On the First-Order Reducibility of Unions of Conjunctive Queries over Inconsistent Databases

Domenico Lembo, Riccardo Rosati, and Marco Ruzzi

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

Abstract. Recent approaches in the research on inconsistent databases have started analyzing the *first-order reducibility* of consistent query answering, i.e., the possibility of identifying classes of queries whose consistent answers can be obtained by a first-order (FOL) rewriting of the query, which in turn can be easily formulated in SQL and directly evaluated through any relational DBMS. So far, the investigations in this direction have only concerned subsets of conjunctive queries over databases with key dependencies. In this paper we extend the study of first-order reducibility of consistent query answering under key dependencies to more expressive queries, in particular to unions of conjunctive queries. More specifically: (i) we analyze the applicability of known FOL-rewriting techniques for conjunctive queries in the case of unions of conjunctive queries. It turns out that such techniques are applicable only to a very restricted class of unions of conjunctive queries; (ii) to overcome the above limitations, we define a new rewriting method which is specifically tailored for unions of conjunctive queries. The method can be applied only to unions of conjunctive queries that satisfy an acyclicity condition on unions of conjunctive queries.

1 Introduction

Consistent query answering. Research in *consistent query answering* (CQA) studies the definition (and computation) of "meaningful" answers to queries posed to databases whose data do not satisfy the integrity constraints (ICs) declared on the database schema [2, 11, 4].

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 [2, 6, 11, 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 [11, 3, 5], and stable model engines like DLV [13] can be used for query processing.

First-order reducibility of Consistent Query Answering. An orthogonal approach to consistent query answering is the one followed by recent theoretical works [2, 6, 10, 12], whose aim is to identify classes of *first-order reducible* queries, i.e., queries 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 efficiently in data complexity. More specifically, in this case, the data complexity of consistent query answering is the one of standard evaluation of FOL queries over databases, i.e., LogSpace. 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 database management system. On the other hand, this approach is only limited to particular subclasses of the problem. In particular, Fuxman and Miller in [10] have studied databases with key dependencies, and have identified a broad subclass of CQs that can be treated according to the above strategy.

Our contribution. In this paper we study first-order reducibility of consistent query answering for unions of conjunctive queries in the presence of key dependencies. More specifically, our contribution can be summarized as follows:

1. first, we analyze the direct applicability of the rewriting technique of [10] to unions of conjunctive queries. In particular, we characterize the subclass of unions of conjunctive queries for which a first-order rewriting can be computed in a modular way, such that the FOL rewriting of a union of conjunctive queries corresponds to the union of the FOL rewritings of each single conjunctive query. Each query belonging to such a subclass is a union of conjunctive queries in which (i) every disjunct can be rewritten exploiting the rewriting technique presented in [10], and (ii) repetition of atom symbols in different disjuncts is limited according to a suitable condition (see Section 3). It turns out that this way of FOL-reducing unions of conjunctive queries is possible only for a very restricted class of unions of conjunctive queries;
2. to overcome the limitations of the previous approach, we define a new rewriting method which is specifically tailored for unions of conjunctive queries. The method can be applied only to a subclass of unions of conjunctive queries, in particular the queries that satisfy an *acyclicity* condition on unions of conjunctive queries: for each such query q, the method produces a FOL-rewriting of the query whose evalutation produces the consistent answers to q. Note that this new defined subclass, properly contains the one described in the previous item.

Relevance of our results. We believe that the relevance of our study is twofold:

1. Extending the study of first-order reducibility of consistent query answering from conjunctive (i.e., select-project-join) queries to more expressive queries is certainly interesting: in this respect, the extension to unions of conjunctive queries is particularly important, since the possibility of expressing unions is probably the most important expressive feature which is missed by the language of conjunctive queries.
2. As explained in Section 5, we argue that the ability of handling unions of conjunctive queries is necessary in order to extend the first-order reduction techniques of consistent query answering to other forms of integrity constraints, specifically to *inclusion dependencies*. Besides key dependencies, inclusion dependencies, and in particular *foreign keys*, are certainly the most important form of integrity constraints in relational schemas. At the best of our knowledge this problem has not been studied yet. Notably, the analysis of first-order reduction of consistent query answering of unions of conjunctive queries constitutes a necessary first step in order to arrive at the definition of analogous methods for (unions of) conjunctive queries under key and foreign key dependencies.

Structure of the paper. In the next section, we present some preliminary definitions. In Section 3 we recall the method for first-order reducibility of conjunctive queries under key dependencies and study under which conditions this technique can be directly applied to unions of conjunctive queries. Then, in Section 4 we define a new query rewriting algorithm specifically designed for unions of conjunctive queries, and discuss formal properties of the method. Finally, we conclude in Section 5.

2 Inconsistent databases and consistent answers

Syntax. We consider to have an infinite, fixed alphabet Γ of constants representing real world objects, and we take into account only database instances having Γ as domain. Moreover, we assume that different constants in Γ denote different objects, i.e., we adopt the so-called *unique name assumption*.

A *database schema* $\mathcal{S}$ is constituted by a *relational signature* $\mathcal{A}$, i.e., a set of relation symbols in which each relation is associated with an arity (positive integer) indicating the number of its attributes, and a set of integrity constraints specified over $\mathcal{A}$. An *attribute* of a relation symbol r is an integer b such that $1 \leq b \leq n$, where n is the arity of r. We consider schemas which contain only *key dependencies* specified 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 symbol 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 and we say that an attribute of r is a *key attribute* if it belongs to the set $key(r)$ (otherwise we say that it is a *non-key attribute*). We denote with the pair $\langle \mathcal{A}, \mathcal{K} \rangle$, a database schema $\mathcal{S}$ with signature $\mathcal{A}$ and set of key dependencies $\mathcal{K}$ over $\mathcal{A}$.

