Query Rewriting under Non-Guarded Rules

Andrea Calì[2,1,3], Georg Gottlob[1,2] and Andreas Pieris[1]

[1]Computing Laboratory, University of Oxford, UK
[2]Oxford-Man Institute of Quantitative Finance, University of Oxford, UK
[3]Department of Information Systems and Computing, Brunel University, UK

firstname.lastname@comlab.ox.ac.uk

Abstract. We address the problem of answering conjunctive queries over knowledge bases, specified by sets of first-order sentences called tuple-generating dependencies (TGDs). This problem is highly relevant to query optimization, information integration, and ontological reasoning. We present a rewriting algorithm, inspired by resolution in Logic Programming, which is capable of dealing with an expressive class of TGDs, called *sticky TGDs*. Given a set of sticky TGDs and a conjunctive query, the algorithm produces a first-order query that can be then evaluated over the data, providing the correct answers. In this way, we establish that conjunctive query answering under sticky TGDs is in the highly tractable class AC_0 in the data complexity.

1 Introduction

Answering queries over knowledge bases is a fundamental problem in knowledge representation. It has been employed in schema integration, information integration, and service discovery (see, e.g., [11]). In the database field, the problem of answering queries on incomplete data under constraints (a.k.a. *dependencies*) is tightly related to the one of query containment [12]; in fact, the two problems are mutually reducible (see, e.g., [3]). A milestone paper in the field is [17]; it addresses conjunctive query containment under functional and inclusion dependencies, a class of constraints that is highly relevant in practice. The results of [17] were later extended in [7].

Other works consider different classes of constraints. For instance, [2, 9] consider constraints tailored to express Entity-Relationship schemas, and [23] deals with expressive constraints based on Answer Set Programming. [6] introduces and studies first-order constraints derived from a “light” version of F-logic [18], called *F-logic Lite*. Another relevant formalisms for knowledge bases, especially in the Semantic Web, is the *DL-lite family*; in [10, 22] tractable query answering techniques under DL-lite knowledge bases are presented.

A more general approach is adopted in the recent papers [3, 4]; in such works, instead of focusing on a specific logic formalism, a general family of languages is considered, called Datalog$^{\pm}$, that captures and extends a large class of knowledge representation formalisms in the literature, in particular the DL-lite family. The constraints of [3, 4] are inspired by the *guarded fragment* of first-order logic [16], and among them are the languages of *linear Datalog*$^{\pm}$, *guarded Datalog*$^{\pm}$, and *weakly-guarded Datalog*$^{\pm}$. It is important to notice that Datalog$^{\pm}$-rules are relevant *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)*, which are generalizations of inclusion dependencies and functional dependencies, respectively; we shall not deal with EGDs in this paper. TGDs are first-order constraints that express an implication from a conjunction of atoms to another. For instance, the TGD $\forall E \forall G\, \mathit{director}(E), \mathit{leads}(E, G) \rightarrow \exists D\, \mathit{manages}(E, D)$ states that each director that leads some group necessarily manages at least one department.

A large part of the aforementioned works on query answering and containment are based on the well-known notion of *chase*, widely used for dependency implication and query containment under dependencies [19, 17]. The chase is a procedure that enforces the satisfaction of TGDs by the addition of suitable tuples. For example, consider an instance D consisting of the atoms $\{\mathit{director}(d), \mathit{leads}(d, g)\}$; the above TGD is not satisfied by D, therefore the chase adds a suitable atom $\mathit{manages}(d, z)$, where z is a so-called *labeled null*, i.e., a placeholder for an unknown value. Several works have recently studied query evaluation problems for settings where the chase terminates and thus generates a finite solution. This is the typical setting, for example, of *data exchange* [15], where the materialization (and therefore the finiteness) of the chase is a requirement. To this aim, syntactic restrictions on TGDs, e.g., *weak acyclicity* [13], have been proposed. Little was known about the cases where the chase is (in general) infinite, with the notable exception of the classical work of Johnson and Klug [17].

In this work, along the lines of [3, 4], we also adopt a general approach. We consider an expressive class of TGDs, called *sticky TGDs (STGDs)*, first introduced in [5] as an addition to the Datalog$^{\pm}$ family. The chase under STGDs is not guaranteed to terminate. Stickiness is formally defined in Section 3 by an efficiently testable condition involving variable-marking. An equivalent definition, which gives a better understanding of such class of constraints, is as follows. For every instance D, assume that in the construction of the chase of D under a set Σ of TGDs, we apply a TGD $\sigma \in \Sigma$ that has a variable X appearing more than once in its body; assume also that X maps (via homomorphism) on the symbol z, and that by virtue of this application the atom $\mathbf{a}$ is introduced. Then, z appears in $\mathbf{a}$ and in all atoms resulting from some chase derivation involving $\mathbf{a}$, "sticking" to them (hence the name "sticky TGDs") [5].

