Consistent Query Answering under Key and Exclusion Dependencies: Algorithms and Experiments

Luca Grieco Domenico Lembo Riccardo Rosati Marco Ruzzi

Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy
{grieco,lembo,rosati,ruzzi}@dis.uniroma1.it

ABSTRACT

Research in *consistent query answering* studies the definition and computation of "meaningful" answers to queries posed to *inconsistent databases*, i.e., databases whose data do not satisfy the integrity constraints (ICs) declared on their schema. Computing consistent answers to conjunctive queries is generally coNP-hard in data complexity, even in the presence of very restricted forms of ICs (single, unary keys). Recent studies on consistent query answering for database schemas containing only key dependencies have analyzed the possibility of identifying classes of queries whose consistent answers can be obtained by a first-order rewriting of the query, which in turn can be easily formulated in SQL and directly evaluated through any relational DBMS. In this paper we study consistent query answering in the presence of key dependencies and *exclusion dependencies*. We first prove that even in the presence of only exclusion dependencies the problem is coNP-hard in data complexity, and define a general method for consistent answering of conjunctive queries under key and exclusion dependencies, based on the rewriting of the query in Datalog with negation. Then, we identify a subclass of conjunctive queries that can be first-order rewritten in the presence of key and exclusion dependencies, and define an algorithm for computing the first-order rewriting of a query belonging to such a class of queries. Finally, we compare the relative efficiency of the two methods for processing queries in the subclass above mentioned. Experimental results, conducted on a real and large database of the computer science engineering degrees of the University of Rome "La Sapienza", clearly show the computational advantage of the first-order based technique.

Categories and Subject Descriptors

H.2.3 [Database Management]: Languages—query languages; F.2.0 [Analysis of Algorithms and Problem Complexity]: General.

CIKM'05, October 31–November 5, 2005, Bremen, Germany.

General Terms

Theory, Algorithms, Experimentation.

Keywords

Inconsistency, Computational Complexity, Query Rewriting.

1. INTRODUCTION

Suppose to have a database whose data violate the integrity constraints (ICs) declared on its schema. What are the answers that have to be returned to queries posed to such a database? The standard approach to this problem is through data cleaning, i.e., by explicitly modifying the data in order to eliminate violation of ICs: only when data are "repaired", i.e., are consistent with the ICs, queries can be answered. However, in many situations it would be much more desirable to derive significant information from the database even in the presence of data inconsistent with the ICs. Indeed, in many application scenarios, the explicit repair of data is not convenient, or even not possible. This happens, for instance, in *data integration* applications, which provide a unified, virtual view of a set of autonomous information sources [5].

This alternative approach is the one followed by research in *consistent query answering*, which studies the definition (and computation) of "meaningful" answers to queries posed to databases whose data do not satisfy the ICs declared on the database schema [1, 14, 4]. All these approaches are based on the following principle: *schema is stronger than data*. In other words, the database schema (i.e., the set of integrity constraints) is considered as the actually reliable information (strong knowledge), while data are considered as information to be revised (weak knowledge). Therefore, the problem amounts to deciding how to "repair" (i.e., change) data in order to reconcile them with the information expressed in the schema. Therefore, the intuitive semantics of consistent query answering can be expressed as follows: a tuple t is a consistent answer to a query q in an inconsistent database $\mathcal{D}$ if t is an answer to q in all the *repairs* of $\mathcal{D}$, i.e., in all the possible databases obtained by (minimally) modifying the data in $\mathcal{D}$ to eliminate violations of ICs.

EXAMPLE 1. *Let $D = \{r(a, b)\}$ be a database whose schema contains the declaration of a key dependency on the first attribute of r. Since the database instance does not violate the key dependency on r, the only repair of the database*

is D itself. Hence, the following query $q(X,Y) :- r(X,Y)$ has the consistent answer $t = \langle a, b\rangle$. Now, let D' be the database instance obtained by adding the fact $r(a,c)$ to D. D' is inconsistent with the key dependency, and has two possible repairs: $\{r(a,b)\}$ and $\{r(a,c)\}$. Since there is no tuple which is an answer to q in both repairs, it follows that there are no consistent answers to the query q in D'. In contrast, observe that the query $q'(X) :- r(X,Y)$ has the answer $\langle a\rangle$ both in D and in D', which can be therefore considered consistent. □

Recent studies in this area have established declarative semantic characterizations of consistent query answering over relational databases, decidability and complexity results for consistent query answering, as well as techniques for query processing [1, 6, 14, 4, 3, 5]. In particular, it has been shown that computing consistent answers of conjunctive queries (CQs) is coNP-hard in data complexity, i.e., in the size of the database instance, even in the presence of very restricted forms of ICs (single, unary keys).

¿From the algorithmic viewpoint, the approach mainly followed is query answering via query rewriting: (i) First, the query that must be processed (usually a conjunctive query) is reformulated in terms of another, more complex query. Such a reformulation is purely intensional, i.e., the rewritten query is independent of the database instance; (ii) Then, the reformulated query is evaluated over the database instance. Due to the semantic nature and the inherent complexity of consistent query answering, Answer Set Programming (ASP) is usually adopted in the above reformulation step [14, 3, 5], and stable model engines like DLV [15] can be used for query processing.

An orthogonal approach to consistent query answering is the one followed by recent theoretical works [1, 6, 13], whose aim is to identify *subclasses* of CQs whose consistent answers can be obtained by rewriting the query in terms of a *first-order* (FOL) query. The advantage of such an approach is twofold: first, this technique allows for computing consistent answers in time polynomial in data complexity (i.e., for such subclasses of queries, consistent query answering is computationally simpler than for the whole class of CQs); second, consistent query answering in these cases can be performed through standard database technology, since the FOL query synthesized can be easily translated into SQL and then evaluated by any relational DBMS. On the other hand, this approach is only limited to polynomial subclasses of the problem. In particular, Fuxman and Miller in [13] have studied databases with key dependencies, and have identified a broad subclass of CQs that can be treated according to the above strategy.

In this paper we study consistent query answering in the presence of key dependencies and *exclusion dependencies*, a well-known class of ICs. Notice that exclusion dependencies are not only typical of relational database schemas, but are also relevant and very common in languages for conceptual modeling, e.g., ontology languages [2]: indeed such dependencies allow for modeling partitioning/disjointness of entities. This makes the study of exclusion dependencies particularly important for the broad applicability of consistent query answering.