A *term* is either a variable or a constant of Γ. 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. 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 $\{\boldsymbol{t} \mid r(\boldsymbol{t}) \in \mathcal{D}\}$.

A *union of conjunctive queries* (UCQ) q of arity n over a (database schema with) signature $\mathcal{A}$ is an expression of the form

$$h(x_1, \ldots, x_n) :\!-\ d_1 \vee \ldots \vee d_m$$

where the atom $h(x_1, \ldots, x_n)$ is called the *head* of the query (denoted by *head*(q)), $d_1 \vee \ldots \vee d_m$ is called the *body* of the query (denoted by *body*(q)), and for each $i \in \{1 \ldots m\}$, d_i, called the *i-th disjunct* of q, is a conjunction of atoms $a_{i,1} \wedge \ldots \wedge a_{i,k}$, whose predicate symbols are in $\mathcal{A}$, such that all the variables occurring in the query head also occur in d_i. If $m = 1$, q is simply called *conjunctive query* (CQ). In a UCQ 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 in a disjunct d of q* if it occurs at least twice in d (otherwise we say that it is *non-shared in d*). Obviously, if q is a CQ, an existential variable shared (resp. non-shared) in the unique disjunct of q will be simply called shared (resp. non-shared) in q.

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 UCQ $h(x_1, \ldots, x_n) :\!-\ d_1 \ldots\ldots\ldots d_m$ and let $\boldsymbol{t} = \langle c_1, \ldots, c_n \rangle$ be a tuple of constants of Γ. A set of facts I is an *image of $\boldsymbol{t}$ w.r.t. q* if there exists a substitution σ of the variables occurring in a disjunct d_i of q such that $\sigma(\mathit{head}(q)) = h(\boldsymbol{t})$ and $\sigma(d_i) = I$. 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 $\boldsymbol{t}$ such that there exists an image I of $\boldsymbol{t}$ 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., $q^{\mathcal{D}} = \{\langle c_1, \ldots, c_n \rangle \mid \mathcal{D} \models \Phi(c_1, \ldots, c_n)\}$, where each t_i is a constant symbol and $\Phi(c_1, \ldots, c_n)$ is the first-order sentence obtained from Φ by replacing each free variable x_i with the constant c_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 $1 \leq j \leq k$.

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

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 $\mathcal{S}$.

The problem in which we are interested is *consistent query answering* [2, 6]: given a database schema $\mathcal{S}$, a database instance $\mathcal{D}$, and a UCQ q, return all tuples $\boldsymbol{t}$ of constants of Γ such that, for each repair $\mathcal{D}'$ of $\mathcal{D}$ under $\mathcal{S}$, $\boldsymbol{t} \in q^{\mathcal{D}'}$. Each such tuple is called consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$.

Furthermore, analogously to [10], we say that consistent query answering for a class $\mathcal{C}$ of UCQs is *FOL-reducible* (or simply that the class $\mathcal{C}$ is FOL-reducible), if for every database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$ and every query $q \in \mathcal{C}$ over $\mathcal{A}$, there exists a FOL query q_f over $\mathcal{A}$ such that for every database instance $\mathcal{D}$, $\boldsymbol{t}$ is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ iff $\boldsymbol{t} \in q_f^{\mathcal{D}}$. We call such a q_f a *FOL-rewriting* of q under $\mathcal{S}$. Notice that FOL-reducibility is a very interesting property from a practical point of view, since FOL queries correspond to queries expressed in relational algebra (i.e., in SQL). Observe also that every FOL query can be evaluated in LogSpace wrt data complexity, i.e., computational complexity w.r.t. the size of the database instance (see e.g., [1]). It follows that if a class $\mathcal{C}$ is FOL-reducible, then consistent query answering for $\mathcal{C}$ is in LogSpace wrt data complexity.

3 FOL-rewriting of UCQs via FOL-rewriting of CQs

It is well known that the consistent query answering problem studied in this paper is coNP-hard in data complexity for generic conjunctive queries (and thus for generic unions of conjunctive queries) [4, 6]. As a consequence, the issue of scalability of query answering with respect to (large) database instances turns out to be crucial [3, 8]. In this respect, an interesting approach is the one that aims at identifying *subclasses* of queries for which the problem is tractable [7, 6], or FOL-reducible [2, 10, 9]. In particular, in [10] the authors study the problem for the class of conjunctive queries, and define a subclass of CQs, called $\mathcal{C}_{tree}$, for which they provide an algorithm for FOL-rewriting under schemas which contain only KDs. The class $\mathcal{C}_{tree}$ is based on the notion of join graph: a *join graph of a conjunctive 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 if 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 if an existential shared variable occurs at least twice in N_i, and at least one occurrence is in a non-key position. According to [10], $\mathcal{C}_{tree}$ is the class of conjunctive queries (a) without repeated relation symbols, (b) in which every join from non-key to key attributes involves the entire key of at least one relation and (c) whose join graph is acyclic. As pointed out in [10], this class of queries is very common, since cycles are rarely present in queries used in practice.