Our main contribution in this paper is an algorithm, similar to resolution and partial evaluation in Logic Programming, for query answering under STGDs. Our approach is based on *query rewriting*, that is: given a query, such query is rewritten into another query that encodes the information about the constraints and that, evaluated on the data, returns the correct answers. We consider the class of conjunctive queries (CQs), also called *select-project-join* queries in the database literature. The algorithm, inspired by the ones in [2, 8], takes as input a set of STGDs and a conjunctive query, and it computes a (finite) rewriting expressed in union of conjunctive queries (UCQs).

Since our rewriting algorithm is sound and complete (established by making use of the notion of chase), terminates under STGDs, and the rewritten query is a UCQs, answering conjunctive queries in this case is in the highly tractable class AC_0 w.r.t. the size of the data[1], as already stated in [5], where no details about a rewriting technique are given. We recall that AC_0 is the complexity class of recognizing words in languages defined by constant-depth Boolean circuits with an (unlimited fan-in) AND and OR gates (see, e.g., [20]). Notice that a UCQs can be immediately translated into an SQL query, therefore the rewriting algorithm is especially suitable for real-world applications based on relational DBMSs. Notice that the size of the rewritten query is worst-case exponential w.r.t. the size of the given query and the size of the given set of constraints. While query answering remains highly tractable in data complexity, this leaves space for further optimizations. Generating "smaller" rewritings is one of the directions currently pursued to improve efficiency of query processing in this context [21].

In [5] it is shown, similarly to what is done in [4], that STGDs enriched with *negative constraints* of the form $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \bot$, where $\varphi(\mathbf{X})$ is a conjunction of atoms and $\bot$ is the constant *false*, and with EGDs (key dependencies, in this case) that do not interact with the STGDs, are capable of expressing languages in the DL-lite family. Therefore, all tractability results on conjunctive query answering in DL-lite are special cases of our results.

2 Preliminaries

In this section we recall some basics on databases, queries, TGDs and the TGD chase procedure.

General. We define the following pairwise disjoint (infinite) sets of symbols: *(i)* a set Γ of *constants* (constitute the "normal" domain of a database), *(ii)* a set Γ_f of *labeled nulls* (used as placeholders for unknown values, and thus can be also seen as variables), and *(iii)* a set Γ_v of *variables* (used in queries and dependencies). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. A lexicographic order is defined on $\Gamma \cup \Gamma_f$, such that every value in Γ_f follows all those in Γ.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. We write r/n to denote that the predicate r has arity n. A *position* $r[i]$ (in a schema $\mathcal{R}$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r/n is a relation, and $t_1, \ldots, t_n$ are terms. For an atom $\mathbf{a}$, we denote as $dom(\mathbf{a})$, $var(\mathbf{a})$ and $pred(\mathbf{a})$ the set of its terms, the set of its variables and its predicate, respectively. These notations naturally extends to sets and conjunctions of

[1] The complexity w.r.t. the data size is known as *data complexity*, and it is relevant since in practice the size of the data is much larger than the size of the schema.

atoms. An atom is called *ground* if all of its terms are constants of Γ. Conjunctions of atoms are often identified with the sets of their atoms.

A *substitution* from one set of symbols S_1 to another set of symbols S_2 is a function $h : S_1 \rightarrow S_2$ defined as follows: *(i)* $\varnothing$ is a substitution (empty substitution), *(ii)* if h is a substitution, then $h \cup \{X \rightarrow Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$, and h does not already contain some $X \rightarrow Z$ with $Y \neq Z$. If $X \rightarrow Y \in h$ we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution $h : \mathit{dom}(A_1) \rightarrow \mathit{dom}(A_2)$ such that: *(i)* if $t \in \Gamma$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2. The notion of homomorphism naturally extends to conjunctions of atoms.

Databases and Queries. A *database (instance)* D for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$ (a.k.a. *facts*), where $r/n \in \mathcal{R}$ and $\mathbf{t} \in (\Gamma \cup \Gamma_f)^n$. We denote as $r(D)$ the set $\{\mathbf{t} \mid r(\mathbf{t}) \in D\}$.

A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, has the form $q(\mathbf{X}) = \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables or constants in Γ, and the length of $\mathbf{X}$ is n. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $\mathit{body}(q)$. A *Boolean CQ (BCQ)* is a CQ of zero arity. The *answer* to a CQ q/n of the form $q(\mathbf{X}) = \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted as $q(D)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \rightarrow \Gamma \cup \Gamma_f$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over D, denoted as $D \models q$, iff $\langle\rangle \in q(D)$, or, equivalently, $q(D) \neq \varnothing$.

