Taming the Infinite Chase: Query Answering under Expressive Relational Constraints

Andrea Calì[2,1] **Georg Gottlob**[1,2] **Michael Kifer**[3]

[1]Computing Laboratory, University of Oxford, UK
[2]Oxford-Man Inst. of Quantitative Finance, University of Oxford, UK
[3]Dept. of Computer Science, SUNY Stony Brook, USA

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

Abstract

A crucial task in Knowledge Representation is answering queries posed over a knowledge base, represented as a set of facts plus a set of rules. In this paper we address the problem of answering conjunctive queries posed over knowledge bases where rules are an extension of Datalog rules, called Datalog$^{\exists}$ rules, that may have existentially quantified variables in the head; this kind of rules are traditionally called *tuple-generating dependencies (TGDs)* in the database literature, but they are broadly used in description logics and in ontological reasoning. In this setting, the *chase* algorithm is an important tool for query answering. So far, most of the research has concentrated on cases where the chase terminates. We define and study large classes of TGDs under which the query evaluation problems remain decidable even in case the chase does not terminate. We provide tight complexity bounds for such cases. Our results immediately extend to query containment.

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 one of the important query optimization and schema integration techniques (Aho, Sagiv, and Ullman 1979; Johnson and Klug 1984; Millstein, Levy, and Friedman 2000), and in knowledge representation it has been used for object classification, schema integration, service discovery, and more (Calvanese, De Giacomo, and Lenzerini. 2002). 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; we focus on the former, and our results immediately extend to the latter.

A practically relevant instance of the query containment problem was first studied by Johnson and Klug in (Johnson and Klug 1984) for functional and inclusion dependencies and later in (Calvanese, De Giacomo, and Lenzerini 1998). Several additional decidability results were obtained by focusing on concrete applications. For instance, in (Calì 2007), constraints specific to the entity-relationship diagrams were considered, and (Calì and Kifer 2006) used constraints derived from a subset of F-logic (Kifer, Lausen, and Wu 1995), called F-logic Lite. (Simkus and Eiter 2007) deals with expressive constraints based on Answer Set Programming.

In this paper, rather than focusing on specific logical theories, we analyze the fundamental difficulty that underlies earlier approaches, such as (Johnson and Klug 1984; Calì 2007; Calì and Kifer 2006). They all considered special classes of so-called *tuple-generating dependencies (TGDs)* and *equality-generating dependencies (EGDs)*, all used the technique called *chase*, and all faced the problem that the chase generates infinite relations, and, in general, query answering and containment are undecidable. In our work, we tackle the problem in a much more general way, by carving 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, that we denote with Datalog$^{\exists}$; such rules are known as *tuple-generating dependencies (TGDs)* in the database literature; 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.

A well-known procedure that enforces the validity of a set of TGDs is the chase (Maier, Mendelzon, and Sagiv 1979; Johnson and Klug 1984). Intuitively, the chase "repairs" violations of TGDs by repeatedly adding atoms, until a fixed-point is reached that satisfies all TGDs; the result of the chase may be infinite. The chase has been intensively used in the area of data exchange (Fagin et al. 2005; Gottlob and Nash 2006; Nash, Deutsch, and Remmel 2006). Also, note that the chase is a form of *tableau*, and it has successfully been applied in terminological reasoning based on description logics (Calvanese et al. 2007; Rosati 2007).

Several authors have recently studied data exchange and query evaluation problems for settings where the chase always terminates and thus produces a finite solution (Fagin et al. 2005). To this aim, restrictions for sets of TGDs, such as *weak acyclicity* (first introduced in (Deutsch and Tannen 2003) and heavily utilized, e.g., in (Fagin et al. 2005)), have been defined that guarantee termination for whatever input database. However, little was known about decidable query evaluation and query containment in case of non-terminating chases that produce an infinite result, with the notable exception of the classical work of Johnson and Klug (Johnson and Klug 1984), that shows by a very involved proof that query

containment is decidable in case constraints are either: *(i) inclusion dependencies (IDs)* alone, i.e., TGDs whose body and head consist both of a single atom only; *(ii)* a special class of IDs and *key dependencies (KDs)*.

The topic of the present paper is to consider significantly larger classes of TGDs. In particular, we define the notions of sets of *guarded TGDs (GTGDs)* and of *weakly guarded TGDs (WGTGDs)*. A TGD is guarded if its body contains an atom called *guard* that covers all variables occurring in the body. Weakly guarded TGDs are a generalization of guarded TGDs that only require guards to cover all variables occurring at *affected* positions only, where affected positions are positions in predicates that may contain some fresh labelled nulls generated during the chase. The notion of guard is crucial, since query evaluation becomes undecidable once we allow the presence of a single non-guarded TGD. Instead, under WGTGDs, the (possibly infinite) result of the chase has bounded treewidth (Theorem 23), and we use this fact together with well-known results about the generalized tree-model property (Courcelle 1990; Goncalves and Grädel 2000) for a short proof that Boolean query evaluation and query containment are decidable under WGTGDs (and thus also with GTGDs). Unfortunately, this decidability result does not allow us to derive useful complexity bounds.