A class of CQs slightly more general than $\mathcal{C}_{tree}$, called $\mathcal{C}^{+}_{tree}$, has been considered in [12], and a new algorithm, called CQ-FolRewrite, for FOL-rewriting of such CQs has been proposed. Conjunctive queries belonging to such a class respect condition (a) and (c) above, but admit also joins from non-key attributes that not necessarily involve the entire key of a relation (i.e., condition (b) above has been removed)[1].

In what follows we consider the algorithm CQ-FolRewrite and study a possible extension of it in order to deal with queries specified in the more expressive language of unions of conjunctive queries. In particular, we consider UCQs where each disjunct is of class $\mathcal{C}^{+}_{tree}$. Notice that, even if CQA of CQs in the class $\mathcal{C}^{+}_{tree}$ is FOL-reducible, CQA for queries that are unions of $\mathcal{C}^{+}_{tree}$ queries is not in general FOL-reducible as shown by the following theorem.

Theorem 1. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$ be a database schema, $\mathcal{D}$ a database instance for $\mathcal{S}$, q a UCQ of arity n over $\mathcal{S}$, and t an n-tuple of constants in Γ. 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) The proof is by reduction of the three-colorability problem to the complement of our problem. $\square$

For the sake of completeness, we show the algorithm CQ-FolRewrite and its subroutine NodeRewrite in Figure 1 and Figure 2, respectively.

```
Algorithm CQ-FolRewrite($q, \mathcal{S}$)
Input: CQ $q \in \mathcal{C}^{+}_{tree}$ with $q = h(x_1, \ldots, x_n) \text{ :- } d$
        schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$
Output: FOL query
begin
    compute $JG(q)$;
    return $\{x_1, \ldots, x_n \mid \bigwedge_{N \in roots(JG(q))} \text{NodeRewrite}(JG(q), N, \mathcal{S})\}$
end
```

Fig. 1. The algorithm CQ-FolRewrite

In the algorithm, we exploit a refined notion of join graph, in which we associate to each node an adornment which specifies the different nature of terms in the atoms, as formally specified below.

Definition 1. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$ be a database schema, q be a CQ over $\mathcal{A}$, and $a = r(x_1, \ldots, x_n)$ be an atom (of arity n) occurring in the body of 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:

[1] The algorithm CQ-FolRewrite takes into account also other forms of integrity constraints specified on the database schema (a.k.a. exclusion dependencies), which are not considered in the present paper.

```
**Algorithm** NodeRewrite($JG(q), N, \mathcal{S}$)
**Input:** Join Graph $JG(q)$;
        node $N$ of $JG(q)$
        schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$
**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**;
  **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** $w_i := z_i$;
    $f_2 := \forall z_{p_1}, \ldots, z_{p_c}. r(w_1, \ldots, w_n) \rightarrow \left( \bigwedge_{N' \in jgsucc(N)} \text{NodeRewrite}(JG(q), N', \mathcal{S}) \right) \wedge \bigwedge_{i \in \{\ell_1, \ldots, \ell_h\}} w_i = x_i$
    **return** $f_1 \wedge f_2$
  **end**
**end**
```

Fig. 2. The algorithm NodeRewrite

- *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.

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, and $jgsucc(N)$

denotes the set of node which are successors on N in the join graph. Furthermore, $roots(JG(q))$ denotes the set of nodes that are roots in $JG(q)$ (notice that each join graph for a query of class $\mathcal{C}^{+}_{tree}$ is actually a set of trees, i.e. a forest). For a detailed description of the algorithm we refer the reader to [12], where also soundness and completeness of CQ-FolRewrite with respect to the problem of consistent query answering for CQs belonging to the $\mathcal{C}^{+}_{tree}$ class are established.

We are now ready to attack the study of consistent query answering for UCQs specified over database schemas with key dependencies. We start by analyzing the possibility of solving the problem for a UCQ q by simply applying the algorithm CQ-FolRewrite to each disjunct d_i of q, and taking as result the query q_f obtained by the union of the FOL queries produced by each such execution of CQ-FolRewrite. In order to do that, in the following we obviously consider UCQs whose disjuncts are of class $\mathcal{C}^{+}_{tree}$. Formally, we provide the algorithm UCQ-FolRewrite shown in Figure 3.

```
**Algorithm** UCQ-FolRewrite($q, \mathcal{S}$)
**Input:** UCQ $q = h(x_1, \ldots, x_n)$ :- $d_1 \vee \ldots \vee d_m$ such that $d_i \in \mathcal{C}^{+}_{tree}$ for $i \in \{1, \ldots, m\}$;
        schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$
**Output:** FOL query
**begin**
  **for** $i := 1$ **to** $m$ **do**
  **begin**
    $q_i = h(x_1, \ldots, x_n)$ :- $d_i$;
    compute $JG(q_i)$;
  **end**
  **return** $\{x_1, \ldots, x_n \mid \bigvee_{i=1}^{m} \bigwedge_{N \in roots(JG(q_i))}$ NodeRewrite$(JG(q_i), N, \mathcal{S})\}$;
**end**
```