Dependencies. Given a schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ over $\mathcal{R}$ is a first-order formula $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $\mathit{body}(\sigma)$ and $\mathit{head}(\sigma)$, respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. Such σ is satisfied by a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq D$.

We now define the notion of *query answering* under TGDs. Given a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $\mathit{mods}(D, \Sigma)$, is the set of all databases B such that $B \models D \cup \Sigma$. The *answer* to a CQ q w.r.t. D and Σ, denoted as $\mathit{ans}(q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(B)$ for each $B \in \mathit{mods}(D, \Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is *positive*, denoted as $D \cup \Sigma \models q$, iff $\mathit{ans}(q, D, \Sigma) \neq \varnothing$. Note that query answering under general TGDs is undecidable [1], even when the schema and the set of TGDs are fixed [3].

We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent [3]. Moreover, it is easy to see that the query output tuple problem (as a decision version of CQ evaluation) and BCQ evaluation are AC$_0$-reducible to each other. Henceforth, we thus focus only on the BCQ evaluation problem. All complexity results carry over to the other problems.

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [19], and later for checking query containment [17]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*. The TGD chase rule comes in two different equivalent fashions: *oblivious* and *restricted* [3], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for better technical clarity. The TGD chase rule is the building block of the chase.

TGD Chase Rule. Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, then: *(i)* define $h' \supseteq h$ such that $h'(Z_i) = z_i$ for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_f$ is a "fresh" labeled null not introduced before, and following lexicographically all those introduced so far, and *(ii)* add to D the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$ if not already in D.

Given a database D and set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase rule in a breadth-first fashion, which leads as result to a (possibly infinite) chase for D and Σ, denoted as $chase(D, \Sigma)$. The formal definition of the chase algorithm is omitted due to space reasons and can be found, for instance, in [3]. We denote as $chase^{[k]}(D, \Sigma)$ the *initial segment* of the chase for D and Σ obtained by applying $k \geqslant 0$ times the TGD chase rule.

Example 1 ([5]). Let $\mathcal{R} = \{r, s\}$. Consider the set Σ of TGDs over $\mathcal{R}$ constituted by the TGDs $\sigma_1 = r(X, Y), s(Y) \rightarrow \exists Z\, r(Z, X)$ and $\sigma_2 = r(X, Y) \rightarrow s(X)$. Let D be the database for $\mathcal{R}$ consisting of the two atoms $r(a, b)$ and $s(b)$. During the construction of $chase(D, \Sigma)$ we first apply σ_1, and we add the atom $r(z_1, a)$, where z_1 is a "fresh" null. Moreover, σ_2 is applicable and we add the atom $s(a)$. Now, σ_1 is applicable and the atom $r(z_2, z_1)$ is obtained, where z_2 is a "fresh" null. Also, σ_2 is applicable and the atom $s(z_1)$ is generated. It is clear that there is no finite chase. Satisfying both σ_1, σ_2 would require to construct the infinite instance $D \cup \{r(z_1, a), s(a), r(z_2, z_1), s(z_1), r(z_3, z_2), s(z_2), \ldots\}$. ■

It is well-known that the (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each database $B \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to B [15, 14]. Using this fact it can be shown that for a BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

3 Sticky TGDs

In this section we recall the class of *sticky TGDs* introduced in [5]. As we shall see, query answering under sticky TGDs is highly tractable in data complexity.

Definition 1 ([5]). Consider a set Σ of TGDs over a schema $\mathcal{R}$. We mark the variables that occur in the body of the TGDs of Σ according to the following marking procedure. First, for each TGD $\sigma \in \Sigma$ and for each variable V in $body(\sigma)$, if there exists an atom $\mathbf{a}$ in $head(\sigma)$ such that V does not appear in $\mathbf{a}$, then we mark each occurrence of V in $body(\sigma)$. Now, we apply exhaustively (i.e., until a fixpoint is reached) the following step: for each TGD $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at position π, then for every TGD $\sigma' \in \Sigma$ (including the case $\sigma' = \sigma$), we mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ at the same position π. We say that Σ is a set of *sticky TGDs (STGDs)* if there is no TGD $\sigma \in \Sigma$ such that a marked variable occurs in $body(\sigma)$ more than once.