Our main contribution lies in the complexity bound for query evaluation under WGTGDs and GTGDs. We show that the complexity of query evaluation (and equivalently, query containment) under WGTGDs is EXPTIME-hard, in case of a fixed set of TGDS, and 2-EXPTIME-hard in case the TGDs are part of the input.

As for upper bounds, let us first remark that we cannot (as one may think at the first glance) directly or easily use known results on guarded logics (Goncalves and Grädel 2000) to derive complexity results for query evaluation, since queries are in general non-guarded. We therefore develop new algorithms, and prove that query answering is EXPTIME-complete in case of bounded predicate arities, and even in case the set of constraints is fixed, and is 2-EXPTIME complete in general. The proof of the upper bound is based on an alternating algorithm that mimicks the chase by using a finite number of configurations. Each of them corresponds to the *cloud* of one atom $\underline{a}$, i.e. the set of atoms in the chase whose arguments either appear in $\underline{a}$ or in the "active domain" of the input database instance.

Then, we derive complexity results for query answering under GTGDs. While in the general case the complexity is the same as for WGTGDs, interestingly, when considering a *fixed* set of dependencies (which is the usual setting in data exchange and in description logics), we get much better results: evaluating Boolean queries is NP-complete (same complexity of answering without constraints (Chandra and Merlin 1977)), and in PTIME in case the query is atomic. Our results subsume the results of (Johnson and Klug 1984) on IDs alone as a special case.

The complexity results in this paper, together with some additional ones that are immediate consequence of them, are summarized in Figure 1, where all complexity bounds are tight, and Σ denotes the set of Datalog$^{\exists}$ rules. By "bounded width" we intend bounded treewidth or even hypertree width

CQ type	**GTGDs** variable Σ	**WGTGDs** variable Σ	**GTGDs** fixed Σ	**WGTGDs** fixed Σ
general	2-EXPTIME	2-EXPTIME	NP	EXPTIME
bounded width, fixed, and atomic	2-EXPTIME	2-EXPTIME	PTIME	EXPTIME

Figure 1: Summary of complexity results

(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. With such results, we subsume both the main decidability and complexity result in (Johnson and Klug 1984), and decidability and complexity results on F-logic lite (Calì and Kifer 2006) as special cases, and we are actually way more general.

A relevant application. F-logic (Kifer, Lausen, and Wu 1995) is 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 can be ignored wrt query answering. The results of our paper apply to obtained set of constraints, since the TGDs are WGTGDs. We should note, however, that our results cover a much larger set of F-logic queries than (Calì and Kifer 2006); in particular, we can consider queries defined with recursive Datalog$^{\exists}$ rules by encoding them into TGDs, as long as such TGDs are WGTGDs. 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.

Further results. We mention some relevant results that will be published elsewhere. *Equality generating dependencies (EGDs)* are a generalization of functional dependencies in relational databases. An EGD, like a TGD, is an implication between a head and a body, where the body is a conjunction of atoms, and the head is an equality atom that equates two variables appearing in the body. Query answering becomes easily undecidable in the presence of EGDs and TGDs, even when TGDs are simple IDs. In particular, in (Calì, Lembo, and Rosati 2003) it is shown that query answering is undecidable under IDs and *functional dependencies (FDs)*, where a FD is a special case of EGD. We define a class of *innocuous* EGDs, that enjoy the property that they can be ignored in the query answering phase, since they do not actually interact with TGDs.

We also describe a semantic condition, called *Polynomial Clouds Criterion (PCC)*, imposing that the number of clouds generated during a chase is polynomial in the size of the input database instance, and the cloud of each generated atom $\underline{a}$ can be obtained in polynomial time from the parent of $\underline{a}$ in the chase. Whenever a set of WGTDs fulfills the PCC, then answering Boolean queries is in NP, and answering atomic queries, as well as queries of bounded hypertree width, is in PTIME.

Preliminaries

In this section we define the basic notions that we shall use throughout the paper. More details can be found in (Calì, Gottlob, and Kifer 2008).

We introduce the following pairwise disjoint sets of symbols: *(i)* an infinite set Δ of *constants*, which constitute the "normal" domain of a database schema $\mathcal{R}$; *(ii)* an infinite set Δ_N of *labeled nulls*, which will be used as "fresh" Skolem terms; *(iii)* an infinite set Δ_V of variables, which are used in queries. Intuitively, a null is a placeholder for an unknown value; two distinct nulls may denote the same value[1]; therefore, a null can be seen as a variable. We assume a lexicographic order on Δ and Δ_N, with every symbol in Δ_N following 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.

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 and conjunctive queries. As mentioned, we denote relational schemas by $\mathcal{R}$, queries by Q, database instances by D, and the answers to a query Q, evaluated on the database instance D, by $Q(D)$. In the following, we shall consider *conjunctive queries (CQs)*, with which we assume the reader is familiar. $vars(Q)$ denotes the set of variables appearing in a CQ Q. Database instances, or simply databases, 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 $P(a_1, ..., a_n)$, where P 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* $P[i]$ in a relational schema is identified by a relational predicate P and its i-th attribute, identified by the integer i.