Fig. 3. The algorithm UCQ-FolRewrite

Example 1. Consider a database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$, such that $\mathcal{A}$ contains the binary relation symbols r_1, r_2 and r_3, and $\mathcal{K}$ contains the dependencies $key(r_1) = \{1\}$, $key(r_2) = \{1\}$, $key(r_3) = \{1\}$. Consider the UCQ

$$q \text{ :- } (r_1(x, y) \wedge r_2(y, z)) \vee (r_3(x, y) \wedge r_2(y, z))$$

over $\mathcal{A}$. The join graphs of each disjunct are as follows:

$$\begin{array}{c}
r_1(x/KU, y/S)\ (N1) \longrightarrow (N2)\ r_2(y/KB, z/U) \\
r_3(x/KU, y/S)\ (N1) \longrightarrow (N2)\ r_2(y/KB, z/U)
\end{array}$$

Now it is easy to see that any disjunct in the query is in class $\mathcal{C}^{+}_{tree}$. Then, the first-order query returned by the execution of UCQ-FolRewrite$(q, \mathcal{S})$ is

$$\begin{aligned} q_f = \{ \ \mid & (\exists x, y.r_1(x, y) \wedge \forall y'.r_1(x, y') \rightarrow \exists z.r_2(y', z)) \vee \\ & (\exists x, y.r_3(x, y) \wedge \forall y'.r_3(x, y') \rightarrow \exists z.r_2(y', z))\} \end{aligned}$$

For such an example it is possible to verify that the query above is actually the FOL-rewriting of the input query q, i.e., for every database instance $\mathcal{D}$, $\boldsymbol{t}$ is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$ iff $\boldsymbol{t} \in q_f^{\mathcal{D}}$. □

Now, the question arises whether the condition that any disjunct in the input query q is in class $\mathcal{C}^+_{tree}$ is sufficient in order to guarantee soundness and, in particular, completeness of the algorithm. The following example shows that actually this is not the case.

Example 2. Assume to have a database schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$, such that $\mathcal{A}$ contains the relation symbol r of arity 2, and $\mathcal{K}$ contains the dependency $key(r) = \{1\}$. Consider the UCQ q :- $r(x, c_1) \vee r(x', c_2)$ over $\mathcal{A}$, in which c_1 and c_2 are different constant symbols. It is immediate to verify that any disjunct in the query is in class $\mathcal{C}^+_{tree}$. Then, the first-order query returned by the execution of $\mathsf{UCQ\text{-}FolRewrite}(q, \mathcal{S})$ is

$$\begin{aligned} q_f = \{ \ \mid & (\exists x, y.r(x, y) \wedge \forall y'.r(x, y') \rightarrow y' = c_1) \vee \\ & (\exists x, y.r(x, y) \wedge (\forall y'.r(x, y') \rightarrow y' = c_2) \ \}. \end{aligned}$$

Now, assume to have the database instance $\mathcal{D} = \{r(a, c_1), r(a, c_2)\}$, which is not legal for $\mathcal{S}$. It is easy to see that $\mathcal{D} \not\models \Phi$, where Φ is the sentence corresponding to the body of q_f, i.e., according to a notation commonly adopted in the database theory for boolean queries, $\langle\rangle \notin \mathcal{D}$, where $\langle\rangle$ indicates the empty tuple. On the other hand, the repairs of $\mathcal{D}$ under $\mathcal{S}$ are $\mathcal{R}_1 = \{r(a, c_1)\}$ and $\mathcal{R}_2 = \{r(a, c_2)\}$, and the body of the query q evaluates to true in both $\mathcal{R}_1$ and $\mathcal{R}_2$, i.e. $\langle\rangle$ is a consistent answer to q in $\mathcal{D}$ under $\mathcal{S}$. □

The example above shows that the algorithm $\mathsf{UCQ\text{-}FolRewrite}$ is in general incomplete (even if it is easy to see that it is always sound). This is mainly due to the fact that separately rewriting single disjuncts does not take into account the interaction that may exist between them. Indeed, the body of the FOL-rewriting that the algorithm constructs for each single disjunct d_i (i.e., $\bigwedge_{N \in roots(JG(q_i))} \mathsf{NodeRewrite}(JG(q_i), N)$) is a FOL formula, which we denote with ϕ, such that, given an assignment of the free variables of ϕ (i.e., a tuple of constants $\boldsymbol{t}$), the sentence $\phi(\boldsymbol{t})$ is satisfied only by those database instances $\mathcal{D}$ such that in any repair of $\mathcal{D}$ there is an image of $\boldsymbol{t}$ w.r.t the disjunct d_i. On the other hand, for a union of conjunctive queries q, for a tuple $\boldsymbol{t}$ to be a consistent answer to q it is sufficient that in any repair of $\mathcal{D}$ there exists an image of the tuple w.r.t. q, i.e. with respect to *any* disjunct d_j of q (in other words, the disjunct which provides the image has not to be the same in any repair). This is actually the case we have in Example 2.

Despite the above limitations of the algorithm, we are able to identify a subclass of conjunctive queries for which the algorithm UCQ-FolRewrite is sound and complete. To this aim, we provide the following definition.