Example 2 ([5]). Consider the following relational schema $\mathcal{R}$:

$$\begin{array}{l} dept(\mathsf{Dept\_Id}, \mathsf{Mgr\_Id}),\\ emp(\mathsf{Emp\_Id}, \mathsf{Dept\_Id}, \mathsf{Area}, \mathsf{Project\_Id}),\\ runs(\mathsf{Dept\_Id}, \mathsf{Project\_Id}),\\ in\_area(\mathsf{Project\_Id}, \mathsf{Area}),\\ external(\mathsf{Ext\_Id}, \mathsf{Area}, \mathsf{Project\_Id}). \end{array}$$

The fact that each department has as a manager an employee can be represented by the TGD

$$dept(V, W) \rightarrow \exists X \exists Y \exists Z\; emp(W, X, Y, Z).$$

The fact that each employee works on some project that falls into his/her area of specialization ran by his/her department can be represented by the TGD

$$emp(V, W, X, Y) \rightarrow \exists Z\; dept(W, Z), runs(W, Y), in\_area(Y, X).$$

Finally, the fact that for each project ran by some department there exists an external cooperator, specialized on the area of the project, that works on it can be represented by the TGD

$$runs(W, X), in\_area(X, Y) \rightarrow \exists Z\; external(Z, Y, X).$$

Let Σ be the set constituted by the above three TGDs. According to the variable-marking procedure in Definition 1, we mark the variables as follows (we mark variables with a cap, e.g., $\hat{X}$):

$$\begin{array}{l} dept(\hat{V}, \hat{W}) \rightarrow \exists X \exists Y \exists Z\, emp(\hat{W}, \hat{X}, \hat{Y}, \hat{Z})\\ emp(\hat{V}, \hat{W}, \hat{X}, \hat{Y}) \rightarrow \exists Z\; dept(\hat{W}, \hat{Z}), runs(\hat{W}, Y), in\_area(Y, X)\\ runs(\hat{W}, X), in\_area(X, Y) \rightarrow \exists Z\; external(Z, Y, X) \end{array}$$

Clearly, for each TGD $\sigma \in \Sigma$, there is no marked variable that occurs in $body(\sigma)$ more than once. Therefore, Σ is a set of STGDs. It is important to observe that the set Σ is neither weakly-acyclic [15], nor guarded [3], nor weakly-guarded [3]. In fact, the class of STGDs is incomparable to the above three known classes of TGDs. ■

We recall that query answering under (general) TGDs is equivalent to query answering under TGDs with single-atom heads [3]. This is established by providing a LOGSPACE transformation from (general) TGDs to TGDs with single-atom heads. Since this transformation preserves the syntactic condition of STGDs, henceforth we assume w.l.o.g. that every TGD has just one atom in its head.

4 Query Answering by Rewriting

We say that a class $\mathcal{C}$ of TGDs is *first-order rewritable*, henceforth abbreviated as *FO-rewritable*, iff for every set Σ of TGDs in $\mathcal{C}$, and for every BCQ q, there exists a first-order query q_{Σ} such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_{\Sigma}$ [4]. Since answering first-order queries is in the class $\textsc{ac}_0$ in data complexity [24], it immediately follows that for FO-rewritable TGDs, query answering is in $\textsc{ac}_0$ in data complexity.

In this section we establish that STGDs are FO-rewritable by providing a query rewriting algorithm, inspired by resolution in Logic Programming, that allow us to reformulate the given BCQ q into a union of BCQs Q_r, that encodes the information about the given STGDs, and then evaluate Q_r over the given database to obtain the correct answer. Formally, a *union of BCQs* over a schema $\mathcal{R}$ is a set Q of BCQs over $\mathcal{R}$, where each $q \in Q$ uses the same predicate symbol in the head. Q is said to have positive answer over a database D, denoted as $D \models Q$, iff there exists $q \in Q$ such that $D \models q$. Note that a union of BCQs is trivially a first-order query. Before we proceed further we give some preliminary definitions.

Given a BCQ q we say that a certain variable is *bound* in q if it occurs more than once in $body(q)$, otherwise is called *unbound*. A *bound term* in q is either a bound variable or a constant of Γ. Given two atoms $\mathbf{a}$ and $\mathbf{b}$ we say that *unify* if there exists a substitution γ, called *unifier* for $\mathbf{a}$ and $\mathbf{b}$, such that $\gamma(\mathbf{a}) = \gamma(\mathbf{b})$. A *most general unifier (MGU)* is a unifier for $\mathbf{a}$ and $\mathbf{b}$, denoted as $\gamma_{\mathbf{a},\mathbf{b}}$, such that for each other unifier γ for $\mathbf{a}$ and $\mathbf{b}$ there exists a substitution γ' such that $\gamma = \gamma' \circ \gamma_{\mathbf{a},\mathbf{b}}$. Note that if two atoms unify, then there exists a MGU. Furthermore, the MGU for two atoms is unique (modulo variable renaming).

We now define the important notion of *applicability* of a TGD to an atom that occurs in the body of a BCQ.

