forall d'$ of the form $g \rightarrow g_2$ with $\pi_2 \subseteq univInd(d', g_2)$ and $\pi = univInd(d', g)$

Step 3. For the other constraints, add the corresponding rules of the standard solution only if they are marked.

Step 4. Build q_{cqa} from q as follows:
- If $\Sigma = $ *CM-complete*, then replace each $g(\bar{\mathbf{x}})$ by $g^r(\bar{\mathbf{x}})$ whenever g occurs in some marked constraint.
- If $\Sigma = $ *loosely-sound*, and there is an IND having g in its right hand side, and π is the set of indexes such that $i \in \pi$ iff X_i is a free-variable or a variable involved in some join w.r.t. q, and there is a g^{π} in Π_{cqa}, then replace $g(\bar{\mathbf{x}})$ by $g^{\pi}(\bar{\mathbf{x}}')$ where $\bar{\mathbf{x}}'$ is the subsequence of $\bar{\mathbf{x}}$ induced by π.

First of all, note that the new algorithm produces only non-redundant rules (i.e. the rules encoding constraints that influence the query answering process). Moreover, it is worth noticing that the rules produced by step 1, corresponding to "easy" constraints are non-disjunctive.[1] This is a pay-as-you-go technique where the usage of complex evaluation algorithms is limited to either intractable cases or to cases in which tractability results are not known. Rules introduced by steps 2 and 4, when Σ=loosely-sound, substitute the unfolding approach of [6] for handling INDs, and reduce, in general, the arity of involved predicates. Moreover, note that the same query may involve both easy and hard constraints, but disjunctive rules are used only for the hard ones.

For example, suppose that we add to the global schema of our ongoing example a new binary relation $c(code, name)$ representing the list of customers, and that $code$ is a key for c. Moreover, suppose that we ask for the query

$$q(X_c, X_n) :- c(X_c, X_n), e(X_c, X_n).$$

retrieving the customers that are also employees of the bank. In this case, the query is quantifier free, and only denial constraints are marked (under the CM-complete semantics) visiting the constraint graph. Indeed, it is easy to see that there is no way to reach

[1] In the "easy" cases the original database can be repaired by simply removing all the conflicting tuples. This can be done because each repair can be obtained from the original database by removing a single tuple among the ones that violate the same constraint. When rules of this kind are employed the answer sets do not correspond to repairs, but CQA still corresponds to cautious reasoning.

m in the constraint graph starting from the query atoms since the arc generated for the IND $\forall x_1 \exists x_2\ m(x_1) \rightarrow e(x_1, x_2)$ goes from m to e. This means that condition $1.a$ is verified, all marked constraints are "easy", and the produced program is:

$$
\begin{array}{l}
e^c(X_c, X_n) \text{ :- } e(X_c, X_n),\ e(X_c, X'_n),\ X_n \neq X'_n.\\
e^c(X_c, X'_n) \text{ :- } e(X_c, X_n),\ e(X_c, X'_n),\ X_n \neq X'_n.\\
c^c(X_c, X_n) \text{ :- } c(X_c, X_n),\ c(X_c, X'_n),\ X_n \neq X'_n.\\
c^c(X_c, X'_n) \text{ :- } c(X_c, X_n),\ c(X_c, X'_n),\ X_n \neq X'_n.\\
e^r(X_c, X_n) \text{ :- } e(X_c, X_n),\ not\ \ e^c(X_c, X_n).\\
c^r(X_c, X_n) \text{ :- } c(X_c, X_n),\ not\ \ c^c(X_c, X_n).\\
q_{cqa}(X_c, X_n) \text{ :- } c^r(X_c, X_n), e^r(X_c, X_n).
\end{array}
$$

where we have also simplified joins of the form $X = X'$. Note that the obtained program is non-disjunctive and stratified and it can be evaluated in polynomial time. In this case, the only answer set of the program contains the consistent answers to the original query.

4 Experiments

In this section we present some of the experiments we carried out to assess the effectiveness of our approach to consistent query answering. The datalog evaluation engine used in the experiment is the DLV^{DB} system [15, 14] coupled, via ODBC, with a MS SQLServer DBMS where input data were stored.