Definition 2. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$ be a database schema, let r be a relation symbol of $\mathcal{A}$ such that $key(r) = \{1, \ldots, n\} \in \mathcal{K}$. Let q be a UCQ over $\mathcal{A}$ and let $a_1 = r(\boldsymbol{x}, \boldsymbol{y})$ and $a_2 = r(\boldsymbol{z}, \boldsymbol{w})$ be two different atoms occurring respectively in two different disjuncts d_1 and d_2 of q, such that $\boldsymbol{x} = x_1, \ldots, x_n$, $\boldsymbol{y} = y_1, \ldots, y_m$, $\boldsymbol{z} = z_1, \ldots, z_n$, $\boldsymbol{w} = w_1, \ldots, w_m$ are sequences of terms. Then, we say that a_1 and a_2 are* interacting *in q if*

1. *the sequences of terms in key position in a_1 and a_2 unify, i.e., there exists a unifier between $\boldsymbol{x}$ and $\boldsymbol{z}$;*
2. *there exists $j \in \{1, \ldots, m\}$ such that y_j and w_j are not identical constants;*
3. *a_1 (resp. a_2) is such that either a_1 is not a leaf in the join graph of d_1 (resp. d_2) or there exists a non-key argument of a_1 (resp. a_2) which is bound.*

Notice that the atoms $r(a, c_1)$ and $r(a, c_2)$ in the example above are interacting atoms in the query q. Now we are able to define the class we were looking for.

Definition 3. *A UCQ q belongs to the class UCQ_{NI} of* non-interacting *UCQs if:*

- *each disjunct d_i of q is in $\mathcal{C}^{+}_{tree}$;*
- *there do not exist two interacting atoms a_1 and a_2 in q.*

It is easy to see that the query in Example 1 belongs to the class UCQ_{NI}, whereas the query in in Example 2 does not.

Theorem 2. *Let $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$ be a database schema, $q \in UCQ_{NI}$ be a query over $\mathcal{S}$. Then, the FOL query q_f returned by the algorithm UCQ-FolRewrite$(q, \mathcal{S})$ is a FOL-rewriting of q under $\mathcal{S}$.*

In other words, the above theorem states that the problem of consistent query answering under key dependencies is FOL-reducible for the class UCQ_{NI}.

4 Algorithm

In this section we try to overcome the limitations of the rewriting technique presented in the previous section, by defining a new FOL-rewriting algorithm for UCQs. Based on such an algorithm, we are able to identify the class of acyclically interacting queries, a class of UCQs which extends the class UCQ_{NI} defined in Section 3, and to prove that acyclically interacting queries constitute a class of FOL-reducible queries under key dependencies.

4.1 The algorithm UCQ-FolRewriteNew

We are now ready to define the algorithm UCQ-FolRewriteNew, a FOL rewriting algorithm for UCQs that, differently from the previous algorithm UCQ-FolRewrite, takes into account the semantic interactions between the query disjuncts.

In the algorithm UCQ-FolRewriteNew(and in the other algorithms iteratively invoked by UCQ-FolRewriteNew and presented in this section), with a little abuse of terminology we call *typed query associated to a query* q the query q^t obtained from q by replacing each atom with its typing. Analogously, the *typed disjunct associated to a disjunct* d is the disjunct d^t obtained from d by replacing each atom with its typing. Coherently to the above definitions, when the operator JG, used for constructing the join graph of a query, is applied to a typed query q^t, the nodes of $JG(q^t)$ respect the typing specified by q^t, i.e., each node of the graph is labeled with the corresponding typing indicated in q^t. We also point out that in the query Q in input to UCQ-FolRewriteNew, each variable symbol only occurs in a single disjunct of Q, and the new variables introduced by the algorithm NodeRewriteNew(see below) are always fresh symbols with respect to all the executions of the algorithm.

The algorithm UCQ-FolRewriteNew is presented in Figure 4.

```
Algorithm UCQ-FolRewriteNew($Q$,$\mathcal{S}$)
Input: a first-order reducible UCQ $Q = h(x_1, \ldots, x_n) \text{ :- } d_1 \vee \ldots \vee d_m$;
       schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$
Output: FOL query (representing the rewriting of $Q$)
begin
  let $Q^t$ be the typed query associated to $Q$;
  for $i := 1$ to $m$ do
    let $d_i^t$ be the typed disjunct associated to $d_i$;
  return $\{x_1, \ldots, x_n \mid \bigvee_{i=1,\ldots,m} \mathsf{DisjunctRewrite}(d_i^t, Q^t, \mathcal{S}, (\emptyset, \emptyset))\}$
end
```

Fig. 4. The algorithm UCQ-FolRewriteNew

Such algorithm calls the algorithm DisjunctRewrite, described in Figure 5, which computes the rewriting of a single disjunct d_i of the UCQ, by recursively calling the subroutine NodeRewriteNew, presented in Figure 6.

The algorithm NodeRewriteNew is actually a new version of the algorithm NodeRewrite presented in Section 3. Notably, NodeRewriteNew (executed on a disjunct d) recursively calls DisjunctRewrite to properly take into account the role of other disjuncts which have relation symbols in common with d. More precisely, the rewriting produced by the algorithm NodeRewriteNew suitably encodes the possibility that, given an assignment of the head variables of the UCQ Q (i.e., a tuple of constants $\boldsymbol{t}$), an "opponent fact" $r(\boldsymbol{c}')$ to a fact $r(\boldsymbol{c})$ (i.e., such that $r(\boldsymbol{c})$

```
Algorithm DisjunctRewrite($d, Q, \mathcal{S}, \mathcal{P}$)
Input:
  a typed union of conjunctive queries $Q = h(x_1, \ldots, x_n)$ :- $d_1 \vee \ldots \vee d_m$;
  a typed disjunct $d$ that appears in $Q$;
  schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$;
  $\mathcal{P} = (M, E)$ where $M$ is a list of atoms and $E$ is a set of equalities;
Output: FOL query (representing the rewriting of the disjunct $d$)
begin
  $q = h(x_1, \ldots, x_n)$ :- $d$;
  compute $JG(q)$;
  return $\{ \bigwedge_{N \in roots(JG(q))}$ NodeRewriteNew($JG(q), N, Q, \mathcal{S}, \mathcal{P}$)$\}$
end
```