We now come to the notion of homomorphism. A *mapping* from one set of symbols, S_1, to another set of symbols, S_2, is a function $\mu : S_1 \rightarrow S_2$ defined as follows: *(i)* $\emptyset$ (empty mapping) is a mapping; *(ii)* if μ_0 is a mapping, then $\mu_0 \cup \{X \rightarrow Y\}$, where $X \in S_1$ and $Y \in S_2$ is a mapping if μ_0 does not already contain some $X \rightarrow Y'$ with $Y \neq Y'$. If $X \rightarrow Y$ is in a mapping μ, we write $\mu(X) = Y$. A *homomorphism* from a set of atoms D_1 to another set of atoms D_2, both over the same relational schema $\mathcal{R}$, is a mapping μ from $\Delta \cup \Delta_N \cup \Delta_V$ to $\Delta \cup \Delta_N \cup \Delta_V$ such that the following conditions hold: *(1)* if $c \in \Delta$ then $\mu(c) = c$; *(2)* if $c \in \Delta_N$ then $\mu(c) \in \Delta \cup \Delta_N$; *(3)* if the atom $R(c_1, \ldots, c_n)$ is in D_1, then the atom $R(\mu(c_1), \ldots, \mu(c_n))$ is in D_2. The notion of homomorphism is naturally extended to atoms as follows. If $f = R(c_1, \ldots, c_n)$ is an atom and μ a homomorphism, we define $\mu(f) = R(\mu(c_1), \ldots, \mu(c_n))$. For a *set* of atoms, $F = \{f_1, \ldots, f_m\}$, we define $\mu(F) = \{\mu(f_1), \ldots, \mu(f_m)\}$. The set of atoms $\{\mu(f_1), \ldots, \mu(f_m)\}$ is also called *image* of F wrt μ. In this case, we say that μ *maps* F to $\mu(F)$. For a *conjunction* of atoms $\Phi = \phi_1 \wedge \ldots \wedge \phi_n$, we use $\mu(\Phi)$ to denote the set of atoms $\mu(\{\phi_1, \ldots, \phi_n\})$.

The notion of a homomorphism serves to define the answers to conjunctive queries. The answers to a conjunctive query Q of the form $Q(X_1, \ldots, X_n) \leftarrow \Phi(\bar{X})$ over a database instance D, denoted $Q(D)$, are defined as follows: an atom $t \in \Delta^n$ is in $Q(D)$ iff there exists a homomorphism μ that maps $\Phi(\bar{X})$ to atoms of D, and $(X_1, \ldots, X_n)$ to t (notice that only tuples made of constants in Δ are allowed to be in the answer).

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 $\forall \bar{X} \forall \bar{Y} \Phi(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z} \Psi(\bar{X}, \bar{Z})$, where $\Phi(\bar{X}, \bar{Y})$ 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 in a database D for $\mathcal{R}$ if, whenever there is a homomorphism h that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to atoms of D, there exists an extension h' of h (i.e., $h' \supseteq h$) that maps the atoms of $\Phi(\bar{X}, \bar{Z})$ to atoms of D.*

To simplify the notation, we will usually omit the 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 2006) and in query answering over incomplete data (Calì, Lembo, and Rosati 2003). Given an incomplete database, i.e., a database that does not satisfy all the constraints in Σ, we first define the set of completions (aka *repairs*) of that database, which we call *solutions*.

Definition 2. *Consider a relational schema $\mathcal{R}$, a set of TGDs Σ, and a database instance D for $\mathcal{R}$. The set of instances B such that $B \models D \cup \Sigma$, is called the set of* solutions *of D given Σ, and is denoted by $sol(\Sigma, D)$.*

Definition 3. *Consider a relational schema $\mathcal{R}$, a set of TGDs Σ, and a database instance D for $\mathcal{R}$. The* answers *to a conjunctive query 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 $\underline{a} \in Q(B)$.*

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, the chase procedure is a process of repairing a database with respect to a set of database dependencies, so that the result of the chase satisfies the dependencies. By abuse of terminology, by "chase" we referboth to the chase procedure and to its output. Chase works on a database through so-called TGD and EGD *chase rules*. TGD rules come in two flavors: *oblivious* and *restricted*, where the restricted one repairs TGDs only when they are not satisfied. We focus our attention to the oblivious one, since it makes proofs technically simpler. The *TGD chase rule* defined below is the building block of the construction of the chase.

(OBLIVIOUS) TGD CHASE RULE. Consider a relational database D for a schema $\mathcal{R}$, and a TGD σ on $\mathcal{R}$ of the

[1]Notice that this does not hold for constants in Δ; we adopt the *unique name assumption*, imposing that different constants in Δ denote different objects.

form $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$. The TGD σ is *applicable* to D if there is a homomorphism h that maps the atoms of $\Phi(\bar{X}, \bar{Y})$ to atoms of D. Let σ be applicable and h_1 be a homomorphism that extends h as follows: for each $X_i \in \bar{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \bar{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, $z_j \notin D$, and z_j lexicographically follows all other labeled nulls already introduced, and it is different from them. The result of the application of the TGD σ is the addition to D of all the atomic formulas in $h_1(\Psi(\bar{X}, \bar{Z}))$ that are not already in D. ■

The chase algorithm consists of an exhaustive application of the TGD chase rule, that may lead to an infinite result. The chase rule is applied in a breadth-first fashion, as described below.