Definition 2. Let $\mathcal{R}$ be a relational schema. Consider a TGD σ over $\mathcal{R}$, a BCQ q over $\mathcal{R}$, and an atom $\mathbf{a} \in body(q)$. We say that σ is *applicable* to $\mathbf{a}$ if the following conditions are satisfied: *(i)* $\mathbf{a}$ and $head(\sigma)$ unify (recall that we assume single-atom head TGDs); we denote as $\gamma_{\mathbf{a},\sigma}$ the MGU for $\mathbf{a}$ and $head(\sigma)$, *(ii)* if the term at position π in $\mathbf{a}$ is either a constant of Γ, or a bound variable in q that occurs in some atom of $body(q)$ other than $\mathbf{a}$, then the variable at position π in $head(\sigma)$ occurs also in $body(\sigma)$, and *(iii)* if a bound variable in q occurs only in $\mathbf{a}$ at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 2$, then either the variable at position π_i in $head(\sigma)$, for each $i \in \{1, \ldots, m\}$, occurs also in $body(\sigma)$, or at positions $\pi_1, \ldots, \pi_m$ in $head(\sigma)$ we have the same existentially quantified variable.

The Algorithm rewrite. We are now ready to present the algorithm rewrite, shown in Figure 1. The rewriting of a BCQ is computed by exhaustively applying two steps: *minimization* and *rewriting*, corresponding to steps (a) and (b) of the algorithm. In what follows we give an informal description of these two steps.

Minimization Step. If there exists a BCQ $q \in Q_r$ such that $body(q)$ contains two atoms $\mathbf{a}$ and $\mathbf{b}$ that unify, then the algorithm computes the BCQ q' by

```
ALGORITHM rewrite
INPUT: Schema $\mathcal{R}$, set $\Sigma$ of STGDs over $\mathcal{R}$, BCQ $q$ over $\mathcal{R}$
OUTPUT: Rewritten query $Q_r$ over $\mathcal{R}$

  $Q_r := \{q\}$; $Q_r^{\mathsf{can}} := \varnothing$; $i := 0$; $Vars := var(q)$;
  repeat
    $Q' := Q_r$; $Q'' := Q_r^{\mathsf{can}}$;
    for each $q \in Q'$ do
    (a) for each $\mathbf{a}, \mathbf{b} \in body(q)$ do
          if $\mathbf{a}$ and $\mathbf{b}$ unify then
            $q' := \overline{\gamma_{\mathbf{a},\mathbf{b}}}(q)$
            $Q_r := Q_r \cup \{q'\}$
            $Q_r^{\mathsf{can}} := Q_r^{\mathsf{can}} \cup \tau(q')$;
    (b) for each $\mathbf{a} \in body(q)$ do
            for each $\sigma \in \Sigma$ do
              if $\sigma$ is applicable to $\mathbf{a}$ then
                $i := i + 1$
                $\rho :=$ rename $(\gamma_{\mathbf{a},\sigma^i}, Vars)$
                $q' := \rho\left(\gamma_{\mathbf{a},\sigma^i}\left(q[\mathbf{a}/body(\sigma^i)]\right)\right)$
                $Q_r := Q_r \cup \{q'\}$
                $Q_r^{\mathsf{can}} := Q_r^{\mathsf{can}} \cup \tau(q')$;
  until $Q'' = Q_r^{\mathsf{can}}$;
  return $Q_r$;
```

Fig. 1. The Algorithm rewrite.

applying the particular MGU $\overline{\gamma_{\mathbf{a},\mathbf{b}}}$ for $\mathbf{a}$ and $\mathbf{b}$ on q, where $\overline{\gamma_{\mathbf{a},\mathbf{b}}}$ is obtained as follows. Let $\mathbf{a} = r(V_1, \ldots, V_n)$ and $\mathbf{b} = r(W_1, \ldots, W_n)$. Construct the atoms $\mathbf{a}'$ and $\mathbf{b}'$ from $\mathbf{a}$ and $\mathbf{b}$ as follows: for an arbitrary $k \in \{1, \ldots, n\}$, *(i)* if both $V_k, W_k \notin Vars$, then replace W_k with V_k, and *(ii)* if $V_k \in Vars$ and $W_k \notin Vars$ (resp., $V_k \notin Vars$ and $W_k \in Vars$), then replace W_k with V_k (resp., V_k with W_k). Recall that $Vars$ is the set of variables of the given query. We define $\overline{\gamma_{\mathbf{a},\mathbf{b}}} = \gamma_{\mathbf{a}',\mathbf{b}'}$. $\overline{\gamma_{\mathbf{a},\mathbf{b}}}$ guarantees that any new variables, introduced during the rewriting process, that appear in q' are unbound; its existence follows from the fact that the TGDs are sticky. The canonical form of q' is computed by applying the transformation τ, and then added to Q_r^{can} which represents the canonical form of Q_r. In particular, τ replaces the unbound variables with the special term "$\star$".