Fig. 5. The algorithm DisjunctRewrite

and $r(\boldsymbol{c}')$ have the same key) that belongs to an image of $\boldsymbol{t}$ w.r.t. a disjunct d_i of Q, might not be part of any image of the same disjunct d_i but may be part of an image of $\boldsymbol{t}$ w.r.t. *another* disjunct d_j of the query. Thus, the formula in the FOL-rewriting must look for the existence of such an image of d_j. It can be shown that this *non-local* check must be performed only in the presence of interacting atoms in Q. More precisely, when NodeRewriteNew is computing the rewriting of a node corresponding to an atom a, it must recursively invoke DisjunctRewrite only for each disjunct d_j such that there is an atom b in d_j that is interacting with a in Q.

In the algorithms DisjunctRewrite and NodeRewriteNew, $\mathcal{P}$ is the pair (M, E) in which M is a list of atoms and E is a set of equalities of terms. Each atom in the list M is obtained by means of the operator Π_{kd} applied to a typing $a = r(x_1/t_1, \ldots, x_n/t_n)$ (i.e., a label of a node of a join graph). $\Pi_{kd}(a)$ returns the atom $r(x_{i_1}, \ldots, x_{i_a})$, where $i_1, \ldots, i_a$ are positions of the arguments of a of type KU or KB, i.e., $x_{i_1}, \ldots, x_{i_a}$ are the key-arguments of the atom $r(x_1, \ldots, x_n)$. The function call $occurs(\Pi_{kd}(a), \mathcal{P})$ returns true if the atom $\Pi_{kd}(a)$ is in the list M or it can be constructed from an atom of M according to the equalities of terms contained in E. Otherwise $occurs(\Pi_{kd}(a), \mathcal{P})$ returns false. This check avoids useless calls of the algorithm DisjunctRewrite and guarantees termination of the procedure. Furthermore, $Int(N)$ denotes the set of disjuncts of Q that contain atoms interacting with the atom corresponding to the node N (and different from the disjunct which N belongs to); $u_1, \ldots, u_n$ denote the terms occurring the interacting atom in the disjunct d_j, and $u_{j_1}, \ldots, u_{j_s}$ are the variables occurring in such atom; τ is an operator which modifies the typing of each atom (in the disjunct d_j and in the query Q in the two invocations $\tau(d_j)$ and $\tau(Q)$, respectively) by assigning KB to the key arguments of the interacting atom, and B to the other (non-key) arguments.

```
**Algorithm** NodeRewriteNew$(JG(q), N, Q, \mathcal{S}, \mathcal{P})$
**Input:** join graph $JG(q)$;
        node $N$ of $JG(q)$;
        a typed query $Q = h(x_1, \ldots, x_n) \text{ :- } d_1 \vee \ldots \vee d_m$;
        schema $\mathcal{S} = \langle \mathcal{A}, \mathcal{K} \rangle$;
        $\mathcal{P} = (M, E)$ where $M$ is a list of atoms and $E$ is a set of equalities;
**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**;
  **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** $w_i := z_i$;
    **if** $occurs(\Pi_{kd}(a), \mathcal{P})$
    **then** $f_2 = \left( \bigwedge_{N' \in jgsucc(N)} \text{NodeRewriteNew}(JG(q), N', Q, \mathcal{S}, \mathcal{P}) \wedge \bigwedge_{i \in \{\ell_1, \ldots, \ell_h\}} w_i = x_i \right)$
    **else begin**
      $M = M \cup \{\Pi_{kd}(a)\}$;
      $E = E \cup \{w_1 = u_1, \ldots, w_n = u_n\}$;
      $f_2 := \forall z_{p_1}, \ldots, z_{p_c}.\ r(w_1, \ldots, w_n) \rightarrow$
          $\left( \bigwedge_{N' \in jgsucc(N)} \text{NodeRewriteNew}(JG(q), N', Q, \mathcal{S}, \mathcal{P}) \wedge \bigwedge_{i \in \{\ell_1, \ldots, \ell_h\}} w_i = x_i \right) \vee$
          $\bigvee_{d_j \in Int(N)} \exists u_{j_1}, \ldots, u_{j_s}.w_1 = u_1 \wedge \ldots \wedge w_n = u_n \wedge$
                $\wedge$ DisjunctRewrite$(\tau(d_j), \tau(Q), \mathcal{S}, \mathcal{P})$;
      **return** $f_1 \wedge f_2$
    **end**
  **end**
**end**
```

Fig. 6. The subroutine NodeRewriteNew

Example 3. Consider again the query q of Example 2, and execute the algorithm $\mathsf{UCQ\text{-}FolRewriteNew}(q, \mathcal{S})$. Then the FOL-rewriting produced by the algorithm is as follows

$$\{\ \mid (\exists y_1.r(y_1, c_1) \wedge \forall y_2.r(y_1, y_2) \rightarrow y_2 = c_1 \vee (\exists y_3.y_1 = y_3 \wedge y_2 = c_2 \wedge r(y_3, c_2))) \vee$$
$$(\exists y_1.r(y_1, c_2) \wedge \forall y_2.r(y_1, y_2) \rightarrow y_2 = c_2 \vee (\exists y_3.y_1 = y_3 \wedge y_2 = c_1 \wedge r(y_3, c_1)))\}.$$

Notice that in such a case the check on the execution of DisjunctRewrite, which we have talked about above, avoids the execution of identical calls of such a procedure. □

We finally point out that the rewriting produced by the algorithm can be refined in order to get a simplified version of it (which could be evaluated in a more efficient way). However, this is outside the scope of the present paper.