4.1 Data Set

We exploited the real-world data integration framework developed in the INFOMIX project (IST-2001-33570) [12] which integrates data from a real university context. In particular, considered data sources were available at the University of Rome "La Sapienza". These comprise information on students, professors, curricula and exams in various faculties of the university. This data is dispersed over several databases in various (autonomous) administration offices.

There are about 35 data sources in the application scenario, which are mapped into 14 global schema relations with about 20 GAV mappings and 29 integrity constraints. We call this data set **Infomix** in the following.

Besides the original source database instance (which takes about 16Mb on DBMS), we obtained bigger instances artificially. Specifically, we generated a number of copies of the original database; each copy is disjoint from the other ones but maintains the same data correlations between instances as the original database. This has been carried out by mapping each original attribute value to a new value having a copy-specific prefix.

Then, we considered two further datasets, namely **Infomix-x-10** and **Infomix-x-50** storing 10 copies (for a total amount of 160Mb of data) and 50 copies (800Mb) of the original database, respectively; clearly, in both cases one of the copies is the original database itself.

	Q_1	Q_2	Q_3	Q_4
Optimized Query Evaluation	co-NP	co-NP	P	P
Query Class	SC	UC	QF	SC
Involved Constraints	K+I	K+I	I	D+I
Query arity	3	2	3	0
N. of source tuples				
infomix	17266	3749	17725	37831
infomix-x-10	172660	37490	177250	378310
infomix-x-50	863300	1873950	886250	1891550

Table 1. Summary of tested queries.

4.2 Tested Queries

As previously pointed out, standard rewriting for CQA makes the time complexity of query evaluation to be in coNP in most cases; however, our optimizations allow in many relevant cases to simplify the rewriting in such a way that the complexity of the evaluation of the corresponding program can be in P.

In order to carry out a comprehensive performance analysis, we designed a set of queries spanning over the following perspectives:

- As for the computational complexity perspective we designed queries whose:
  - evaluation complexity with standard rewriting stays in coNP and evaluation complexity with optimized rewriting stays in P;
  - evaluation complexity with standard rewriting stays in coNP and evaluation complexity with optimized rewriting remains in coNP;
- As for the constraints perspective we designed queries involving:
  - Arbitrary Denial constraints only (D in the following)
  - Key constraints only (K in the following)
  - Inclusion dependencies only (I in the following)
  - Arbitrary Denial and Inclusion dependencies (D+I in the following)
  - Key constraints and Inclusion dependencies (K+I in the following)
- As for the query class perspective we designed:
  - Unrestricted Conjunctive queries (UC in the following)
  - Quantifier-free queries (QF in the following)
  - Simple Conjunctive queries (SC in the following)

We designed and ran several queries. Due to space constraints we concentrate here on some of them. Table 1 summarizes their characteristics. Here *Number of source tuples* indicates the number of tuples of all source relations involved by the query.

4.3 Compared Methods

In order to assess the characteristics of the proposed optimizations, we measured the execution time of each query with both the standard and the optimized rewriting. Specifically, we tested: *(i)* complete semantics with standard rewriting; *(ii)* complete semantics with optimized rewriting; *(iii)* loosely sound semantics with standard rewriting; *(iv)* loosely sound semantics with optimized rewriting. In Figure 1, the first group of four graphs compares *(i)* and *(ii)*; the second group of four graphs compares *(iii)* and *(iv)*, whereas the last group puts together *(ii)*, and *(iv)*.

4.4 Results and discussion

All tests have been carried out on an Intel Core 2 Duo T7300, 2.0 GHz, with 2 Gb Ram, running Windows 7 Operating System. We set a time limit of 30 minutes after which query execution has been killed. Results obtained for tested queries and methods (showing times in seconds) are illustrated in Figure 1. The bar for a method is absent in the graphs if query answering time was higher than the limit.

From the analysis of the figures and the characteristics of the queries reported in Table 1, we may draw the following observations: The optimized rewriting provides important performance improvements in both Q_1, Q_2, and Q_4. The only exception is for query Q_3 under the complete semantics, for which no optimization was possible and, consequently, standard and optimized rewritings coincide. Performance improvements of the optimized rewriting w.r.t. the standard one have been registered up to 80%[2] for query Q_4, 76% for Q_1, 60% for Q_3, and 47% for Q_2. The proposed optimizations always reduce execution times and do not introduce computational overhead.