REWRITING STEP. During the i-th application of the rewriting step, if there exists a TGD σ and a BCQ $q \in Q_r$ containing an atom $\mathbf{a}$ such that σ is applicable to $\mathbf{a}$, then the algorithm computes the BCQ $q' = \rho\left(\gamma_{\mathbf{a},\sigma^i}\left(q[\mathbf{a}/body(\sigma^i)]\right)\right)$, that is, the BCQ obtained from q by replacing $\mathbf{a}$ with $body(\sigma^i)$, where σ^i is obtained by replacing each variable V that occurs in σ with V^i, then applying the MGU $\gamma_{\mathbf{a},\sigma^i}$ for $\mathbf{a}$ and $head(\sigma^i)$, and finally applying the renaming substitution ρ, where ρ is constructed from $\gamma_{\mathbf{a},\sigma^i}$ according to the procedure rename (see Figure 2). By considering σ^i (instead of σ) actually we rename, using the integer i, the variables of the TGD σ which is applicable to $\mathbf{a}$. This is needed to ensure that the set of variables that appear in the rewritten query, and the set of variables that occur in TGDs are disjoint, and also to avoid undesirable clutters between new variables introduced during different applications of the rewriting step. The application of the renaming substitution ρ is needed to guarantee that any new variables in q' are unbound. Finally, the canonical form of q' is added to Q_r^{can}.

```
PROCEDURE rename
INPUT: Substitution $\gamma$, set $Vars$ of variables
OUTPUT: Renaming substitution $\rho$

    $\mu := \varnothing$;
    for each assertion $V \to U \in \gamma$ do
        if $V \in Vars$ and $U \notin Vars$ then
            if there is no assertion $V' \to U$ in $\mu$ then
                $\mu := \mu \cup \{V \to U\}$;
    $\rho := \mu^{-1}$;
    return $\rho$;
```

Fig. 2. The Procedure rename.

Example 3. Let $\mathcal{R} = \{p, r, s\}$. Consider the set Σ of STGDs over $\mathcal{R}$ constituted by the TGDs $\sigma_1 = r(X,Y) \to \exists Z\, s(X,Z,Z)$ and $\sigma_2 = s(X,Y,Z) \to p(X,Z,Z)$. Consider also the BCQ $q_0 = \exists A \exists B \exists C\, p(A,B,C), s(A,B,B)$ over $\mathcal{R}$. Clearly, σ_2 is applicable to $\mathbf{a} = p(A,B,C) \in body(q_0)$. During the first application of the rewriting step we get the BCQ $q_1 = \rho(\gamma_{\mathbf{a},\sigma_2^1}(q_0[\mathbf{a}/body(\sigma_2^1)]))$, where $\gamma_{\mathbf{a},\sigma_2^1} = \{X^1 \to A, B \to Z^1, C \to Z^1\}$, and $\rho = \{Z^1 \to B\}$. Hence, $q_1 = \exists A \exists B \exists Y^1\, s(A,Y^1,B), s(A,B,B)$; note that Y^1 is a new variable. The canonical form of q_1 is obtained by replacing the unbound variable Y^1 with the special term "$\star$". Observe now that the atoms $s(A,Y^1,B)$ and $s(A,B,B)$ in the body of q_1 unify. Hence, the minimization step is applied and we get the BCQ $q_2 = \exists A \exists B\, s(A,B,B)\}$; note that $\tau(q_2) = \exists B\, s(\star,B,B)$. Finally, observe that σ_1 is applicable to the atom $\mathbf{a} = s(A,B,B) \in body(q_2)$. During the second application of the rewriting step we get the BCQ $q_3 = \rho'(\gamma_{\mathbf{a},\sigma_1^2}(q_2[\mathbf{a}/body(\sigma_1^2)]))$, where $\gamma_{\mathbf{a},\sigma_1^2} = \{X^2 \to A, Z^2 \to B\}$, and $\rho' = \varnothing$. Hence, $q_3 = \exists A \exists Y^2\, r(A,Y^2)\}$; clearly, $\tau(q_3) = r(\star,\star)$. ■

Soundness and Completeness. In what follows we show that the algorithm rewrite produces a so-called *perfect rewriting*, i.e., a rewritten query that produces the correct answers under STGDs when evaluated over the given database. To this aim we need the notion of *rewrite graph.*

