Ontological Reasoning with F-Logic Lite and Its Extensions

Andrea Calì[2,1,4], Georg Gottlob[1,2], Michael Kifer[3], Thomas Lukasiewicz[1] 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 Computer Science, Stony Brook University, USA
[4]Department of Information Systems and Computing, Brunel University, UK

{andrea.cali,georg.gottlob,thomas.lukasiewicz,andreas.pieris}@comlab.ox.ac.uk
kifer@cs.sunysb.edu

Introduction

Answering queries posed over knowledge bases is a central problem in knowledge representation and database theory. In the database area, checking query containment is an important query optimization and schema integration technique (Aho, Sagiv, and Ullman 1979; Johnson and Klug 1984). In knowledge representation it has been used for object classification, schema integration, service discovery, and more. In the presence of a knowledge base, the problem of query containment is strictly related to that of query answering; indeed, the two are reducible to each other (Calì, Gottlob, and Kifer 2008b); we focus on the latter, and our results immediately extend to the former. A practically relevant instance of the query containment problem was first studied in (Johnson and Klug 1984) for functional and inclusion dependencies, and later, for instance, in (Calvanese, Giacomo, and Lenzerini 1998). *DL-lite* ontologies (Poggi et al. 2008) have gained importance in the area of the Semantic Web, since they provide tractable query answering while keeping high flexibility and expressiveness. Rather than focusing on specific logical theories, we analyze the fundamental difficulty that underlies earlier approaches, such as (Johnson and Klug 1984; Calì and Kifer 2006; Calì, Lembo, and Rosati 2003). All such works considered special classes of so-called *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)* (a generalization of key dependencies and functional dependencies), and used the technique called *chase*. The chase (Maier, Mendelzon, and Sagiv 1979) is a well-known procedure of enforcing the validity of a set of TGDs, successfully applied in database theory, data exchange (Fagin et al. 2005), and terminological reasoning (Calvanese et al. 2007). Several authors have studied query evaluation problems for settings where the chase always terminates and thus produces a finite solution (Fagin et al. 2005). To this aim, restrictions on TGDs, such as *weak acyclicity* (Deutsch 2002), have been defined. In cases where the chase is possibly infinite, query answering (and containment) may become undecidable (Chandra and Vardi 1985; Mitchell 1983; Johnson and Klug 1984; Calì, Lembo, and Rosati 2003). Suitable restrictions on TGDs are then needed to make the problem decidable.

Our approach significantly generalizes the above ones. We carve out a very large class of constraints for which the infinite chase can be tamed. The class of constraints we deal with is an extension of Datalog rules; such rules are known as *tuple-generating dependencies (TGDs)* in the database literature (Beeri and Vardi 1984). A TGD is an implication between two conjunction of atoms (called *body* and *head*, where the head is implied by the body), where some of the variables in the head can be existentially quantified. TGDs are successfully employed in expressing database constraints. For example, the fact that each employee works in some department can be expressed by the TGD:

$$\forall E \, \textit{employee}(E) \rightarrow \exists D \, \textit{works\_in}(E, D).$$

Note that the above TGD is actually an inclusion dependency (Johnson and Klug 1984). The fact that each department is specialized in the areas of the projects which runs can be expressed by the TGD:

$$\forall D \forall P \forall A \, \textit{runs}(D, P), \textit{in\_area}(P, A) \rightarrow \textit{dep}(D, A)$$

The aim of the present paper is to define significantly larger classes of TGDs than those studied in the literature, especially in cases where the chase is not guaranteed to terminate; indeed, the methods adopted to find our results hinge on the notion of chase. In particular, inspired by guarded logics (Goncalves and Grädel 2000), we define the notions of sets of *linear TGDs (LTGDs)*, *guarded TGDs (GTGDs)*, and *weakly guarded TGDs (WGTGDs)*, which form the languages called *linear Datalog*$^{\pm}$, *guarded Datalog*$^{\pm}$, and *weakly-guarded Datalog*$^{\pm}$, respectively, in the Datalog$^{\pm}$ family. Decidability of query answering under such classes of rules follows from the fact that the chase under WGTGDs has bounded treewidth, together with the well-known results about the generalized tree-model property (Courcelle 1990; Goncalves and Grädel 2000); however, this result does not help us in deriving useful complexity bounds. Our main contribution lies in the complexity bounds, summarized in Figure 1, for query evaluation under linear, guarded, and weakly-guarded Datalog$^{\pm}$. They regard answering for *Boolean conjunctive queries (BCQs)*, which is equivalent to answering for conjunctive queries. Notice that one cannot directly or easily use known results on guarded

BCQ type	LTGDs	GTGDs	WGTGDs
general	PSPACE	2EXPTIME	2EXPTIME
bounded width, fixed, atomic	PSPACE	2EXPTIME	2EXPTIME

Complexity results for variable TGDs.