Our contribution can be summarized as follows:

1. We prove that consistent answering of conjunctive queries for databases with only exclusion dependencies is coNP-hard in data complexity, i.e., the problem presents the same complexity lower bound already known for databases with only key dependencies [6, 4].

2. We define a method for consistent query answering under key dependencies and exclusion dependencies based on the rewriting of the query in Datalog$^\neg$ [10], a well-known extension of Datalog that allows for using negation in the body of program rules. The rewriting extends the one defined in [5] to the presence of exclusion dependencies. The rewriting is used by INFOMIX,[1] a system for the integration of inconsistent data, based on the use of the DLV system.

3. We extend the work of [13] to the presence of exclusion dependencies in the database schema. In particular, we identify the class of $\mathcal{KE}$*-simple* queries (a subclass of CQs) that can be first-order rewritten in the presence of both key dependencies and exclusion dependencies, and define an algorithm for computing the first-order rewriting of a query belonging to such a class of queries. We point out that our algorithm, though inspired by the one of [13], in the presence of only key dependencies applies to a broader class of queries than the class considered first-order rewritable in [13]. Therefore, the technique of the present paper is relevant also for consistent query answering under only key dependencies.

4. We compare the relative efficiency of these two methods for processing $\mathcal{KE}$-simple queries. To this aim, we have realized a software module that implements the above two rewriting methods. Then, we have compared query answering based on the rewriting in Datalog$^\neg$ and evaluation in the DLV system [15] with the method based on first-order rewriting and query evaluation on MySQL DBMS. We have conducted our experiments on a real and large database of the computer science engineering degrees of the University of Rome "La Sapienza".

Our experimental results clearly show, for $\mathcal{KE}$-simple queries, the computational advantage of the specialized first-order based technique over the more general one based on Datalog$^\neg$. In particular, the results indicate that the advantage of the first-order based technique grows with the number of database tuples that violate the ICs. Such results thus provide, in a general sense, an experimental validation of the first-order based approach: its computational advantage is not only theoretical, but also can be effectively measured when applied to practical, realistic scenarios. However, it turns out that the general method based on Datalog$^\neg$, although not specifically tailored for $\mathcal{KE}$-simple queries, proves particularly efficient in the presence of few data inconsistencies.

In the next section, we briefly introduce the formal framework of consistent query answering. In Section 3, we prove coNP-hardness of consistent query answering under only exclusion dependencies, and present our Datalog$^\neg$ rewriting and our algorithm for first-order rewriting in the presence of key and exclusion dependencies. In Section 4, we present our experimental results, and in Section 5 we address related work and conclude the paper.

[1] `http://sv.mat.unical.it/infomix`.

2. INCONSISTENT DATABASES AND CONSISTENT ANSWERS

Syntax. A *database schema* $\mathcal{S}$ is a triple $\langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$, where:

- $\mathcal{A}$ is a relational signature.
- $\mathcal{K}$ is a set of *key dependencies* over $\mathcal{A}$. A key dependency (KD) over $\mathcal{A}$ is an expression of the form $key(r) = \{i_1, \ldots, i_k\}$, where r is a relation of $\mathcal{A}$, and, if n is the arity of r, $1 \leq i_j \leq n$ for each j such that $1 \leq j \leq k$. We assume that at most one KD is specified over a relation r.
- $\mathcal{E}$ is a set of *exclusion dependencies* over $\mathcal{A}$. An exclusion dependency (ED) over $\mathcal{A}$ is an expression of the form $r_1[i_1, \ldots, i_k] \cap r_2[j_1, \ldots, j_k] = \emptyset$, where r_1, r_2 are relations of $\mathcal{A}$, and, if n_1 and n_2 are the arities of r_1 and r_2 respectively, for each ℓ such that $1 \leq \ell \leq k$, $1 \leq i_\ell \leq n_1$ and $1 \leq j_\ell \leq n_2$.

A *term* is either a variable or a constant symbol. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where p is a relation symbol of arity n and $t_1, \ldots, t_n$ is a sequence of n terms (either variables or constants). An atom is called *fact* if all the terms occurring in it are constants. A *database instance* $\mathcal{D}$ for $\mathcal{S}$ is a set of facts over $\mathcal{A}$. We denote as $r^{\mathcal{D}}$ the set $\{t \mid r(t) \in \mathcal{D}\}$.

A *conjunctive query* of arity n is an expression of the form $h(x_1, \ldots, x_n) \coloneq a_1, \ldots, a_m$, where the atom $h(x_1, \ldots, x_n)$, is called the *head* of the query (denoted by $head(q)$), and $a_1, \ldots, a_m$, called the *body* of the query (and denoted by $body(q)$), is a set of atoms, such that all the variables occurring in the query head also occur in the query body. In a conjunctive query q, we say that a variable is a *head variable* if it occurs in the query head, while we say that a variable is *existential* if it only occurs in the query body. Moreover, we call an existential variable *shared* if it occurs at least twice in the query body (otherwise we say that it is *non-shared*). A *FOL query* of arity n is an expression of the form $\{x_1, \ldots, x_n \mid \Phi(x_1, \ldots, x_n)\}$, where $x_1, \ldots, x_n$ are variable symbols and Φ is a first-order formula with free variables $x_1, \ldots, x_n$.

Semantics. First, we briefly recall the standard evaluation of queries over a database instance. Let q be the CQ $h(x_1, \ldots, x_n) \coloneq a_1, \ldots, a_m$ and let $t = \langle c_1, \ldots, c_n \rangle$ be a tuple of constants. A set of facts I is an *image of* t *w.r.t.* q if there exists a substitution σ of the variables occurring in q such that $\sigma(head(q)) = h(t)$ and $\sigma(body(q)) = I$. Given a database instance $\mathcal{D}$, we denote by $q^{\mathcal{D}}$ the evaluation of q over $\mathcal{D}$, i.e., $q^{\mathcal{D}}$ is the set of tuples t such that there exists an image I of t w.r.t. q such that $I \subseteq \mathcal{D}$.