Definition 3. Consider a set Σ of STGDs expressed over a schema $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. The *rewrite graph* of rewrite$(\mathcal{R},\Sigma,q)$, denoted as $RG(\mathcal{R},\Sigma,q)$, is a triple $\langle V,E,\lambda\rangle$, where V is the node set, E is the edge set, and λ is a labeling function $V \to$ rewrite$(\mathcal{R},\Sigma,q)$. For the BCQ q add a node v in V, and let $\lambda(v) = q$. The rewrite graph is constructed inductively as follows. Suppose that the minimization step is applied because the atoms $\mathbf{a}, \mathbf{b} \in body(\lambda(v))$ unify, where $v \in V$. Add in V a node u and let $\lambda(u) = q'$, where q' is the constructed BCQ, and if there exists a parent node of v, denoted as $par(v)$, then add in E an arc $par(v) \frown u$ (if u is not already in the graph). Now, suppose that the rewriting step is applied because the STGD σ is applicable to $\mathbf{a}$, where $\sigma \in \Sigma$ and $\mathbf{a} \in body(\lambda(v))$, for some node $v \in V$. Add in V a node u and let $\lambda(u) = q'$, where q' is the constructed BCQ, and also add in E an arc $v \frown u$ (if u is not already in the graph). The *depth* of a node v of $RG(\mathcal{R},\Sigma,q)$, written as $depth(v)$, is the length of the (unique) path from a node with zero *in-degree*, i.e., a node with no incoming edges, to v; a node with zero in-degree has depth zero.

Observe that the rewrite graph $RG(\mathcal{R}, \Sigma, q)$, as defined above, is a forest comprised by at most $|body(q)|$ disjoint *rooted trees*, i.e., trees in which one node has been designated as the root node. Each rooted tree has a root node v (which may be the only node in the tree) such that $\lambda(v)$ is a BCQ obtained by applying the minimization step on some query $\lambda(u)$, where u is a node with zero depth. Intuitively, the existence of a vertex v with depth $d \geqslant 0$ implies that the BCQ $\lambda(v)$ was obtained during the rewriting process starting from q and applying either the rewriting step or the minimization step at least d times.

We continue to establish two useful technical results.

Lemma 1. *Let $\mathcal{R}$ be a relational schema. Consider a database D for $\mathcal{R}$, a set Σ of STGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. Let $RG(\mathcal{R}, \Sigma, q) = \langle V, E, \lambda \rangle$. If $chase^{[i]}(D, \Sigma) \models \lambda(v)$, for $i \geqslant 0$ and $v \in V$, then there exists $j \geqslant i$ such that $chase^{[j]}(D, \Sigma) \models q$.*

Proof (sketch). By induction on the depth $d \geqslant 0$ of node v. □

Lemma 2. *Let $\mathcal{R}$ be a relational schema. Consider a database D for $\mathcal{R}$, a set Σ of STGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. Let $RG(\mathcal{R}, \Sigma, q) = \langle V, E, \lambda \rangle$. If $chase^{[i]}(D, \Sigma) \models \lambda(v)$, for $i \geqslant 0$ and $v \in V$, then there exists $v' \in V$ with $depth(v') \geqslant depth(v)$ such that $D \models \lambda(v')$.*

Proof (sketch). By induction on the number of applications of the chase rule. □

We are now ready, by using the above two technical lemmas, to establish that the algorithm rewrite produces a perfect rewriting, i.e., is sound and complete.

Theorem 1. *Let $\mathcal{R}$ be a relational schema. Consider a database D for $\mathcal{R}$, a set Σ of STGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. Then, $D \cup \Sigma \models q$ iff $D \models$ rewrite$(\mathcal{R}, \Sigma, q)$.*

Proof. For notational convenience, let $Q_r = \mathsf{rewrite}(\mathcal{R}, \Sigma, q)$. The claim is equivalent to $D \models Q_r$ iff $chase(D, \Sigma) \models q$. By hypothesis, there exists a BCQ $p \in Q_r$ such that $D \models p$, or, equivalently, $chase^{[0]}(D, \Sigma) \models p$. Let $RG(\mathcal{R}, \Sigma, q) = \langle V, E, \lambda \rangle$ be the rewrite graph of Q_r, and suppose that $p = \lambda(v)$, for some $v \in V$. By Lemma 1 we get that there exists $i \geqslant 0$ such that $chase^{[i]}(D, \Sigma) \models q$, which implies that $chase(D, \Sigma) \models q$, as needed. Conversely, there exists a homomorphism h such that $h(body(q)) \subseteq chase(D, \Sigma)$. Since $h(body(q))$ is finite it follows that there exists $i \geqslant 0$ such that $h(body(q)) \subseteq chase^{[i]}(D, \Sigma)$, or, equivalently, $chase^{[i]}(D, \Sigma) \models q$. Suppose that $q = \lambda(v)$, where $v \in V$. By definition of the rewrite graph, $depth(v) = 0$. By Lemma 2 we get that there exists $u \in V$ with $depth(u) \geqslant 0$ such that $D \models \lambda(u)$. Since $\lambda(u) \in Q_r$ we conclude that $D \models Q_r$. □

Termination. We finally establish termination of the algorithm rewrite under STGDs. The following lemma shows that new variables, introduced during the rewriting process, occur at most once in the BCQs constructed during the rewriting process.