BCQ type	LTGDs	GTGDs	WGTGDs
general	NP	NP	EXPTIME
bounded width, fixed, atomic	in AC_0	PTIME	EXPTIME

Complexity results for fixed TGDs.

Figure 1: Summary of complexity results.

logics (Goncalves and Grädel 2000) to derive complexity results for query evaluation, since queries are in general non-guarded. All complexity bounds (except the AC_0 one) are tight. By "bounded width" we mean bounded treewidth or even hypertree width (Gottlob, Leone, and Scarcello 2002). Notice that complexity in the case of fixed queries and fixed TGDs is the so-called *data complexity*, i.e., the complexity wrt the data only, which is of particular interest in database applications.

We then introduce *key dependencies (KDs)* (which are a special case of EGDs) together with TGDs, showing cases where the addition of KDs does not increase the complexity of query answering; such cases are characterized by the notion of *separability* between KDs and TGDs. A syntactic, sufficient criterion for separability is defined by means of the notion of *non-conflicting* KDs (Calì, Gottlob, and Lukasiewicz 2009). When KDs are non-conflicting, they can be ignored (apart from an initial check), and therefore the complexity remains the same as in the case of the TGDs alone. Therefore, all the results in Figure 1 also hold in the presence of non-conflicting KDs.

Applications. As a first application of Datalog$^{\pm}$, we consider F-logic (Kifer, Lausen, and Wu 1995), a formalism for object-oriented deductive databases. For a smaller but still powerful version of F-logic, called F-logic Lite (Calì and Kifer 2006), we show how to encode F-logic Lite using TGDs and a single EGD, which is non-conflicting. The results of this paper apply to this case, since the TGDs used in F-logic Lite are WGTGDs. In addition, our results cover a much larger set of F-logic queries than (Calì and Kifer 2006). Also, we prove that query answering under F-logic Lite can be decided in NP and combining this with a hardness result (reduction from 3-COLORABILITY), we prove that query answering (and query containment) in F-logic Lite is NP-complete. We show that the upper complexity bound for query answering under F-logic Lite can be obtained as a special case of WGTGDs, characterized by a semantic condition called *polynomial cloud criterion (PCC)* (Calì, Gottlob, and Kifer 2008b). As a second application of Datalog$^{\pm}$, we consider a class of LTGDs and KDs that encodes an extended version of the *Entity-Relationship (ER)* (Chen 1976) formalism that we call EER. We call such class of KDs and TGDs *conceptual dependencies (CDs)* (Calì, Gottlob, and Pieris 2009). This class is certainly relevant in data modeling, given the wide adoption of the ER formalism in the industry. Interestingly, conceptual dependencies are in general not non-conflicting, and therefore not separable. However, we give a necessary and sufficient syntactic condition that precisely characterizes separable classes of CDs. For separable CDs, then, the complexity of BCQ answering is the same as for linear Datalog$^{\pm}$.

Our results handily subsume both the main decidability and complexity results in (Johnson and Klug 1984), as well as the decidability and complexity results on F-logic lite (Calì and Kifer 2006) as special cases. Moreover, both the results on EER and those on linear Datalog$^{\pm}$, with the addition of *negative constraints* (Calì, Gottlob, and Lukasiewicz 2009), properly generalize the languages in the DL-lite family (Poggi et al. 2008).

The results in this paper appeared in (Calì, Gottlob, and Kifer 2008b) (see also the extended version (Calì, Gottlob, and Kifer 2008a)), (Calì, Gottlob, and Lukasiewicz 2009), and (Calì, Gottlob, and Pieris 2009).

Formal Definitions

We introduce the following pairwise disjoint sets of symbols: *(i)* an infinite set Δ of *constants*, which constitute the "normal" domain of a database; *(ii)* an infinite set Δ_N of *labeled nulls*, which will be used as "fresh" Skolem terms. Intuitively, a null is a placeholder for an unknown value. Different nulls may represent the same value; therefore, nulls can be seen as variables. However, different constants represent different values (*unique name assumption*). We assume a lexicographic order on $\Delta \cup \Delta_N$, such that every symbol in Δ_N follows all symbols in Δ. Sets of variables (or sequences, with a slight abuse of notation) are denoted as $\bar{X}$, with $\bar{X} = X_1, \ldots, X_k$, for some $k > 0$.