Given a FOL query q and a database instance $\mathcal{D}$, we denote by $q^{\mathcal{D}}$ the evaluation of q over $\mathcal{D}$, i.e., $q^{\mathcal{D}} = \{t_1, \ldots, t_n \mid \mathcal{D} \models \Phi(t_1, \ldots, t_n)\}$, where each t_i is a constant symbol and $\Phi(t_1, \ldots, t_n)$ is the first-order sentence obtained from Φ by replacing each free variable x_i with the constant t_i.

Then, we define the semantics of queries over inconsistent databases. A database instance $\mathcal{D}$ *violates* the KD $key(r) = \{i_1, \ldots, i_k\}$ iff there exist two *distinct* facts $r(c_1, \ldots, c_n)$, $r(d_1, \ldots, d_n)$ in $\mathcal{D}$ such that $c_{i_j} = d_{i_j}$ for each j such that $1 \leq j \leq k$. Moreover, $\mathcal{D}$ violates the ED $r_1[i_1, \ldots, i_k] \cap r_2[j_1, \ldots, j_k] = \emptyset$ iff there exist two facts $r_1(c_1, \ldots, c_n)$, $r_2(d_1, \ldots, d_m)$ in $\mathcal{D}$ such that $c_{i_\ell} = d_{j_\ell}$ for each ℓ such that $1 \leq \ell \leq k$.

Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema. A database instance $\mathcal{D}$ is *legal for* $\mathcal{S}$ if $\mathcal{D}$ does not violate any KD in $\mathcal{K}$ and does not violate any ED in $\mathcal{E}$.

A set of ground atoms $\mathcal{D}'$ is a *repair* of $\mathcal{D}$ under $\mathcal{S}$ iff: (i) $\mathcal{D}' \subseteq \mathcal{D}$; (ii) $\mathcal{D}'$ is legal for $\mathcal{S}$; (iii) for each $\mathcal{D}''$ such that $\mathcal{D}' \subset \mathcal{D}'' \subseteq \mathcal{D}$, $\mathcal{D}''$ is not legal for $\mathcal{S}$. In words, a repair for $\mathcal{D}$ under $\mathcal{S}$ is a maximal subset of $\mathcal{D}$ that is legal for $\mathcal{S}$.

Let q be a CQ. A tuple t is a *consistent answer* to q in $\mathcal{D}$ under $\mathcal{S}$ iff, for each repair $\mathcal{D}'$ of $\mathcal{D}$ under $\mathcal{S}$, $t \in q^{\mathcal{D}'}$.

EXAMPLE 2. *Consider the database schema* $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$, *where* $\mathcal{A}$ *comprises the relations* $Journal(title, editor)$, $ConfPr(title, editor)$ *and* $Editor(name, country)$, $\mathcal{K}$ *comprises the dependencies* $key(Journal) = \{1\}$, $key(ConfPr) = \{1\}$, $key(Editor) = \{1\}$, $\mathcal{E}$ *comprises the dependency* $Journal[1] \cap ConfPr[1] = \emptyset$. *Consider the database instance* $\mathcal{D}$ *described below*

$$\{Journal(\mathsf{TODS}, \mathsf{ACM}), Journal(\mathsf{TODS}, \mathsf{IEEE}), Editor(\mathsf{ACM}, \mathsf{USA}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}), ConfPr(\mathsf{PODS05}, \mathsf{SV}), Editor(\mathsf{IEEE}, \mathsf{USA})\}.$$

It is easy to see that $\mathcal{D}$ *is not consistent with the KDs on Journal and ConfPr of* $\mathcal{S}$. *Then, the repairs of* $\mathcal{D}$ *under* $\mathcal{S}$ *are:*

$$\{Journal(\mathsf{TODS}, \mathsf{ACM}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}), Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{ACM}), ConfPr(\mathsf{PODS05}, \mathsf{SV}), Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{IEEE}), ConfPr(\mathsf{PODS05}, \mathsf{ACM}), Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}$$

$$\{Journal(\mathsf{TODS}, \mathsf{IEEE}), ConfPr(\mathsf{PODS05}, \mathsf{SV}), Editor(\mathsf{ACM}, \mathsf{USA}), Editor(\mathsf{IEEE}, \mathsf{USA})\}.$$

Let $q(x, z) \coloneq Journal(x, y), Editor(y, z)$ *be a user query. The consistent answers to* q *in* $\mathcal{D}$ *under* $\mathcal{S}$ *are* $\{\langle \mathsf{TODS}, \mathsf{USA} \rangle\}$. □

3. QUERY ANSWERING

Computational Complexity. The problem of computing consistent answers to conjunctive queries over inconsistent databases in the presence of KDs (under the repair semantics introduced in Section 2) is coNP-hard in data complexity [4, 6]. In the following, we prove that such a problem is coNP-hard in data complexity also for schemas in which only EDs occur[2].

THEOREM 3. *Let* $\mathcal{S} = \langle \mathcal{A}, \emptyset, \mathcal{E} \rangle$ *be a database schema containing only EDs,* $\mathcal{D}$ *a database instance for* $\mathcal{S}$, q *a CQ of arity* n *over* $\mathcal{S}$, *and* t *an* n*-tuple of constants. The problem of establishing whether* t *is a consistent answer to* q *in* $\mathcal{D}$ *under* $\mathcal{S}$ *is coNP-hard with respect to data complexity.*

Proof (sketch). We prove coNP-hardness by reducing the 3-colorability problem to the complement of our problem. Consider a graph $G = \langle V, E \rangle$ with a set of vertices V and edges E. We define a relational schema $\mathcal{S} = \langle \mathcal{A}, \emptyset, \mathcal{E} \rangle$ where $\mathcal{A}$ consists of the relation *edge* of arity 2, and the relation *col* of arity 5, and $\mathcal{E}$ contains the dependencies $col[3] \cap col[4] = \emptyset$, $col[3] \cap col[5] = \emptyset$, $col[4] \cap col[5] = \emptyset$. The instance $\mathcal{D}$ is defined as follows:

$$\mathcal{D} = \{col(n, 1, n, \_, \_), col(n, 2, \_, n, \_), col(n, 3, \_, \_, n) \mid n \in V\} \cup \{edge(x, y) \mid \langle x, y \rangle \in E\}.$$

[2]We consider the decision problem associated to query answering (see e.g., [6])