We first give an important notion in the chase, namely the *level* of an atom. Let D the *initial* database from which the chase is constructed. Then: *(1)* The atoms in D are said to have level 0. *(2)* Suppose a TGD rule for $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$ is applied at some point in the construction of the chase, and let h, h_1 be as in the TGD chase rule. If the atom with highest level among those in $h_1(\Phi(\bar{X}, \bar{Y}))$ has level k, then every atom in $h_1(\Psi(\bar{X}, \bar{Z}))$ that is actually added has level $k + 1$.

Let D be a database and Σ a set of TGDs Then, the chase of D with respect to Σ, denoted $chase(\Sigma, D)$, is the database constructed by an iterative application of the TGD chase rules as follows: let $I_1, \ldots, I_k$ be all possible images of bodies of TGDs in Σ wrt some homomorphism, and let $\underline{a}_i$ be the atom with highest level in I_i; let M be such that $level(\underline{a}_M) = \min_{1 \leq i \leq k}\{level(\underline{a}_i)\}$: among the possible applications of TGDs, choose the lexicographically first among those that utilize a homomorphism from the body of a TGD to I_M.

Containment of queries over relational databases has long been considered a fundamental problem in query optimization – especially query containment under constraints such as TGDs.

Definition 4. *Consider a relational schema $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, and two conjunctive queries Q_1, Q_2 expressed over $\mathcal{R}$. We say that Q_1 is contained in Q_2 under Σ, denoted $Q_1 \subseteq_\Sigma Q_2$, if for every database instance D for $\mathcal{R}$ such that $D \models \Sigma$ we have $Q_1(D) \subseteq Q_2(D)$.*

Query containment and answering under TGDs as defined above are tightly connected, as we explain in the following.

Theorem 5. *(Johnson and Klug 1984; Fagin et al. 2005; Nash, Deutsch, and Remmel 2006). Consider a relational schema $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, a conjunctive query Q, and a ground atom $\underline{a}$; we have that $\underline{a} \in ans(Q, \Sigma, D)$ iff $\underline{a} \in Q(chase(\Sigma, D))$. As a special case, if Q is Boolean,*

This result is important, and it holds because the (possibly infinite) chase is a *universal solution* (Fagin et al. 2005), i.e., for every $B \in sol(\Sigma, D)$, there exists a homomorphism that maps $chase(\Sigma, D)$ onto B. In (Nash, Deutsch, and Remmel 2006) it is shown that the chase constructed with respect to TGDs is defined also when it is infinite, and it is a universal solution.

The following is straightforwardly obtainable from (Johnson and Klug 1984; Nash, Deutsch, and Remmel 2006).

Theorem 6. *Consider a relational schema $\mathcal{R}$, a set Σ of TGDs on $\mathcal{R}$, and two queries Q_1, Q_2 on $\mathcal{R}$. We have that $Q_1 \subseteq_\Sigma Q_2$ iff $\lambda(head(Q_1)) \in Q_2(chase(\Sigma, \lambda(body(Q_1))))$, where λ is a freezing homomorphism for Q_1, i.e., a homomorphism that maps every distinct variable in Q_1, into a distinct fresh constant.*

We now define three relevant decision problems, and prove their equivalence.

Definition 7.

- *The* conjunctive query evaluation decision problem CQ-Eval *is defined as follows. Given a conjunctive query Q, a set of TGDs Σ, a database D and a ground atom $\underline{a}$, decide whether $\underline{a} \in ans(Q, \Sigma, D)$.*
- *The* Boolean conjunctive query evaluation problem BCQ-Eval *is defined as follows. Given a Boolean conjunctive query Q, a set of TGDs Σ, and a database D, decide whether $\langle\rangle \in ans(Q, \Sigma, D)$, i.e., Q has a positive answer (also written $\Sigma \cup D \models Q$).*
- *The* conjunctive query containment decision problem CQ-Cont *is defined as follows. Given two CQs Q_1 and Q_2, and a set Σ of TGDs, decide whether for every database instance D it holds $Q_1(D) \subseteq Q_2(D)$.*

The following result is folklore, and implicit in (Chandra and Merlin 1977).