We will refer to a relational schema $\mathcal{R}$, assuming that database instances (also called databases), queries and dependencies use predicates in $\mathcal{R}$. We assume the reader is familiar with the relational model. As mentioned, we denote relational schemas by $\mathcal{R}$, queries by Q, database instances by D, and the result of evaluating a query Q on a database instance D by $Q(D)$. In the following, we shall consider *conjunctive queries (CQs)*, with which we assume the reader is familiar. Boolean CQs (BCQs) are those with no variables in the head (i.e., with arity zero). We denote as $vars(Q)$ the set of variables appearing in a CQ Q. Database instances will be constructed with values from $\Delta \cup \Delta_N$, and they will be possibly infinite.

By an *atom* we mean an atomic formula of the form $r(t_1, \ldots, t_n)$, where r is an n-ary predicate (also called relation name). The constants and labeled nulls appearing in an atom $\underline{a}$ are denoted by $dom(\underline{a})$. This notation extends to sets and conjunctions of atoms. A *position* $r[i]$ in a relational schema is identified by a relational predicate r and its i-th attribute, identified by the integer i.

We assume the reader is familiar with the notion of a *homomorphism*—a mapping of symbols that is an identity on Δ, and transforms atoms of a set into atoms of another set. We refer the reader, for instance, to (Calì, Gottlob, and Kifer 2008b) for the details.

A major issue in this work are database dependencies, which are defined over a relational schema. In the relational

model, one of the most important classes of dependencies are *tuple-generating dependencies (TGDs)*, which are a generalization of inclusion dependencies.

Definition 1 *Given a relational schema $\mathcal{R}$, a TGD σ is a first-order formula of the form $\forall\bar{X}\forall\bar{Y}\,\varphi(\bar{X},\bar{Y}) \rightarrow \exists\bar{Z}\,\psi(\bar{X},\bar{Z})$, where $\varphi(\bar{X},\bar{Y})$ and $\psi(\bar{X},\bar{Z})$ are conjunctions of atoms over $\mathcal{R}$, called* body *and* head *of the TGD, and denoted $body(\sigma)$ and $head(\sigma)$, respectively. Such a dependency is satisfied by a database D for $\mathcal{R}$ if, whenever there is a homomorphism h that maps the atoms of $\varphi(\bar{X},\bar{Y})$ to atoms of D, there exists an extension h' of h (i.e., $h' \supseteq h$) that maps the atoms of $\psi(\bar{X},\bar{Z})$ to atoms of D.*

Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs.

We now define the notion of *query answering* under TGDs. A similar notion is used in data exchange (Fagin et al. 2005; Gottlob and Nash 2008) and in query answering over incomplete data (Calì, Lembo, and Rosati 2003). Given a database D and a set Σ of TGDs, we first define the set of instances B, such that $B \models D \cup \Sigma$, as the set of *solutions* of D given Σ, denoted $sol(\Sigma, D)$. The *answers* to a CQ Q on D given Σ, denoted $ans(Q, \Sigma, D)$, is the set of ground atoms $\underline{a}$ such that for every $B \in sol(\Sigma, D)$, it holds that $\underline{a} \in Q(B)$. For a BCQ, if $ans(Q, \Sigma, D) \neq \varnothing$, then we write $D \cup \Sigma \models Q$; otherwise, we write $D \cup \Sigma \not\models Q$.

The *chase* process was introduced in order to enable checking implication of dependencies (Maier, Mendelzon, and Sagiv 1979), and later also for checking query containment (Johnson and Klug 1984). Informally, chase is a process of repairing a database with respect to a set of database dependencies by adding tuples that may contain labeled nulls to denote unknown values. We will use the term "chase" both for the chase procedure and for its output. We do not describe the chase in detail here, and we refer the interested reader, for instance, to (Calì, Gottlob, and Kifer 2008b; Deutsch, Nash, and Remmel 2008). The (possibly infinite) chase of a database D w.r.t. a set Σ of TGDs, denoted as $chase(\Sigma, D)$, is a *universal solution* of D given Σ, i.e., for each database $B \in sol(\Sigma, D)$, there exists a homomorphism from $chase(\Sigma, D)$ to B (Fagin et al. 2005; Deutsch, Nash, and Remmel 2008). Using this fact it can be shown that for a BCQ Q, $D \cup \Sigma \models Q$ iff $chase(\Sigma, D) \models Q$.