As for the comparison among the semantics, we can observe that generally the complete semantics allows for faster response times. The only exception is on Query Q_3 which comprises inclusion dependencies only. In this case, the rewriting for the complete semantics must choose the tuples to be deleted, whereas the loosely sound semantics can just work on the original data set.

Finally, it is worth noticing that the scaling of the optimized algorithms over the three data sets is generally better than the standard ones.

Observe that it can be interesting to evaluate execution times of all rewritings with the addition of magic sets applied in cascade on them. Our intuition is that magic sets optimization and our optimizations are complementary and, consequently, their benefits may be summed up if the query involves some constant. Clearly, it would be important to evaluate the impact of the overhead introduced by this approach on the overall response time.

5 Conclusion and Ongoing Work

In this paper we presented an approach that allows to efficiently handle consistent query answering under different semantics and integrity constraints. The effectiveness of the approach is based on optimized algorithms capable of identifying both tractable queries and portions of the queries that may be treated efficiently. The approach is part of a complete system for data integration based on ASP. Its query evaluator engine allows to carry out queries directly on the databases where data reside, even in an ASP context. Results of our experimental activity demonstrate the effectiveness of the approach. As far as ongoing work, we are investigating for more optimizations that can be included in the algorithms to further improve query answering performances.

[2] This information has been computed over the unhalted queries only.