Where each occurrence of the meta-symbol _ denotes a different constant not occurring elsewhere in the database. Intuitively, to represent the fact that vertex $n \in V$ is assigned with color $i \in \{1,2,3\}$, $\mathcal{D}$ assigns to *col* a tuple in which i occurs as second component and n occurs as first and also as $2+i$-th component. The EDs of $\mathcal{S}$ impose that consistent instances assign no more than one color to each node. Finally, we define the query

$$q \quad \leftarrow \quad edge(x,y), col(x,z,w_1,w_2,w_3), col(y,z,w_4,w_5,w_6).$$

On the basis of the above construction it is possible to show that G is 3-colorable (i.e., for each pair of adjacent vertices, the vertices are associated with different colors) if and only if the empty tuple $\langle\rangle$ is not a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ (i.e., the boolean query q has a negative answer). □

Datalog$^\neg$ Rewriting. We now provide a sound and complete query rewriting technique for consistent query answering in the presence of key and exclusion dependencies. To this aim, we make use of Datalog$^\neg$, i.e., Datalog enriched with (unstratified) negation, under stable model semantics [10]. From a computational point of view, Datalog$^\neg$ is coNP-complete with respect to data complexity, and therefore is well suited for dealing with the high computational complexity of our problem.

The rewriting that we present in the following extends the one proposed in [4] for CQs specified over database schemas with KDs, in order to properly handle the presence of EDs. The rewriting is employed in the system INFOMIX. Analogously to other proposals that solve consistent query answering via query rewriting (although for different classes of constraints and query languages, see, e.g., [14, 3]), the basic idea of the technique is to encode the constraints of the relational schema into a Datalog$^\neg$ program, such that the stable models of the program yield the repairs of the database instance $\mathcal{D}$.

DEFINITION 4. *Given a CQ[3] q and a schema $\mathcal{S}$, the* Datalog$^\neg$ *program $\Pi(q,\mathcal{S})$ is defined as the following set of rules[4]:*

1. *the rule corresponding to the definition of q;*
2. *for each relation $r \in \mathcal{S}$, the rules*

$$\begin{array}{rcl} r(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \ , \ not\ \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \\ \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \ , \ r(\tilde{\mathbf{x}},\tilde{\mathbf{z}}) \ , \ y_1 \neq z_1 \\ & \cdots & \\ \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \ , \ r(\tilde{\mathbf{x}},\tilde{\mathbf{z}}) \ , \ y_m \neq z_m \end{array}$$

   where: in $r(\tilde{\mathbf{x}},\tilde{\mathbf{y}})$ the variables in $\tilde{\mathbf{x}}$ correspond to the attributes constituting the key of the relation r; $\tilde{\mathbf{y}} = y_1,\ldots,y_m$ and $\tilde{\mathbf{z}} = z_1,\ldots,z_m$.
3. *for each exclusion dependency $(r[i_1,\ldots,i_k] \cap s[j_1,\ldots,j_k]) = \emptyset$ in $\mathcal{E}$, with $r \neq s$, the rules:*

$$\begin{array}{rcl} \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \ , \ s(\tilde{\mathbf{x}},\tilde{\mathbf{z}}) \\ \overline{s}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) & :- & s_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}}) \ , \ r(\tilde{\mathbf{x}},\tilde{\mathbf{z}}) \end{array}$$

   where $\tilde{\mathbf{x}} = x_1,\ldots,x_k$, i.e., the variables in $\tilde{\mathbf{x}}$ correspond to the sequence of attributes of r and s involved in the ED.
4. *for each exclusion dependency $r[\ell_1,\ldots,\ell_h] \cap r[m_1,\ldots,m_h] = \emptyset$ in $\mathcal{E}$, the rules:*

$$\begin{array}{rcl} \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \ , \ r(\tilde{\mathbf{y}},\tilde{\mathbf{w}}_1,\tilde{\mathbf{w}}_2) \ , \\ \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{y}},\tilde{\mathbf{z}}) \ , \ r(\tilde{\mathbf{w}}_1,\tilde{\mathbf{x}},\tilde{\mathbf{w}}_2) \ , \\ \overline{r}(\tilde{\mathbf{x}},\tilde{\mathbf{x}},\tilde{\mathbf{z}}) & :- & r_{\mathcal{D}}(\tilde{\mathbf{x}},\tilde{\mathbf{x}},\tilde{\mathbf{z}}). \end{array}$$

Furthermore, we denote with $\tau(\mathcal{D})$ the database instance obtained from $\mathcal{D}$ by replacing each predicate symbol r with $r_{\mathcal{D}}$.

Informally, for each relation r, $\Pi(q,\mathcal{S})$ contains (i) a relation $r_{\mathcal{D}}$ that represents $r^{\mathcal{D}}$; (ii) a relation r that represents a subset of $r^{\mathcal{D}}$ that is consistent with the KD for r and the EDs that involve r; (iii) an auxiliary relation $\overline{r}$ that represents the "complement" of r, i.e., the subset of $r^{\mathcal{D}}$ that together with r results inconsistent with the EDs and KDs on the schema. Notice that the extension of r depends on the choice made for $\overline{r}$ (and vice-versa), and that such choices are made in a non-deterministic way (enforced by the use of the unstratified negation). The above rules force each stable model M of $\Pi(q,\mathcal{S}) \cup \tau(D)$ to be such that r^M is a maximal subset of tuples from $r^{\mathcal{D}}$ that are consistent with both the KD for r and the EDs in $\mathcal{E}$ that involve r.