Containment of relational queries has long been considered a fundamental problem in query optimization—especially query containment under constraints such as TGDs. CQ containment and CQ answering are mutually PTIME-reducible. Moreover, BCQ answering is LOGSPACE-equivalent to CQ answering. We shall henceforth consider BCQ answering only. Finally, it can be shown that every set Σ of TGDs can be transformed in LOGSPACE in another set Σ' of TGDs *with single-atom head*, such that for every BCQ Q and for every instance D it holds that $D \cup \Sigma \models Q$ iff $D \cup \Sigma' \models Q$.

F-logic Lite

In this section we present F-logic Lite and then show that query answering and containment under this formalism are NP-complete (Calì and Kifer 2006).

F-logic Lite is a small but expressive subset of F-logic, a well-known formalism introduced for object-oriented deductive databases. Roughly speaking, F-logic Lite omits negation and default inheritance, and allows only a limited form of cardinality constraints.

To avoid a lengthy introduction, we do not use the standard F-logic notation and instead represent F-logic Lite using the following predicates:

- $\mathsf{member}(O, C)$: object O is a member of class C.
- $\mathsf{sub}(C_1, C_2)$: class C_1 is a subclass of class C_1.
- $\mathsf{data}(O, A, V)$: attribute A has value V on object O.
- $\mathsf{type}(O, A, T)$: attribute A has type T for object O (recall that in F-logic classes are also objects).
- $\mathsf{mandatory}(A, O)$: attribute A is mandatory for object (class) O, i.e., it must have at least one value for O.
- $\mathsf{funct}(A, O)$: A is a functional attribute for the object (class) O, i.e., it can have at most one value for O.

These predicates are related to each other by the following twelve deductive rules, which we denote as Σ_{FLL}.

ρ_1: $\mathsf{member}(V, T) \leftarrow \mathsf{type}(O, A, T), \mathsf{data}(O, A, V)$.
ρ_2: $\mathsf{sub}(C_1, C_2) \leftarrow \mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2)$.
ρ_3: $\mathsf{member}(O, C_1) \leftarrow \mathsf{member}(O, C), \mathsf{sub}(C, C_1)$.
ρ_4: $V = W \leftarrow \mathsf{data}(O, A, V), \mathsf{data}(O, A, W), \mathsf{funct}(A, O)$.
Note that this is the only EGD in this axiomatization.
ρ_5: $\mathsf{data}(O, A, V) \leftarrow \mathsf{mandatory}(A, O)$.
Note that this is a TGD with an existentially quantified variable in the head (variable V; quantifiers are omitted).
ρ_6: $\mathsf{type}(O, A, T) \leftarrow \mathsf{member}(O, C), \mathsf{type}(C, A, T)$.
ρ_7: $\mathsf{type}(C, A, T) \leftarrow \mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T)$.
ρ_8: $\mathsf{type}(C, A, T) \leftarrow \mathsf{type}(C, A, T_1),\ \mathsf{sub}(T_1, T)$.
ρ_9: $\mathsf{mandatory}(A, C) \leftarrow \mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1)$.
ρ_{10}: $\mathsf{mandatory}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{mandatory}(A, C)$.
ρ_{11}: $\mathsf{funct}(A, C) \leftarrow \mathsf{sub}(C, C_1),\ \mathsf{funct}(A, C_1)$.
ρ_{12}: $\mathsf{funct}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{funct}(A, C)$.

It can be shown that the only EGD in the above rules, that is, rule ρ_4, does not actually interact with the TGDs, and therefore the chase can ignore it. More precisely, the above set Σ_{FLL} can be transformed into an equivalent one with a single EGD that syntactically falls into the class of *non-conflicting* keys w.r.t. the TGDs in Σ_{FLL} (Calì, Gottlob, and Lukasiewicz 2009). For more details on non-conflicting keys see the section where we consider key dependencies.

By a reduction from the 3-COLORABILITY problem, we can show that conjunctive query answering under F-logic Lite rules is NP-hard. For the upper bound, we can use a technique of (Calì and Kifer 2006) to show that every query can be answered by looking only at a finite segment of the chase of polynomial size in Σ_{FLL}. Hence conjunctive query answering under F-logic Lite is in NP.

Theorem 2 *BCQ answering under F-logic Lite rules is* NP*-complete.*

(Weakly) Guarded Datalog$^{\pm}$

This section introduces the classes of *guarded* and *weakly guarded* TGDs, also called *guarded* and *weakly guarded Datalog$^{\pm}$*, respectively, which have a number of useful properties.