Lemma 8. *The problems CQ-Eval and BCQ-Eval are* LOGSPACE*-equivalent.*

Proof. Notice that BCQ-Eval can be trivially made into a special instance of CQ-Eval, e.g., by adding a propositional atom as head atom. It thus suffices to show that CQ-eval polynomially reduces to BCQ-eval. Let (Q, D, Σ, a) be an instance of CQ-Eval. Assume the head atom of Q is $R_h(X_1, \ldots, X_k)$ and $a = R_a(c_1, \ldots, c_k)$. Then, define Q' to be the Boolean conjunctive query whose atoms are those in $body(Q)$ plus $R^*_h(X_1, \ldots, X_k)$, where R^*_h is a fresh predicate symbol not occurring in D and Q (and therefore not in Σ, since the TGDs in Σ are on the same relational schema). Moreover, let $a' = R^*_h(c_1, \ldots, c_k)$. It is easy to see that $a \in Q(chase(\Sigma, D))$ iff $chase(\Sigma, D \cup a') \models Q'$. □

The following corollary descends from Theorems 6 and 5.

Corollary 9. *Under TGDs, the problems CQ-eval and CQ-Cont are mutually* PTIME*-reducible.*

By Lemma 8 and by Corollary 9, we can conclude that the three following problems are LOGSPACE-equivalent: *(1)* CQ-eval under TGDs, *(2)* BCQ-eval under TGDs, *(3)* CQ-Cont under TGDs. Henceforth, we will concentrate on only one of these problems, namely the BCQ-eval problem. All complexity results carry over to the other problems.

The following lemma allows us to restrict our attention on a restricted class of TGDs.

Lemma 10. *Let Q be a Boolean conjunctive query over a schema $\mathcal{R}$, and let Σ be a set of TGDs on $\mathcal{R}$. Then, from Σ we can construct in* LOGSPACE *a set of TGDs Σ' such that each TGD in Σ' has only one atom in its head and for each instance D, $\Sigma \cup D \models Q$ iff $\Sigma' \cup D \models Q$.*

Proof. To obtain Σ' from Σ is sufficient to replace each rule of the form $r : body(\bar{X}) \rightarrow head_1(\bar{Y}), head_2(\bar{Y}), \ldots, head_k(\bar{Y})$, where $k > 1$ and $\bar{Y}$ is the set of all the variables that appear in the head (that may include part of $\bar{X}$), with the following set of rules: $body(\bar{X}) \rightarrow V(\bar{Y})$, $V(\bar{Y}) \rightarrow head_1(\bar{Y})$, $V(\bar{Y}) \rightarrow head_2(\bar{Y})$, $\ldots$, $V(\bar{Y}) \rightarrow head_k(\bar{Y})$ where V is a fresh predicate symbol, having the same arity as the number of variables in $\bar{Y}$; notice also that in general not all the variables in $\bar{Y}$ also appear in $\bar{X}$. It is straightforward to see that, except for the atoms of the form $V(\bar{Y})$, $chase(\Sigma, D)$ and $chase(\Sigma', D)$ coincide. The atoms of the form $V(\bar{Y})$, being introduced only in the transformation above, do not match any predicate symbol in Q, hence, $chase(\Sigma, D) \models Q$ iff $chase(\Sigma', D) \models Q$. □

Henceforth, we shall always assume without loss of generality that every TGD has a singleton atom in its head.

We now come to the notions of treewidth, tree decompositions and bounded-treewidth model property (see, e.g., (Goncalves and Grädel 2000)). Roughly, the treewidth of a database instance D (or of an arbitrary set of atoms whose arguments may contain labeled nulls) is the treewidth of the Gaifman graph of D.

Consider a graph $G = (V, E)$; a *tree decomposition of* G consists of a tree $T = (N, A)$ and a labeling function $\lambda : N \rightarrow 2^V$ with the following properties:

(i) for every $v \in V$, there exists $n \in N$ such that $v \in \lambda(N)$;
(ii) for every arc $(v_1, v_2) \in E$, there exists $n \in N$ such that $\{v_1, v_2\} \subseteq \lambda(n)$;
(ii) for every $v \in V$, the set $\{n \in N \mid v \in \lambda(n)\}$ induces a (connected) subtree in T.

The *Gaifman graph* of a relational instance D (or of a generic set of atoms) is a nondirected graph defined as follows:

- the nodes are the symbols in $dom(D)$ (in general, constants in Δ and nulls in Δ_N);
- there exists an arc (c_1, c_2) between c_1 and c_2 if there exist some atom in D that has both c_1 and c_2 as arguments.

The *witth* of a tree decomposition (T, λ), with $T = (N, A)$, of graph $G = (V, E)$ is the integer $\max_{n \in N} |\lambda(n)|$. The *treewidth* of a graph $G = (V, E)$, denoted $tw(G)$, is the minimum width among all tree decompositions. Given a relational instance D (or an arbitrary set of atoms), its treewidth $tw(D)$ is defined as the treewidth of its Gaifman graph.

A class $\mathcal{C}$ of formulae has the bounded treewidth model property if for each $\phi \in \mathcal{C}$, whenever ϕ is satisfiable, then it is possible to compute a number $f(\phi)$ such that ϕ has a model of treewidth $\leq f(\phi)$.

The following straightforwardly follows from (Courcelle 1990); see also (Goncalves and Grädel 2000).