Example 2.(contd.) The Datalog$^\neg$ rewriting $\Pi(q,\mathcal{S})$ of the query $q(x,z) \text{ :- } Journal(x,y), Editor(y,z)$ is the following program:

$$\begin{array}{rcl} q(x,z) & :- & Journal(x,y), Editor(y,z) \\ Journal(x,y) & :- & Journal_{\mathcal{D}}(x,y) \ , \ not\ \overline{Journal}(x,y) \\ Editor(x,y) & :- & Editor_{\mathcal{D}}(x,y) \ , \ not\ \overline{Editor}(x,y) \\ ConfPr(x,y) & :- & ConfPr_{\mathcal{D}}(x,y) \ , \ not\ \overline{ConfPr}(x,y) \\ \overline{Journal}(x,y) & :- & Journal_{\mathcal{D}}(x,y) \ , \ Journal(x,z) \ , \ z \neq y \\ \overline{Editor}(x,y) & :- & Editor_{\mathcal{D}}(x,y) \ , \ Editor(x,z) \ , \ z \neq y \\ \overline{ConfPr}(x,y) & :- & ConfPr_{\mathcal{D}}(x,y) \ , \ ConfPr(x,z) \ , \ z \neq y \\ \overline{Journal}(x,y) & :- & Journal_{\mathcal{D}}(x,y) \ , \ ConfPr(x,z) \\ \overline{ConfPr}(x,y) & :- & ConfPr_{\mathcal{D}}(x,y) \ , \ Journal(x,z) \end{array}$$

The first rule of the rewriting encodes the query. The second, third and fourth rule establish the relationship between each relation and the corresponding complementary predicate. The fifth, sixth, and seventh rule encode the KDs of $\mathcal{S}$, whereas the last two rules encode the ED. □

We now state correctness of our encoding with respect to the semantics of consistent query answering.

THEOREM 5. *let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema, $\mathcal{D}$ be a database instance for $\mathcal{S}$, and q be a CQ over $\mathcal{S}$. A tuple t is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ iff $t \in q^M$ for each stable model M of $\Pi(q,\mathcal{S}) \cup \tau(\mathcal{D})$.*

From the above theorem and Theorem 3 it follows that the consistent query answering problem under KDs and EDs is coNP-complete in data complexity.

FOL Rewriting. Let us now consider a different approach to consistent query answering, which aims at identifying *subclasses* of queries for which the problem is tractable. This is the line followed in [1, 6, 13]. In particular, in [13] the authors define a subclass of CQs, called $\mathcal{C}_{tree}$, for which

[3]The present rewriting is not actually restricted to CQs, since it can be immediately extended to general Datalog$^\neg$ queries.

[4]Without loss of generality, we assume that the attributes in the key precede all other attributes in r, that $i_1 = j_1 = 1,\ldots,i_k = j_k = k$, $\ell_1 = 1,\ldots,\ell_h = h$, and $m_1 = h+1,\ldots,m_h = h+h$.

they prove tractability of consistent query answering in the presence of KDs, and provide a FOL rewriting technique. The class $\mathcal{C}_{tree}$ is based on the notion of join graph: a *join graph of a query* q is the graph that contains (i) a node N_i for every atom in the query body, (ii) an arc from N_i to N_j iff an existential shared variable occurs in a non-key position in N_i and occurs also in N_j, (iii) an arc from N_i to N_i iff an existential shared variable occurs at least twice in N_i, and one occurrence is in a non-key position. According to [13], $\mathcal{C}_{tree}$ is the class of conjunctive queries (a) without repeated relation symbols, (b) in which every join condition involves the entire key of at least one relation and (c) whose join graph is acyclic. As pointed out in [13], this class of queries is very common, since cycles are rarely present in queries used in practice. However, no repeated symbols may occur in the queries, and queries must have joins from non-key attributes of a relation to the entire key of another one.

We now extend the work of [13] as follows:

- We refine the class $\mathcal{C}_{tree}$ by allowing join conditions in which not necessarily the entire key of one relation has to be involved, but it is sufficient that, for each pair of attributes, at least one attribute must belong to a key (i.e., we allow for joins involving portions of key). In such a way, we obtain a new class, called $\mathcal{C}^{+}_{tree}$, larger than $\mathcal{C}_{tree}$, for which consistent query answering is polynomial in the presence of KDs. In other words, $\mathcal{C}^{+}_{tree}$ is the class of conjunctive queries for which only condition (a) and (c) above hold.
- We refine the class $\mathcal{C}^{+}_{tree}$ in order to obtain a class of queries, called $\mathcal{KE}$*-simple*, for which consistent query answering is polynomial in the presence of both KDs and also EDs.
- We provide a new algorithm for computing the FOL rewriting for $\mathcal{KE}$*-simple* queries. In the algorithm, we exploit the notion of join graph of [13], but we enrich the structure of the graph by associating to each node an adornment which specifies the different nature of terms in the atoms (see below), in order to deal with $\mathcal{KE}$*-simple* queries.

Let us describe in detail our technique. Henceforth, given a CQ q, we denote by R_q the set of relation symbols occurring in *body*(q). Given a database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ and a CQ q, we denote by $O_{\mathcal{E}}(q)$ the set of relation symbols $O_{\mathcal{E}}(q) = \{s \mid r[j_1, \ldots, j_k] \cap s[\ell_1, \ldots, \ell_k] = \emptyset \in \mathcal{E} \text{ and } r \in R_q\}$. In words, $O_{\mathcal{E}}(q)$ contains each relation symbol $s \in \mathcal{A}$ such that there exists an exclusion dependency between s and r in $\mathcal{E}$, where r is a relation symbol occurring in *body*(q).

DEFINITION 6. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema. A conjunctive query q is* $\mathcal{KE}$-simple *if $q \in \mathcal{C}^{+}_{tree}$, and*

- *there exists no pair of relation symbols r, s in $O_{\mathcal{E}}(q)$ such that there exists an exclusion dependency between r and s in $\mathcal{E}$,*
- *there exists no relation symbol r in $O_{\mathcal{E}}(q)$ such that there exists $r[i_1, \ldots, i_k] \cap s[j_1, \ldots, j_k] = \emptyset$ in $\mathcal{E}$, and either key$(r) \not\supseteq \{i_1, \ldots, i_k\}$ or key$(s) \not\supseteq \{j_1, \ldots, j_k\}$, where s is a relation symbol in R_q.*

In words, a query q is $\mathcal{KE}$-simple if it belongs to the class $\mathcal{C}^{+}_{tree}$, and if both there are no EDs between relations that are in $O_{\mathcal{E}}(q)$, and each ED between a relation $r \in R_q$ and a relation $s \in O_{\mathcal{E}}(q)$ does not involve non-key attributes of r or s. Notice that this last condition does not limit the applicability of our approach in many practical cases. For example, in relational databases obtained from ER-schemas, EDs are typically specified between keys.