4.2 Termination and correctness

The algorithm UCQ-FolRewriteNew in general does not terminate. In order to charaterize the class of queries for which the algorithm terminates, we give the following definitions.

Definition 4. *Given two atoms $a = r(x_1, \ldots, x_n, w_1, \ldots, w_m)$, $b = r(y_1, \ldots, y_n, z_1, \ldots, z_m)$, where $key(r) = \{1, \ldots, n\}$, we say that a and b are* key-unifiable *if, for each i s.t. $1 \leq i \leq n$: (i) x_i is a variable; or (ii) y_i is a variable; or (iii) $x_i = y_i$. If a and b are key-unifiable, we denote by $\sigma_{a \rightarrow b}$ the substitution $\{y_i \leftarrow x_i \mid 1 \leq i \leq n$ and y_i is a variable$\}$.*[2]

Definition 5. *A UCQ $Q = \{x_1, \ldots, x_m \mid d_1 \vee \ldots \vee d_n\}$ has a* $\vee$-cycle *if there exists a sequence $d^1_{i_1}, \ldots, d^k_{i_k}$ (with $k > 1$) and a sequence of relation symbols $r_{j_1}, \ldots, r_{j_{k-1}}$ such that:*

- *$d^1_{i_1} = d_{i_1}$;*
- *$i_k = i_1$;*
- *for each h s.t. $1 \leq h \leq k-1$, r_{j_h} occurs both in $d^h_{i_h}$ and in $d^{h+1}_{i_{h+1}}$;*
- *let a be the atom with relation r_{j_h} in $d^h_{i_h}$ and let b be the atom with relation r_{j_h} in $d_{i_{h+1}}$. Then, a and b are key-unifiable. Moreover, for each h s.t. $1 \leq h \leq k-1$, $d^{h+1}_{i_{h+1}} = \sigma_{a \rightarrow b}(d_{i_{h+1}})$;*
- *the key arguments of $r_{j_1}(d^k_{i_k})$ contain at least one existential variable not occurring in the key arguments of $r_{j_1}(d^1_{i_1})$.*

In a $\vee$-cycle, the disjuncts $d^j_{i_j}$ and $d^{j+1}_{i_{j+1}}$ are connected through two atoms a (occurring in d_{i_j}) and b (occurring in $d_{i_{j+1}}$) such that a and b are on the same relation symbol r. The key arguments of a are "passed" to b, thus $d_{i_{j+1}}$ is transformed according to such a substitution.

[2] In the definition of key-unifiable atoms, head variables are considered as constants.

Example 4. Let us consider the UCQ

$$q :- (r_1(x,y) \wedge r_2(y,z)) \vee (r_2(x,y) \wedge r_1(y,z))$$

We now show that there is an $\vee$-cycle in q which starts from the atom $r_1(x,y)$ of the first disjunct. Indeed, the $\vee$-cycle is due to: (i) the presence of the atom $r_1(x,y)$ in the first disjunct and the atom $r_1(y,z)$ in the second disjunct, which constitutes the first part of the cycle; (ii) the presence of the atom $r_2(x,y)$ in the second disjunct and the atom $r_2(y,z)$ in the first disjunct, which constitutes the second part of the cycle; (iii) the fact that, after this cycle, the key argument of the atom $r_1(x,y)$ in the first disjunct is unbound. $\square$

It is easy to verify that a a necessary condition for a UCQ Q to have a $\vee$-cycle is the presence of interacting atoms in Q.

Definition 6. *A UCQ $Q = \{x_1, \ldots, x_m \mid d_1 \vee \ldots \vee d_n\}$ belongs to the class UCQ_{AI} of* acyclically interacting *UCQs if:*

1. *each conjunction d_i is such that the CQ $\{x_1, \ldots, x_m \mid d_i\}$ belongs to $\mathcal{C}^{+}_{tree}$;*
2. *there are no $\vee$-cycles in Q.*

Informally, according to the above definition, a UCQ Q is acyclically interacting if the interacting atoms in the query disjuncts are such that they do not constitute a $\vee$-cycle in Q, i.e., a cycle of interactions that, starting from an atom $r(\boldsymbol{x})$, cycles back to the same atom introducing at least one existential variable in the key arguments of the atom.

From the semantic viewpoint, the presence of an $\vee$-cycle implies that, when checking for the opponents of an image $r(\boldsymbol{t})$ of a query atom a, we need to check for the opponents of another image $r(\boldsymbol{t}')$ of a, where $\boldsymbol{t}'$ has in its key arguments a new value that does not occur neither in $\boldsymbol{t}$ nor in the query Q. This immediately implies non-termination of the algorithm UCQ-FolRewriteNew, since at every such iteration there are new key arguments in the call to NodeRewriteNew for the atom a. Vice versa, the absence of such a cycle implies termination of the algorithm, since no new term (with respect to the terms occurring in the query Q) is introduced in the calls to NodeRewriteNew, hence the number of possible instantiations of the calls to NodeRewriteNew is finite.

The following property formalizes the fact that the class of acyclically interacting queries is precisely the class of UCQs for which the algorithm UCQ-FolRewriteNew terminates.