We first give the notion of an *affected* position of a relational schema w.r.t. a set of TGDs. Given a relational schema $\mathcal{R}$ and a set of TGDs Σ over $\mathcal{R}$, an *affected position* in $\mathcal{R}$ w.r.t. Σ is defined inductively as follows (here we use lowercase Greek letters to denote positions). Let π_h be a position in the head of a TGD σ in Σ. *(a)* If an existentially quantified variable appears in π_h, then π_h is affected w.r.t. Σ. *(b)* If the same universally quantified variable X appears both in position π_h, and in the body of σ in affected positions *only*, then π_h is affected w.r.t. Σ.

Example 1 Consider the following set of TGDs:

$$\begin{array}{lrcl} \sigma_1: & r_1(X,Y), r_2(X,Y) & \rightarrow & \exists Z\, r_2(Y,Z) \\ \sigma_2: & r_2(X,Y), r_2(W,X) & \rightarrow & r_1(Y,X) \end{array}$$

Notice that $r_2[2]$ is affected since the variable Z in σ_1 is existentially quantified. Considering again σ_1, the variable Y appears in $r_2[2]$ but also in $r_1[2]$, therefore it does not make the position $r_2[1]$ affected. In σ_2, X appears in the affected position $r_2[2]$ but also in $r_2[1]$, which is not affected; therefore, it does not make the position $r_1[2]$ affected. On the other hand, in σ_2, Y appears in $r_2[2]$ and nowhere else, thus $r_1[1]$ is affected. ∎

Definition 3 *Given a TGD σ of the form $\varphi(\bar{X},\bar{Y}) \rightarrow \exists \bar{Z}\, \psi(\bar{X},\bar{Z})$, we say that σ is a* (fully) guarded TGD (GTGD) *if there exists an atom in $body(\sigma)$, called a* guard*, that contains all the universally quantified variables of σ, i.e., all the variables $\bar{X},\bar{Y}$ that occur in $\varphi(\bar{X},\bar{Y})$.*

Example 2 The TGD $r_1(X,Y,Z), r_2(Y), r_3(X,Z) \rightarrow \exists W\, r_4(W,X)$ is guarded; in particular, the guard is the atom $r_1(X,Y,Z)$, since it contains all the universally quantified variables of the TGD. ∎

Definition 4 *Given a TGD σ of the form $\varphi(\bar{X},\bar{Y}) \rightarrow \exists \bar{Z}\, \psi(\bar{X},\bar{Z})$, belonging to a set of TGDs Σ over a schema $\mathcal{R}$, we say that σ is a* weakly guarded TGD (WGTGD) *w.r.t. Σ if there exists an atom in $body(\sigma)$, called a* weak guard*, that contains all the universally quantified variables of σ that appear* only *in positions that are affected w.r.t. Σ.*

Example 3 Consider the two TGDs in Example 1. In σ_1 both atoms are guards (and obviously weak guards), since they contain all universally quantified variables in the TGD. In σ_2, Y is the sole variable that appears in affected positions only. Therefore, $r_2(X,Y)$ is a weak guard. ∎

It is important to keep in mind that the transformation to singleton-atom head rules, mentioned in the preliminaries section, preserves the guardedness and weak guardedness properties. Therefore, we can assume that TGDs have singleton-atom heads.

Decidability. The following theorem, proved by reduction from a Turing machine, shows that a *single* unguarded rule can destroy the decidability of basic reasoning tasks under TGDs.

Theorem 5 *There is a fixed set of TGDs Σ_u, and a fixed atomic BCQ Q, such that all but one TGDs of Σ_u are guarded and it is undecidable whether, for a database D, $D \cup \Sigma_u \models Q$ (equivalently, whether $chase(\Sigma_u, D) \models Q$).*

Theorem 6 *Given a relational schema $\mathcal{R}$, a set of WGTGDs Σ, a BCQ Q, and a database instance D for $\mathcal{R}$, the problem whether $D \cup \Sigma \models Q$ (equivalently, whether $chase(\Sigma, D) \models Q$) is decidable.*

This theorem establishes decidability of query answering under WGTGDs, but it tells little about the complexity. Let us then focus on complexity issues.

Complexity. The following theorem characterizes the complexity of reasoning under WGTGDs.

Theorem 7 *Let Σ be a set of WGTGDs, let D be an instance, and let Q be a BCQ. Determining whether $D \cup \Sigma \models Q$ (equivalently, whether $chase(\Sigma, D) \models Q$) is* EXPTIME*-complete in case of bounded predicate arities, and even in case Σ is fixed; it is* 2EXPTIME *complete in general.*

For guarded TGDs, the results are summarized in Figure 1.