For $\mathcal{KE}$-simple CQs, we present in the following a query rewriting algorithm which, given a query q, produces a FOL rewriting, whose evaluation over any database instance $\mathcal{D}$ for the database schema $\mathcal{S}$ returns the consistent answers to q in $\mathcal{D}$ under $\mathcal{S}$. The basic idea of the algorithm is to specify a set of conditions, expressible in FOL, that, if verified over a database instance $\mathcal{D}$, for a given tuple t, guarantee that in any repair of $\mathcal{D}$ there is an image of t w.r.t q, i.e., t is a consistent answer to q in $\mathcal{D}$. We point out that, for non-$\mathcal{KE}$-simple CQs, such conditions cannot be specified in FOL. Observe that, in our approach, the FOL rewriting is then in turn translated into SQL, and query evaluation is performed by means of standard DBMS query answering techniques. This further encoding does not present particular difficulties, and due to space limit we omit such transformation.

In order to construct our join graph we need the following definition.

DEFINITION 7. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema, q be a CQ, and $a = r(x_1, \ldots, x_n)$ be an atom (of arity n) occurring in R_q. Then, let key$(r) = \{i_1, \ldots, i_k\}$ belong to $\mathcal{K}$, and let $1 \leq i \leq n$. The* type of the i-th argument of a in q, *denoted by type(a, i, q) is defined as follows:*

1. *If $i_1 \leq i \leq i_k$, then:*
   - *if x_i is a head variable of q, a constant, or an existential shared variable, then type$(a, i, q) = KB$;*
   - *if x_i is an existential non-shared variable of q, then type$(a, i, q) = KU$.*
2. *Otherwise ($i \notin \{i_1, \ldots, i_k\}$):*
   - *if x_i is a head variable of q or a constant, then type$(a, i, q) = B$;*
   - *if x_i is an existential shared variable of q, then type$(a, i, q) = S$;*
   - *if x_i is an existential non-shared variable of q, then type$(a, i, q) = U$.*

Terms typed by KB or B are called *bound terms*, otherwise they are called *unbound*. We call the *typing of a in q* the expression of the form $r(x_1/t_1, \ldots, x_n/t_n)$, where each t_i is the type of the argument x_i in q.

The following algorithm KEFolRewrite computes the FOL rewriting to a $\mathcal{KE}$-simple conjunctive query q. In the algorithm, $JG(q)$ denotes the join graph of q, in which each node N_i is labelled with the typing of the corresponding atom a_i in q. Furthermore, *roots*$(JG(q))$ denotes the set of nodes that are roots in $JG(q)$ (notice that for $\mathcal{KE}$-simple queries the join graph is a forest, since it is acyclic).

Algorithm KEFolRewrite$(q, \mathcal{S})$
Input: $\mathcal{KE}$-simple CQ q (whose head variables are $x_1, \ldots, x_n$); schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$
Output: FOL query (representing the rewriting of q)
begin

Algorithm FolTree(N,$\mathcal{E}$)
Input: node N of $JG(q)$; set of EDs $\mathcal{E}$
Output: FOL formula
begin
  let $a = r(x_1/t_1, \ldots, x_n/t_n)$ be the label of N;
  for $i := 1$ **to** n **do**
    if $t_i \in \{KB, B\}$ **then** $v_i := x_i$
    else $v_i := y_i$, where y_i is a new variable
  if each argument of a is of type B or KB **then** $f_1 := r(x_1, \ldots, x_n)$
  else begin
    let $i_1, \ldots, i_m$ be the positions of the arguments of a of type S, U, KU;
    $f_1 := \exists y_{i_1}, \ldots, y_{i_m}.\, r(v_1, \ldots, v_n)$
  end;
  for each ED $r[j_1, \ldots, j_k] \cap s[\ell_1, \ldots, \ell_k] = \emptyset \in \mathcal{E}$ **do**
  begin
    let m be the arity of s;
    for $i := 1$ **to** m **do**
      if $i \in \{\ell_1, \ldots, \ell_k\}$ **then if** $i = \ell_c$ **then** $z_i = v_{j_c}$
      else $z_i = y'_i$ where y'_i is a new variable;
    let $y'_{i_1}, \ldots, y'_{i_k}$ be the new variables above introduced;
    $f_1 = f_1 \wedge \neg \exists y'_{i_1}, \ldots, y'_{i_k}\, .\, s(z_1, \ldots, z_m)$
  end
  if there exists no argument in a of type B or S **then return** f_1
  else begin
    let $p_1, \ldots, p_c$ be the positions of the arguments of a of type U, S or B;
    let $\ell_1, \ldots, \ell_h$ be the positions of the arguments of a of type B;
    for $i := 1$ **to** c **do**
      if $t_{p_i} = S$ **then** $z_{p_i} := x_{p_i}$ **else** $z_{p_i} := y''_i$, where y''_i is a new variable
    for $i := 1$ **to** n **do**
      if $t_i \in \{KB, KU\}$ **then** $w_i := v_i$ **else** $w_i := z_i$;
    $f_2 := \forall z_{p_1}, \ldots, z_{p_c}.\, r(w_1, \ldots, w_n) \rightarrow \left(\bigwedge_{N' \in jgsucc(N)} \mathsf{FolTree}(N') \right) \wedge \bigwedge_{i \in \{\ell_1, \ldots, \ell_h\}} w_i = x_i$
    return $f_1 \wedge f_2$
  end
end

Figure 1: The algorithm FolTree

  compute $JG(q)$;
  return $\{x_1, \ldots, x_n \mid \bigwedge_{N \in roots(JG(q))} \mathsf{FolTree}(N, \mathcal{E})\}$
end

Basically, the algorithm builds the join graph of q and then builds the first-order query by invoking the algorithm FolTree on all the nodes that are roots of the join graph.