Theorem 11. *If a set of first-order formulae has the bounded-treewidth model property, then checking satisfiability for it is decidable.*

(Weakly) Guarded TGDs

This section introduces the special classes of *guarded* and *weakly guarded* TGDs , which have a number of useful properties.

We first give the notion of *affected* position of a relational schema, given a set of TGDs Σ.

Definition 12. *Given a relational schema $\mathcal{R}$ and a set of TGDs Σ over $\mathcal{R}$, an* affected position *in $\mathcal{R}$ wrt 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 wrt 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 wrt Σ.*

Example 1. Consider the following set of TGDs:

$$\begin{array}{lrcl} \sigma_1: & P_1(X.Y), P_2(X,Y) & \rightarrow & \exists Z\, P_2(Y,Z) \\ \sigma_2: & P_2(X.Y), P_2(W,X) & \rightarrow & P_1(Y,X) \end{array}$$

Notice that $P_2[2]$ is affected since Z in σ_1 is existentially quantified in σ_1. Considering again σ_1, the variable Y appears in $P_2[2]$ but also in $P_1[2]$, therefore it does not make the position where it appears in the head affected. Variable X in σ_2 appears in the affected position $P_2[2]$ but also in $P_2[1]$, which is not affected; therefore, it does not make the position where it appears in the head affected. On the contrary, Y appears in $P_2[2]$ and nowhere else, thus causing $P_1[1]$ to be affected. □

Definition 13. *Given a TGD σ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$, we say that σ is a* (fully) guarded TGD (GTGD) *if there exists an atom in the body, called a* guard*, that contains all the universally quantified variables of σ, i.e., all the variables $\bar{X}, \bar{Y}$ that occur in $\Phi(\bar{X}, \bar{Y})$, the body of σ.*

Example 2. The TGD $R_1(X, Y, a), R_2(Y), R_3(X, b) \rightarrow R_4(Z, X)$ is guarded; in particular, the guard is the atom $R_1(X, Y, a)$, since it contains all the universally quantified variables of the TGD. □

Definition 14. *Given a TGD σ of the form $\Phi(\bar{X}, \bar{Y}) \rightarrow \Psi(\bar{X}, \bar{Z})$, belonging to a set of TGDs Σ on a schema $\mathcal{R}$, we say that σ is a* weakly guarded TGD (WGTGD) *wrt Σ if there exists an atom in the body, called a* weak guard*, that contains all the universally quantified variables of σ that appear* only *in positions that are affected wrt Σ.*

Example 3. Consider again the two TGDs in Example 1. In σ_1 both atoms are obviously guards (and weak guards of course), since they contain all the universally quantified variables. In σ_2, Y is the only variable that appears in affected positions only is Y; therefore, the first atom is a weak guard. □

It is important to realize that the transformation described in the proof of Lemma 10 preserves the guardedness and weak guardedness properties. Therefore, we can still assume that TGDs have singleton heads.

The following theorem shows that even a *single* unguarded rule can destroy the decidability of simplest reasoning tasks under TGDs (Calì, Gottlob, and Kifer 2008).

Theorem 15. *There is a fixed set of TGDs Σ_u such that all but one TGDs of Σ_u are guarded and it is undecidable whether $D \cup \Sigma_u \models Q$, or, equivalently, whether $Q \in chase(\Sigma_u, D)$.*

Proof (sketch). The theorem is proved by simulating a Turing machine (TM) with TGDs, making use of a single non-guarded TGD. Note that, by using two guarded rules and a single unguarded rule, we can define an infinite grid as follows. Let D contain (among other atoms) the atom $index(0)$. Consider the following TGDs:

$$index(x) \rightarrow \exists y\ next(x, y)$$
$$next(x, y) \rightarrow index(y)$$
$$next(x, x'), next(y, y') \rightarrow grid(x, y, x', y')$$

These rules, the third of which is unguarded, define an infinite grid whose points have co-ordinates x and y (representing tape position and time, respectively), and where for each point (x, y) its tape successor (x', y) and its time successor (x, y') is also encoded. Note that only the last of the TGDs is non-guarded.

It is not hard to see that we can simulate the progress of a TM using suitable initialization atoms in D and guarded TGDs. To this aim, we need additional predicates $cursor(Y, X)$ (the cursor is in position X at time Y), $state(Y, S)$ ($\mathcal{M}$ is in state S at time Y), $content(X, Y, A)$ (at time Y, the content of position X in the tape is A), plus some auxiliary predicates. The following rule encodes the behaviour of the TM on all transition rules that move the cursor to the right:

$$grid(S_1, A_1, S_2, A_2, right, X_1, Y_1, X_2, Y_2),$$
$$cursor(Y_1, X_1),$$
$$state(Y_1, S_1), content(X_1, Y_1, A_1) \rightarrow$$
$$cursor(Y_2, X_2), content(X_1, Y_2, A_2),$$
$$state(Y_2, S_2), mark(Y_1, X_1)$$