Linear Datalog$^{\pm}$

We now introduce *linear Datalog$^{\pm}$* as a variant of guarded Datalog$^{\pm}$, where we prove that query answering is in AC_0 in data complexity; for a complete picture of the complexity of query answering under linear Datalog$^{\pm}$ see Figure 1 (first column). Nonetheless, linear Datalog$^{\pm}$ is still expressive enough for representing ontologies.

A TGD is *linear (LTGD)* iff it has a singleton body atom, i.e., is of the form $\varphi(\bar{X},\bar{Y}) \rightarrow \exists \bar{Z}\, \psi(\bar{X},\bar{Z})$, where $\varphi(\bar{X},\bar{Y})$ is an atom. Notice that linear Datalog$^{\pm}$ generalizes the well-known class of inclusion dependencies.

We first define inductively the *derivation depth* of an atom in $chase(\Sigma, D)$ as follows. The atoms in D have derivation depth 0. Let $\underline{a}$ be an atom generated by atoms $\underline{a}_1, \ldots, \underline{a}_n$ through the application of a TGD $\sigma \in \Sigma$. Let also d be the maximum depth of an atom among $\underline{a}_1, \ldots, \underline{a}_n$. Then, the derivation depth of $\underline{a}$ is $d+1$. We denote with $chase^{\gamma}(\Sigma, D)$ the segment of the chase constituted by atoms of derivation depth at most γ.

Definition 8 *A set of TGDs Σ has the* bounded derivation-depth property (BDDP) *iff, for every database D for $\mathcal{R}$ and for every BCQ Q over $\mathcal{R}$, whenever $D \cup \Sigma \models Q$, then $chase^{\gamma}(\Sigma, D) \models Q$, where γ depends only on Q and $\mathcal{R}$.*

Since for LTGDs all descendants of an atom $\underline{a}$ in the chase forest depend only on $\underline{a}$ (we do not define how the chase can be represented as a forest; we refer the reader to (Calì, Gottlob, and Kifer 2008b) for details), then the depth of $\underline{a}$ coincides with the number of applications of the TGD chase rule that are necessary to generate it. Hence, LTGDs have the BDDP.

It is possible to prove that TGDs with the BDDP are first-order rewritable. A class $\mathcal{C}$ of TGDs is *first-order rewritable*, abbreviated as *FO-rewritable*, iff for every set of TGDs Σ in $\mathcal{C}$, and for every BCQ Q, there exists a first-order formula ϕ such that, for every database instance D, $D \cup \Sigma \models Q$ iff $D \models \phi$. It is immediate to notice that, under FO-rewritable TGDs, BCQ answering is in AC_0 in data complexity. We then get the main result of this section.

Theorem 9 *BCQ answering under LTGDs is in* AC_0 *in data complexity.*

Adding Key Dependencies

We now introduce additional constraints to TGDs, namely *key dependencies (KDs)*, with which we assume the reader is familiar. In the presence of TGDs and KDs, the chase procedure needs to take into account the KDs too; violations of KDs are repaired by *equating* values: when equating two symbols, if both of them are constants of Δ, then there is a *hard violation* and the chase *fails*. In this case $D \cup \Sigma_T \cup \Sigma_K$ is unsatisfiable (Σ_T and Σ_K denote the sets of TGDs and KDs, respectively). In certain favourable cases, the addition of KDs do not interact with TGDs. This is captured by the following notion.

Definition 10 *Let Σ_T and Σ_K be sets of TGDs and KDs over a schema $\mathcal{R}$, respectively. Then, Σ_K is* separable *from Σ_T iff for every database D for $\mathcal{R}$, the following conditions are satisfied: (1) if the chase of D w.r.t. Σ_T and Σ_K fails, then D does not satisfy Σ_K; (2) if the chase does not fail, then for every BCQ Q over $\mathcal{R}$, $chase(\Sigma_T \cup \Sigma_K, D) \models Q$ iff $chase(\Sigma_T, D) \models Q$.*

It is possible to show that, if TGDs and KDs are separable, then the complexity of BCQ answering is the same as in the case with TGDs only. A syntactic criterion that is sufficient for separability is shown in (Calì, Gottlob, and Lukasiewicz 2009); TGDs and KDs satisfying the criterion are said *non-conflicting*. Due to lack of space, we refer the interested reader to (Calì, Gottlob, and Lukasiewicz 2009).