The algorithm FolTree is defined in Figure 1. Roughly speaking, the algorithm $\mathsf{FolTree}(N, \mathcal{E})$ returns a first-order formula that constitutes the encoding of the whole subtree of the join graph of the query whose root is the node N. To do that, the algorithm computes two subformulas f_1 and f_2. The formula f_1 contains an atom whose predicate is the predicate r labelling the node N, in which the unbound variables of r are renamed with new existentially quantified variables. Furthermore, f_1 contains an atom of the form $\neg \exists y'_{i_1}, \ldots, y'_{i_k}\, .\, s(z_1, \ldots, z_m)$ for each ED that involves r and a relation s. Intuitively, when evaluated over a database instance $\mathcal{D}$, each such atom checks that there are no facts of the form $s(t_s) \in \mathcal{D}$ that violate the ED together with a fact of the form $r(t_r) \in \mathcal{D}$, which is in an image I of a tuple t w.r.t. the input query q, i.e., the atom guarantees that I is not contradicted w.r.t. the ED. The formula f_2 is empty only when all non-key arguments of the atom r are existential non-shared variables (i.e., of type U). Otherwise, the formula f_2 is a universally quantified implication. In such an implication, the antecedent is an atom whose predicate is r, and the consequent is a conjunction of equality conditions and other subformulas: more precisely, there is an equality condition for each non-key argument in r of type B, and a subformula for each successor N' of N in the join graph of q, computed by recursively invoking FolTree on N'. Intuitively, f_2 enforces the joins between r and each atom labelling the successors of r in the join graph of q. At the same time f_2 ensures that, when evaluated over a database instance $\mathcal{D}$, if there exists a fact of the form $r(\overline{t_r}) \in \mathcal{D}$ that violates the KD specified on r together with a fact of the form $r(t_r) \in \mathcal{D}$, which is in the image of a tuple t w.r.t. q, $r(\overline{t_r})$ belongs to another image of t w.r.t. q. In other words, the atom guarantees that in any repair there exists an image of t (w.r.t. the KD on r). Such a check is iterated for other KDs by recursively invoking FolTree. The following example illustrates the way the algorithm works.

Example 2.(contd.) It is easy to verify that the query $q(x, z) :- Journal(x, y), Editor(y, z)$ is $\mathcal{KE}$-simple.

$$Journal(x/KB, y/S)\,(N1) \longrightarrow (N2)\, Editor(y/KB, z/B)$$

Now, by applying the algorithm KEFolRewrite and FolTree we obtain:

$$\begin{aligned}
\mathsf{KEFolRewrite}(q) &= \{x, z \mid \mathsf{FolTree}(N1)\} \\
\mathsf{FolTree}(N1) &= \exists y_2.\, Journal(x, y_2) \wedge \neg \exists y'_2.\, ConfPr(x, y'_2) \wedge \\
&\quad \forall y.\, Journal(x, y) \rightarrow (\mathsf{FolTree}(N2)) \\
\mathsf{FolTree}(N2) &= Editor(y, z) \wedge \forall y''_2.\, Editor(y, y''_2) \rightarrow y''_2 = z.
\end{aligned}$$

relations		integrity constraints	
$faculty/3$	$exam\_plan/10$	$key(faculty) = \{1,2\}$	$key(plan\_status) = \{1\}$
$course\_assignment/3$	$degree/5$	$key(exam\_plan) = \{1\}$	$key(positioning) = \{1\}$
$positioning/2$	$course/4$	$key(university) = \{1\}$	$key(prof\_data) = \{1\}$
$plan\_status/2$		$key(exam\_type) = \{1\}$	$key(degree) = \{1\}$
$prof\_data/3$		$key(course) = \{1\}$	$key(exam) = \{2\}$
$university/3$		$key(master\_exam) = \{1\}$	
$bachelor\_exam/2$		$key(bachelor\_exam) = \{1\}$	
$master\_exam/2$		$course\_assignment[2] \cap professor[1] = \emptyset$	
$exam\_type/2$		$master\_exam[1] \cap bachelor\_exam[1] = \emptyset$	
$exam/4$		$course[3,4] \cap bachelor\_exam[1,2] = \emptyset$	

Figure 2: A portion of the test database schema

By evaluating the rewriting over $\mathcal{D}$ we get $\{\langle \mathsf{TODS}, \mathsf{USA} \rangle\}$, i.e., the set of consistent answers to q in $\mathcal{D}$ under $\mathcal{S}$. □

Next, we state soundness and completeness of the algorithm.

THEOREM 8. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K}, \mathcal{E} \rangle$ be a database schema, q be a $\mathcal{KE}$-simple conjunctive query over $\mathcal{S}$, and q_r be the FOL rewriting returned by* KEFolRewrite(q)*. Then, for every database instance $\mathcal{D}$ for $\mathcal{S}$, a tuple t is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ iff $t \in q_r^{\mathcal{D}}$.*

As a corollary, consistent query answering for $\mathcal{KE}$-simple conjunctive queries over database schemas with KDs and EDs is polynomial in data complexity.

4. EXPERIMENTS

We now present some experimental results comparing the FOL and the Datalog$^\neg$ rewriting previously described. To perform the experiments, we implemented a rewriting module that translates CQs issued over the database schema into both FOL queries and Datalog$^\neg$ queries. FOL queries are in turn translated by the module into SQL queries. Then, we ran the SQL queries on a MySQL 4.1.10 instance of the test database, while we executed Datalog$^\neg$ queries on DLV [15]. The experiments were conducted on a double processor machine, with 3 GHz Pentium IV Xeon CPU and 2 GB of main memory, running the Linux operating system.

The test database holds information about the computer science engineering degrees of the university of Rome "La Sapienza" and contains 27 tables with an overall size of over 200.000 tuples. In Figure 2, we present the portion of the test database schema that is relevant for the queries (in the figure, "r/n" indicates that relation r is of arity n).
Due to space limits, we only report details about three of the queries we tested:

$$
\begin{aligned}
Q_0 &= q(C) : -faculty(C, U, 'INGEGNERIA').\\
Q_2 &= q(S, D, P) : -positioning(PS, P), plan\_status(ST, DE),\\
&\quad exam\_plan(C, S, PS, DT, ST, '1', U1, U2, U3, U4).\\
Q_3 &= q(N, D, NP, CP) : -master\_exam(C, N, T, '5'),\\
&\quad exam\_type(T, D),
\end{aligned}
$$

The queries have been posed on various instances of the test database with an increasing number of pairs of tuples violating some ICs. Figure 3, shows experimental results. In the charts 3(a), 3(b) and 3(c), the execution time of the SQL encoding and of the Datalog$^\neg$ program are compared for queries Q_0, Q_2, and Q_3. As expected, from a certain inconsistency level on, the execution time of the Datalog$^\neg$ encoding has an exponential blow-up; in contrast, the execution time for the SQL encoding is constant on the average, and for Q_3 (Figure 3(b)) it decreases: although this might be surprising, it turns out that some inconsistency allows the SQL engine to prune the search space for query answering. Moreover, the chart presented in Figure 3(d) compares, on a logarithmic scale, the execution time of all queries at the highest inconsistency level. It shows that the SQL encoding is always more efficient when the degree of data inconsistency grows; however, it turns out that the method based on Datalog$^\neg$ and DLV proves particularly efficient in the presence of few data inconsistencies.