Such a rule has also obvious sibling rules for "left" and "stay" moves. Additional "inertia" rules are needed, that make use of the $mark$ predicate; these rules ensure that all non-marked positions in the tape are not modified. It is a straightforward exercise to find a *fixed* number of such rules. Notice that the fact that the rules have multiple atoms in the head is not a loss of generality by virtue of Lemma 10. Finally, we assume without loss of generality that our Turing machine has a single halting state s_0 which is encoded by the atom $halt(s_0)$ in D. We add a guarded rule

$$state(Y, S), halt(S) \rightarrow stop(S)$$

It is now clear that the machine halts iff $chase(\Sigma_u, D) \models \exists X\ stop(X)$, i.e., iff $\Sigma_u \cup D \models \exists X\ stop(X)$. We have thus reduced the halting problem to the problem of answering atomic queries to a database under Σ_u. The latter problem is thus undecidable. □

Definition 16. *Let Σ be a set of WGTGDs, D be a database, and $U = chase(\Sigma, D)$. The* guarded chase graph *$GCG(\Sigma, D)$ is defined as follows. The set of vertices is constituted by the atoms of U, and there are two kind of arcs:* normal *and* dotted*. Consider a TGD ρ with a weak guard $\underline{\gamma} \in body(\rho)$, which was used in an application of a TGD rule using a homomorphism h that maps the body and head of ρ to $chase(\Sigma, D)$. Then (i) there is a* dotted *arc from every atom in $h(body(\rho)) - \{h(\underline{\gamma})\}$ to every atom in $h(head(\rho))$; (ii) there is a* normal *arc from $h(\underline{\gamma})$ to every atom of $h(head(\rho))$.*

It is easily seen that the graph obtained from $GCG(\Sigma, D)$ by omitting all dotted arcs is a forest.

Definition 17. *The forest obtained from $GCG(\Sigma, D)$ by dropping all dotted arcs is called the* guarded chase forest *and is denoted by $GCF(\Sigma, D)$.*

Definition 18. *Let D be a possibly infinite relational instance for a schema $\mathcal{R}$, and let S be a set. We are mainly interested in sets S such that $S \subseteq dom(D)$, but for the sake of generality, we do not exclude other sets here.*

- *An $[S]$-join forest of D is an undirected labeled forest $T = (V, E, \lambda)$, whose labeling function $\lambda : V \rightarrow D$ is such that :*
  (i) $D \subseteq \lambda(V)$, and
  (ii) T is $[S]$-connected, i.e., for each $c \in dom(D) - S$, the set $\{v \in V \mid c \text{ occurs in } \lambda(v)\}$ induces a connected subtree in T.
- *We say that D is $[S]$-acyclic iff D has an $[S]$-join forest.*

The above definition generalizes the classical notion of hypergraph acyclicity (Beeri et al. 1981) of an instance (or, equivalently, of a query). In fact, an instance or a query (seen as an instance) is hypergraph-acyclic iff it is $[\emptyset]$-acyclic.

The following Lemma follows straightforwardly from the definitions of $[S]$-acyclicity.

Lemma 19. *Given a database instance D for a schema $\mathcal{R}$, and a set S, if D is $[S]$-acyclic, then $tw(D) \leq |S| + w$, where w is the maximum arity of any predicate symbol in $\mathcal{R}$.*

Proof (sketch). A tree decomposition $(T = (V, E), \chi)$ of width $\leq |S| + w$ can be obtained from an $[S]$-join tree $(T = (V, E), \lambda)$ of B by defining $\forall v \in V, \chi(v) = S \cup \lambda(v)$. □

Definition 20. *Let D be an instance for a schema $\mathcal{R}$. Let $dom(D)$ be the* active domain *of D, i.e., $dom(D)$ contains all constants and labeled nulls that occur in D. The Herbrand Base $HB(D)$ of D is the set of all atoms that can be formed using the predicate symbols of $\mathcal{R}$ and arguments in $dom(D)$. We define:*

- $chase^{\perp}(\Sigma, D) = chase(\Sigma, D) \cap HB(D)$, *and*
- $chase^{+}(\Sigma, D) = chase(\Sigma, D) - chase^{\perp}(\Sigma, D)$

Notice that $chase^{\perp}(\Sigma, D) \cup chase^{+}(\Sigma, D) = chase(\Sigma, D)$ and $chase^{\perp}(\Sigma, D) \cap chase^{+}(\Sigma, D) = \emptyset$. Moreover, if D is null-free (which will be the case in many applications), then $chase^{\perp}(\Sigma, D)$ is the finite set of all null-free atoms in $chase(\Sigma, D)$, while $chase^{+}(\Sigma, D)$ may be infinite.