Lemma 3. *Let $\mathcal{R}$ be a relational schema. Consider a set Σ of STGDs over $\mathcal{R}$, a BCQ q over $\mathcal{R}$, and a BCQ $q' \in \mathsf{rewrite}(\mathcal{R}, \Sigma, q)$. Each variable in $var(q') \setminus var(q)$ is unbound in q'.*

Proof (sketch). For notational convenience, let $Q_r = \mathsf{rewrite}(\mathcal{R}, \Sigma, q)$. We denote by Q_r^i the initial segment of Q_r obtained starting from q and applying $i \geqslant 0$ times either the minimization step or the rewriting step of the algorithm rewrite. The proof is by induction on i. Note that the use of the particular MGU (instead of applying an arbitrary one) during the minimization step of the algorithm rewrite, and also the application of the renaming substitution (see Figure 2) during the rewriting step are crucial. □

We are now ready, by exploiting Lemma 3, to establish termination of the algorithm rewrite under STGDs.

Theorem 2. *Let $\mathcal{R}$ be a relational schema. Consider a set Σ of STGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. The algorithm rewrite with input $\mathcal{R}$, Σ and q terminates.*

Proof. It suffices to show that the maximum number of BCQs that can appear in the canonical form of the rewritten query $\mathsf{rewrite}(\mathcal{R}, \Sigma, q)$, denoted as Q_c, is finite. By Lemma 3, if a new variable, introduced during the rewriting process, occurs in a BCQ $p \in \mathsf{rewrite}(\mathcal{R}, \Sigma, q)$, then is unbound in p. Thus, by definition of τ, none of these new variables can appear in Q_c since they are replaced with the special term "$\star$". Hence, the set of terms used to construct Q_c corresponds to the set of variables and constants occurring in the BCQ q plus the special term "$\star$"; thus, is finite. Moreover, only predicates of $\mathcal{R}$ can appear in Q_c which is also finite. The claim follows since the number of atoms that can be constructed using a finite set of terms and a finite set of predicates is finite. □

By combining Theorems 1 and 2 we immediately get the main complexity results on STGDs (first stated in [5], where many more results are shown).

Corollary 1 ([5]). *The class of STGDs is FO-rewritable.*

Since answering first-order queries is in the class AC_0 in data complexity [24], we immediately get the following result.

Corollary 2 ([5]). *BCQ answering under STGDs is in AC_0 in data complexity.*

Acknowledgments. The research leading to these results has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013)/ERC grant agreement no. 246858 – DIADEM. The authors also acknowledge support by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). Georg Gottlob's work was also supported by a Royal Society Wolfson Research Merit Award.

References

1. C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP*, pages 73–85, 1981.
2. A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of ER*, pages 270–284, 2001.
3. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, pages 70–80, 2008.
4. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, pages 77–86, 2009.
5. A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological query answering. Submitted for publication, 2010.
6. A. Calì and M. Kifer. Containment of conjunctive object meta-queries. In *Proc. of VLDB*, pages 942–952, 2006.
7. A. Calì, D. Lembo, and R. Rosati. Decidability and complexity of query answering over incosistent and incomplete databases. In *Proc. of PODS*, pages 260–271, 2003.
8. A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI*, pages 16–21, 2003.
9. A. Calì and D. Martinenghi. Querying incomplete data over extended er schemata. *TPLP*, 2010. to appear.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
11. D. Calvanese, G. D. Giacomo, and M. Lenzerini. Conjunctive query containment and answering under description logic constraints. *ACM Trans. Comput. Log.*, 9(3), 2008.
12. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOCS*, pages 77–90, 1977.
13. A. Deutsch. *XML query reformulation over mixed and redundant storage.* PhD thesis, Dept. of Computer and Information Sciences, Univ. of Pennsylvania, 2002.
14. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisisted. In *Proc. of PODS*, pages 149–158, 2008.
15. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
16. M. E. Goncalves and E. Grädel. Decidability issues for action guarded logics. In *Description Logics*, pages 123–132, 2000.
17. D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.
18. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843, 1995.
19. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
20. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.
21. H. Pérez-Urbina, I. Horrocks, and B. Motik. Practical aspects of query rewriting for owl 2. In *Proc. of OWLED*, 2009.
22. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.
23. M. Simkus and T. Eiter. DNC: Decidable non-monotonic disjunctive logic programs with function symbols. In *Proc. of LPAR*, pages 514–530, 2007.
24. M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS*, pages 266–276, 1995.