5. CONCLUSIONS

The present work provides a general experimental validation of the first-order rewriting approach to the optimization of consistent query answering. Of course, the applicability of our technique is limited to the class of $\mathcal{KE}$-simple queries. For general CQs, the use of a more expressive, and computationally harder, query language like Datalog$^\neg$ is necessary.

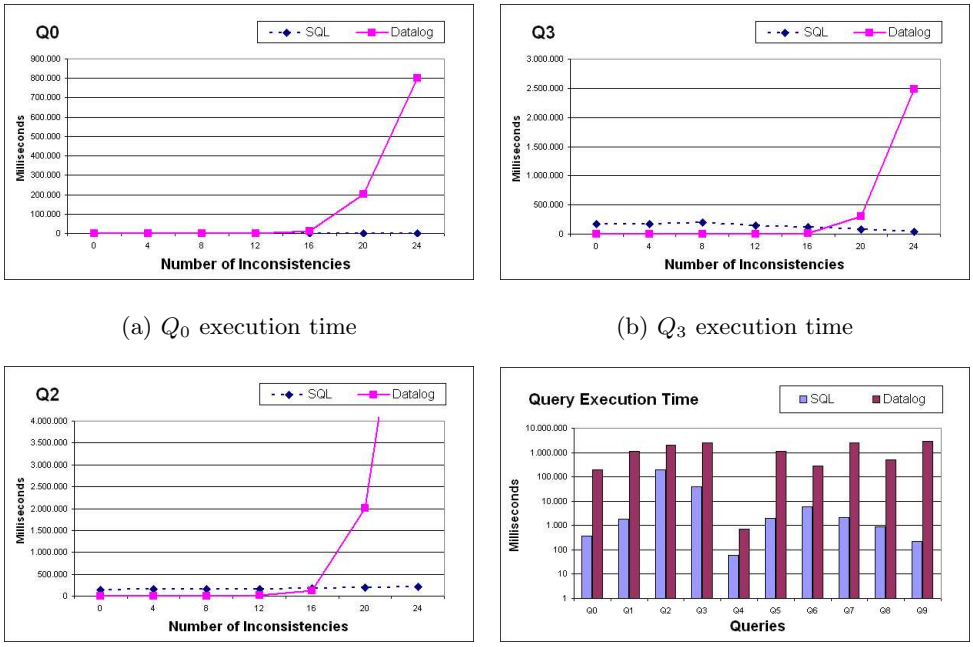
Very recently, the first prototype implementations of consistent query answering have appeared, and the first efforts towards optimization of query processing are emerging.

Within INFOMIX, several optimizations are currently under development to improve consistent query answering for more expressive classes of queries [9, 8]. In this respect, binding propagation techniques based on magic sets might significantly reduce execution time for Datalog$^\neg$ programs on DLV [11], even if the coNP structure of the Datalog$^\neg$ encoding suggests that the efficiency of the SQL rewriting can be hardly reached (especially for a large number of inconsistencies).

The ConQuer system [12] implements an extension of the technique of [13] which allows to rewrite in SQL queries belonging to the class $\mathcal{C}_{tree}$ enriched with aggregates. Experiments show that the overhead of evaluating rewritten queries is not onerous if compared with evaluation of the original query over the inconsistent database. Therefore, [12] focuses on comparing standard query answering and consistent query answering, while our experiments compare two different query answering techniques. In this respect, we point out that optimization of our SQL rewriting was outside the scope of the present paper.

Finally, Hippo [7] is a system for consistent answering of union of conjunctive queries without existential variables in the presence of denial constraints. Hence, this approach is different from our in terms of both query language and integrity constraints allowed. Moreover, Hippo techniques are not based on rewritings.

As future work, we aim at extending our approach to other forms of ICs (e.g., foreign keys) and at optimizing the SQL rewriting produced by KEFolRewrite.



(a) Q_0 execution time

(b) Q_3 execution time

(c) Q_2 execution time

(d) SQL vs. Datalog¬

Figure 3: Experimental Results

6. ACKNOWLEDGMENTS

This research has been partially supported by the Project INFOMIX (IST-2001-33570) funded by the EU.

7. REFERENCES

[1] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS'99*, pages 68–79, 1999.

[2] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[3] Loreto Bravo and Leopoldo Bertossi. Logic programming for consistently querying data integration systems. In *Proc. of IJCAI 2003*, pages 10–15, 2003.

[4] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS 2003*, pages 260–271, 2003.

[5] Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, pages 16–21, 2003.

[6] Jan Chomicki and Jerzy Marcinkowski. On the computational complexity of minimal-change integrity maintenance in relational databases. In *Inconsistency Tolerance*, pages 119–150, 2005.

[7] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *Proc. of CIKM 2004*, pages 417–426, 2004.

[8] Chiara Cumbo, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Enhancing the magic-set method for disjunctive datalog programs. In *Proc. ICLP 2004)*, pages 371–385, 2004.

[9] Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Efficient evaluation of logic programs for querying data integration systems. In *Proc. of ICLP'03*, pages 163–177, 2003.

[10] Thomas Eiter, Georg Gottlob, and Heikki Mannilla. Disjunctive Datalog. *ACM Trans. on Database Systems*, 22(3):364–418, 1997.

[11] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets and their application to data integration. In *Proc. of ICDT 2005*, pages 306–320, 2005.

[12] Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: Efficient management of inconsistent databases. In *Proc. of SIGMOD 2005*, pages 155–166, 2005.

[13] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *Proc. of ICDT 2005*, pages 337–351, 2005.

[14] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.

[15] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. on Computational Logic*, 2005. To appear.