Clouds and F-logic Lite

In this section, we briefly explain how the results on F-logic Lite can be obtained as a special case of weakly guarded TGDs. Henceforth, we shall assume that Σ_{FLL} does not contain the rule ρ_4, which can be ignored, as stated earlier in the paper. First, we define the *cloud* of an atom in the chase. Let Σ be a set of WGTGDs over a schema $\mathcal{R}$ and D be an instance for $\mathcal{R}$. For every atom $\underline{a}$ of $chase(\Sigma, D)$ the *cloud* of $\underline{a}$ w.r.t. Σ and D, denoted $cloud(\Sigma, D, \underline{a})$, is the set of all atoms in the chase whose arguments either appear in $\underline{a}$ or in the "active domain" of the input database instance D (i.e., all arguments of atoms in D).

It can be shown that conjunctive query answering is in NP if the following conditions hold.

(1) Σ_{FLL} is weakly guarded.

(2) Σ_{FLL} is such that, for every instance D, there are polynomially many clouds (up to D-isomorphisms, i.e., isomorphisms that leave all arguments of atoms in D unchanged). That is, for every instance D there exists a polynomial pol such that the number of clouds in $chase(\Sigma, D)$ (up to D-isomorphisms) is bounded by $pol(|D|)$.

(3) There is a polynomial pol' such that for each instance D and for each atom a: *(i)* if $\underline{a} \in D$, then $cloud(\Sigma, D, \underline{a})$ can be computed in time $pol'(|D|)$, and *(ii)* if $\underline{a} \notin D$, then $cloud(\Sigma, D, \underline{a})$ can be computed in time $pol'(|D|)$ from D, $\underline{a}$, and $cloud(\Sigma, D, \underline{b})$, where $\underline{b}$ is the predecessor of $\underline{a}$ in the chase forest of $chase(\Sigma, D)$.

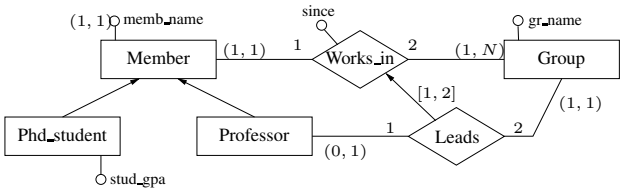

Figure 2: Example EER Schema.

A set of TGDs satisfying the above three properties is said to enjoy the *polynomial cloud criterion (PCC)*. It can be shown that F-logic Lite has this property—see (Calì, Gottlob, and Kifer 2008b) for more details. Thus, the complexity results for F-logic Lite follow from the results for WGTGDs as a special case. We believe that PCC could be used to characterize the complexity of additional ontology languages.

Extended ER in Linear Datalog$^{\pm}$

In this final section, we consider a conceptual modeling formalism, which can be expressed by means of linear Datalog$^{\pm}$ plus KDs.

The formalism, which we call EER (Extended Entity-Relationship), derives from the Entity-Relationship model, where for simplicity we assume all relationships to be binary (the general case, with arbitrary relationship arity, can be treated similarly (Calì, Gottlob, and Pieris 2009)). It can be summarised as follows: *(1)* entities and relationships can have attributes; an attribute can be mandatory (instances have at least one value for it), and functional (instances have at most one value for it); *(2)* entities can participate in relationships; a participation of an entity E in a relationship R can be mandatory (instances of E participate at least once), and functional (instances of E participate at most once); *(3)* is-a relations can hold between entities and between relationships; in the latter case, a permutation $[1, 2]$ or $[2, 1]$ specifies the correspondence among the components.

A knowledge base in the above formalism is called an EER schema. An example one is shown in Figure 2, where the graphical notation is obvious. It is straightforwardly seen that every EER schema can be expressed by means of a relational schema with LTGDs and KDs of a particular form; every set of dependencies in such a form is classified as *conceptual dependencies (CDs)*. Query answering under CDs is not FO-rewritable; this is due to the fact that the KDs in general are not non-conflicting w.r.t. the TGDs (which in this case are linear). We therefore provide a syntactic characterization of a class of CDs, called *non-conflicting CDs (NC-CDs)*, which is FO-rewritable. Before syntactically defining NCCDs, we need a preliminary notion, that is, the notion of *CD-graph*.

Definition 11 *Consider a set Σ of CDs over a schema $\mathcal{R}$. The CD-graph for $\mathcal{R}$ and Σ is defined as follows: (1) the set of nodes is the set of positions in $\mathcal{R}$; (2) if there is a TGD σ in Σ such that the same variable appears in a position π_b in the body and in a position π_h in the head, then there is an arc from π_b to π_h.*

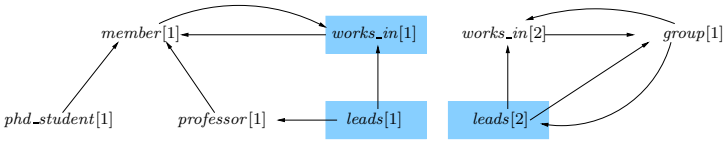

Figure 3: CD-graph for the EER schema in Figure 2; k-nodes are shaded.