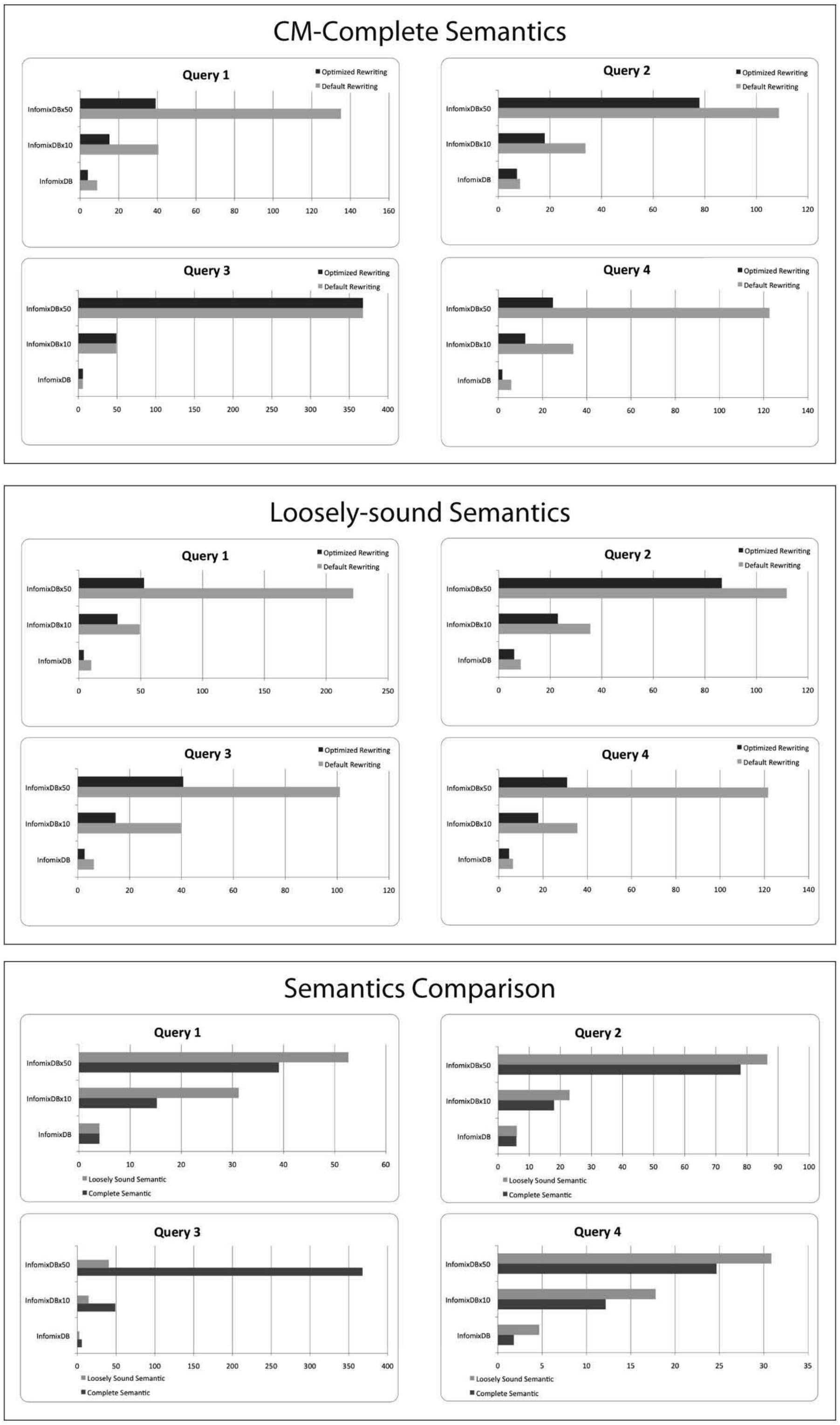

Fig. 1. Query Evaluation Execution Times.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
2. M. Arenas, L. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming*, 3(4):393–424, 2003.
3. R. Ben-Eliyahu and R. Dechter. Propositional Semantics for Disjunctive Logic Programs. *Annals of Mathematics and Artificial Intelligence*, 12(1-2):53–87, March 1994.
4. L. E. Bertossi, A. Hunter, and T. Schaub, editors. *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, Berlin / Heidelberg, January 2005. Springer.
5. A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS'03 – Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, June 09 - 11, 2003, San Diego, California*, pages 260–271, New York, NY, USA, 2003. ACM.
6. A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *IJCAI'03 – Proceedings of the 18th international joint conference on Artificial intelligence, August 09 - 15, 2003, Acapulco, Mexico*, pages 16–21, San Francisco, CA, USA, August 2003. Morgan Kaufmann Publishers Inc.
7. J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, February 2005.
8. T. Eiter, G. Gottlob, and H. Mannila. Disjunctive datalog. *ACM Transactions on Database Systems (TODS)*, 22(3):364–418, September 1997.
9. M. Gelfond and V. Lifschitz. The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen, editors, *ICLP/SLP'88 – Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
10. M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3-4):365–385, August 1991.
11. M. Lenzerini. Data integration: a theoretical perspective. In *PODS'02 – Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, June 03 - 05, 2002, Madison, Wisconsin*, pages 233–246, New York, NY, USA, 2002. ACM.
12. N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis. The INFOMIX system for advanced integration of incomplete and inconsistent data. In *SIGMOD'05 – Proceedings of the 2005 ACM SIGMOD international conference on Management of data, June 14 - 16, 2005, Baltimore, Maryland*, pages 915–917, New York, NY, USA, 2005. ACM.
13. J. Minker. On Indefinite Data Bases and the Closed World Assumption. In D. W. Loveland, editor, *CADE'82 – Proceedings 6th Conference on Automated Deduction, June 79, 1982, New York, USA*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308, Berlin / Heidelberg, June 1982. Springer.
14. G. Terracina, E. De Francesco, C. Panetta, and N. Leone. Enhancing a DLP System for Advanced Database Applications. In D. Calvanese and G. Lausen, editors, *RR 2008 – Proceedings of the second International Conference on Web Reasoning and Rule Systems , October 31 - November 1, 2008, Karlsruhe, Germany*, volume 5341 of *Lecture Notes in Computer Science*, pages 119–134, Berlin / Heidelberg, October 2008. Springer.
15. G. Terracina, N. Leone, V. Lio, and C. Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory and Practice of Logic Programming*, 8(2):129–165, March 2008.
16. M. Y. Vardi. The Complexity of Relational Query Languages (Ext. Abstract). In *STOC'82 – Proceedings of the 14th annual ACM symposium on Theory of computing, May 05 - 07, 1982, San Francisco, California, USA*, pages 137–146, New York, NY, USA, 1982. ACM.