Theorem 3. *Let Q be a UCQ, and let $\mathcal{S}$ be a schema. The execution of the algorithm UCQ-FolRewriteNew with input Q terminates if and only if Q is acyclically interacting.*

Moreover, the following theorem establishes soundness and completeness of the algorithm UCQ-FolRewriteNew for the class of acyclically interacting UCQs.

Theorem 4. *If Q is acyclically interacting, 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 $t \in Q_r^{\mathcal{D}}$, where Q_r is the FOL query returned by* UCQ-FolRewriteNew(Q) *(i.e., the FOL query returned by the algorithm* UCQ-FolRewrite(Q) *is a FOL-rewriting of q under $\mathcal{S}$).*

As a corollary of the above theorem, we obtain that the class UCQ_{AI} is FOL-reducible.

Finally, we point out that:

- the class of UCQ_{AI} is a proper superset of the class of UCQs UCQ_{NI} analyzed in Section 3 and for which the algorithm UCQ-FolRewrite is complete;
- if Q is a query in the class UCQ_{NI}, the algorithms UCQ-FolRewrite(Q) and UCQ-FolRewriteNew(Q) return exactly the same FOL query.

5 Discussion and Conclusions

We believe that the study of first-order reducibility of consistent query answering for unions of conjunctive queries is relevant per se, since the possibility of expressing unions in queries is an important feature which has practical relevance. However, we argue that the ability of handling unions of conjunctive queries is necessary in order to solve via FOL-rewriting techniques the problem of consistent query answering for (unions of) conjunctive queries issued over database schemas which contain keys and foreign keys under the *loosely-sound* semantics, a repair semantics which allows for properly dealing with both incomplete and inconsistent databases[3] [4, 5].

Formally, given a database schema $\mathcal{S}$ which contains keys and foreign keys, a *loosely-sound repair* of a database $\mathcal{D}$ is any database legal for $\mathcal{S}$ that contains a repair (as so far intended in the present paper and formally specified in Section 2) of $\mathcal{D}$ under $\mathcal{S}'$, where $\mathcal{S}'$ is obtained from $\mathcal{S}$ disregarding foreign key dependencies. Roughly speaking, such semantics adds the ability to deal with inconsistent databases to the first-order semantics commonly adopted for dealing with incomplete databases. Indeed, in intuitive terms, it maintains the ability of the first-order semantics to deliberately add facts to a database instance (property that can be exploited to satisfy those dependencies that can be satisfied by adding facts, as foreign keys), but it also allows for a (minimal) deletion of facts, thus enabling the repairing of database instances with respect to those dependencies, as key dependencies, that may generate inconsistency according to the first-order semantics.

Notably, as showed in [5], in order to solve consistent query answering for (unions of) conjunctive queries under the loosely-sound semantics, it is possible to separately dealing with keys and foreign keys. According to the procedure provided in [5], a query q is first processed only according to the foreign keys issued over the database schema. Such a pre-processing produces a union of

[3] Notice that also inclusion dependencies of a particular form which guarantees decidability of the consistent query answering problem might be considered [4].

conjunctive queries Q. Then, it is sufficient to solve consistent query answering for the UCQ Q over the same database schema in which foreign keys have been dropped. It is immediate to see, that if the query Q obtained is of class UCQ_{AI}, then to solve the second problem we can apply the algorithm UCQ-FolRewrite presented in this paper.

Consequently, even if still preliminary, the analysis of first-order reduction of unions of conjunctive queries we have presented turns out to be a necessary first step in order to arrive to the definition of analogous methods for (unions of) conjunctive queries under key and foreign key dependencies.

6 Acknowledgments

This research has been partially supported by the FET project INFOMIX: Boosting Information Integration, funded by EU under contract number IST-2001-33570, the FET project TONES: Thinking ONtologiES, funded by the EU under contract number FP6-7603, and by MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison Wesley Publ. Co., 1995.
2. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the Eighteenth ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79, 1999.
3. Loreto Bravo and Leopoldo Bertossi. Logic programming for consistently querying data integration systems. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (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 *Proceedings of the Twentysecond ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (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 *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (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 Leopoldo Bertossi, Anthony Hunter, and Torsten Schaub, editors, *Inconsistency Tolerance*, volume 3300 of *Lecture Notes in Computer Science*, pages 119–150. Springer, 2005.
7. Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. Computing consistent query answers using conflict hypergraphs. In *Proceedings of the Thirteenth International Conference on Information and Knowledge Management (CIKM 2004)*, pages 417–426, 2004.

8. Thomas Eiter, Michael Fink, Gianluigi Greco, and Domenico Lembo. Efficient evaluation of logic programs for querying data integration systems. In *Proceedings of the Nineteenth International Conference on Logic Programming (ICLP'03)*, pages 163–177, 2003.
9. Ariel Fuxman, Elham Fazli, and Renée J. Miller. ConQuer: Efficient management of inconsistent databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 155–166, 2005.
10. Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *Proceedings of the Tenth International Conference on Database Theory (ICDT 2005)*, volume 3363 of *LNCS*, pages 337–351. Springer, 2005.
11. Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.
12. Luca Grieco, Domenico Lembo, Marco Ruzzi, and Riccardo Rosati. Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In *Proceedings of the Fourteenth International Conference on Information and Knowledge Management (CIKM 2005)*, pages 792–799, 2005.
13. Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 2006. To appear.