A node corresponding to a position derived from an entity (resp., a relationship) is called an *e-node* (resp., an *r-node*). Moreover, an r-node corresponding to a position which is a unary key in a relationship is called a *k-node*. We are now ready to define NCCDs.

Definition 12 *Consider a set Σ of CDs over a schema $\mathcal{R}$, and let G be the CD-graph for $\mathcal{R}$ and Σ. Σ is said to be* non-conflicting *if the following condition is satisfied. For each path $v_1 \frown v_2 \frown \ldots \frown v_m$ in G, where $m \geqslant 3$, such that: (1) v_1 is an e-node; (2) $v_2, \ldots, v_{m-1}$ are r-nodes; (3) v_m is a k-node, there exists a path of only r-nodes from v_m to v_2.*

The schema in Figure 2 has the CD-graph depicted in Figure 3, where we omit the attributes to avoid clutter. It is easy to see that the CDs are non-conflicting. The main result, which precisely characterizes the class of separable EER schemata, follows.

Theorem 13 *Given a set Σ of CDs, we have that Σ is non-conflicting iff Σ is separable.*

For separable CDs, BCQ answering is in AC_0 in data complexity (Calì, Gottlob, and Pieris 2009); we therefore immediately get the following result.

Theorem 14 *BCQ answering under NCCDs 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

Aho, A.; Sagiv, Y.; and Ullman, J. D. 1979. Equivalence of relational expressions. *SIAM J. Comput.* 8(2):218–246.

Beeri, C., and Vardi, M. Y. 1984. A proof procedure for data dependencies. *J. ACM* 31(4):718–741.

Calì, A., and Kifer, M. 2006. Containment of conjunctive object meta-queries. In *Proc. of VLDB*, 942–952.

Calì, A.; Gottlob, G.; and Kifer, M. 2008a. Taming the infinite chase. Unpublished technical report, available from the authors or at `http://benner.dbai.tuwien.ac.at/staff/gottlob/CGK.pdf`.

Calì, A.; Gottlob, G.; and Kifer, M. 2008b. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, 70–80.

Calì, A.; Gottlob, G.; and Lukasiewicz, T. 2009. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, 77–86.

Calì, A.; Gottlob, G.; and Pieris, A. 2009. Tractable query answering over conceptual schemata. In *Proc. of ER*, 175–190.

Calì, A.; Lembo, D.; and Rosati, R. 2003. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS*, 260–271.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-lite family. *J. Autom. Reasoning* 39(3):385–429.

Calvanese, D.; Giacomo, G. D.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *Proc. of PODS*, 385–429.

Chandra, A. K., and Vardi, M. Y. 1985. The implication problem for functional and inclusion dependencies. *SIAM J. Comput.* 14:671–677.

Chen, P. P. 1976. The entity-relationship model: towards a unified view of data. *ACM TODS* 1(1):124–131.

Courcelle, B. 1990. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation* 85(1):12–75.

Deutsch, A.; Nash, A.; and Remmel, J. B. 2008. The chase revisisted. In *Proc. of PODS*, 149–158.

Deutsch, A. 2002. *XML query reformulation over mixed and redundant storage*. Ph.D. Dissertation, Dept. of Computer and Information Sciences, Univ. of Pennsylvania.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data exchange: Semantics and query answering. *Theor. Comput. Sci.* 336(1):89–124.

Goncalves, M. E., and Grädel, E. 2000. Decidability issues for action guarded logics. In *Description Logics*, 123–132.

Gottlob, G., and Nash, A. 2008. Efficient core computation in data exchange. *J. ACM* 55(2).

Gottlob, G.; Leone, N.; and Scarcello, F. 2002. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* 64(3).

Johnson, D. S., and Klug, A. C. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.* 28(1):167–189.

Kifer, M.; Lausen, G.; and Wu, J. 1995. Logical foundations of object-oriented and frame-based languages. *J. ACM* 42:741–843.

Maier, D.; Mendelzon, A. O.; and Sagiv, Y. 1979. Testing implications of data dependencies. *ACM Trans. Database Syst.* 4(4):455–469.

Mitchell, J. 1983. The implication problem for functional and inclusion dependencies. *Information and Control* 56:154–173.

Poggi, A.; Lembo, D.; Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.