Lemma 21. *If Σ is a set of WGTGDs and D an instance, then $chase^{+}(\Sigma, D)$ is $[dom(D)]$-acyclic.*

Lemma 22. *If Σ is a set of WGTGDs and D an instance of a schema $\mathcal{R}$, then $tw(chase(\Sigma, D)) \leq |D| + w$, where w is the maximum arity of a predicate in $\mathcal{R}$.*

Proof. Follows easily from Lemmata 19 and 21. □

Theorem 23. *Given a relational schema $\mathcal{R}$, a set of WGTGDs Σ, a Boolean conjunctive query Q, and a database instance for $\mathcal{R}$, the problem of checking whether $\Sigma \cup D \models Q$ (or, equivalently, $chase(\Sigma, D) \models Q$) is decidable.*

Proof. Here we consider the query Q as an existentially-quantified logical sentence. We rely on the fact that both $chase(\Sigma, D) \wedge Q$ and $chase(\Sigma, D) \wedge \neg Q$ have a (possibly infinite) model of bounded treewidth, when they are satisfiable. This follows from the fact that $chase(\Sigma, D)$ is universal for D under Σ and has bounded treewidth. Our claim now follows by a well-known result of Courcelle (Courcelle 1990), also found in (Goncalves and Grädel 2000), that generalizes an earlier result of Rabin (Rabin 1969). This result (Theorem 11) states that classes of first-order logic (more generally, monadic second-order logic) that enjoy the bounded treewidth model property are decidable. □

The above theorem establishes decidability of query answering under WGTGDs, but it tells little about the complexity. This is the subject of the next section.

Complexity

In this section we present several complexity results about query answering under guarded and weakly-guarded TGDs.

EXPTIME Hardness

Theorem 24. *Given a relational schema $\mathcal{R}$, a set of WGTGDs Σ, a Boolean conjunctive query Q, and a database instance D for $\mathcal{R}$, the problem of determining whether $chase(\Sigma, D) \models Q$ is* EXPTIME*-hard. In the case where the arity of predicates in $\mathcal{R}$ is not fixed, the same problem is 2-*EXPTIME *hard. The same results hold in case of atomic queries and even fixed queries.*

Proof (sketch). The proof is by simulation of a PSPACE Alternating Turing Machine (ATM); in the case of unlimited arity of the predicates in $\mathcal{R}$, we can simulate an EXPSPACE ATM. A detailed proof can be found in the full version of this paper (Calì, Gottlob, and Kifer 2008). □

Squid Decompositions

In this section we define notion of *squid decomposition*, and prove a lemma called "Squid Lemma" which will be a useful tool for proving the upper complexity bound of the query answering problem.

Definition 25. *A* squid decomposition *$\delta = (h, H, T)$ of a Boolean conjunctive query Q consists of a mapping $h : vars(Q) \rightarrow vars(Q)$ of Q and a partition of $h(Q)$ into two sets H and T such that $T = h(Q) - H$ and such that T is $[vars(H)]$-acyclic. We refer to H as the* head *of δ, and to T as the* tentacles *of δ. The set of all squid decompositions of Q is referred to as $squidd(Q)$.*

One may imagine the set H in a squid decomposition as the head of a squid, and the set T as a forest of tentacles attached to that head, in a way similar to what is done in (Glimm et al. 2008). Note that a squid decomposition of Q is not necessarily a query folding (Chandra and Merlin 1977; Qian 1996) of Q, because h does not need to be an endomorphism of Q, in other terms, we do not require that $h(Q) \subseteq Q$. Of course, h is a homomorphism.

Example 4. Consider the following Boolean conjunctive query (the schema is obvious):

$$Q() \leftarrow R(X,Y), R(X,Z), R(Y,Z),$$
$$R(Z,V_1), R(V_1,V_2), R(V_2,V_3), R(V_3,V_4), R(V_4,V_5),$$
$$R(V_1,V_6), R(V_6,V_5), R(V_5,V_7),$$
$$R(Z,U_1), S(U_1,U_2,U_3), S(U_3,U_4,U_5)$$

A possible squid decomposition is the following homomorphism h defined as: $h(V6) = V2$, $h(V4) = h(V5) = h(V7) = V3$, and as the identity on the other variables. The result of the decomposition is the query shown in Figure 2, where its join graph is depicted, in order to distinguish the head from the tentacles. □

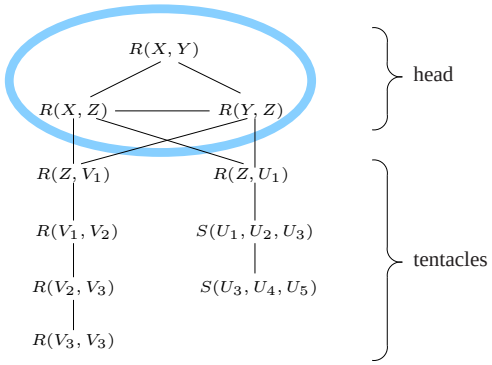

Figure 2: Squid decomposition from Example 4

The following Lemma, whose proof can be found in the full paper (Calì, Gottlob, and Kifer 2008), will be used as a main tool in the subsequent complexity analysis.

Lemma 26 (Squid Lemma)**.** *Let Σ be a set of WGTGDs on a schema $\mathcal{R}$, D a (ground) database instance for $\mathcal{R}$, and Q a conjunctive query, then $chase(\Sigma, D) \models Q$ iff there is a squid decomposition $\delta = (h, H, T) \in squidd(Q)$ and a homomorphism $\theta : dom(h(Q)) \rightarrow dom(chase(\Sigma, D))$ such that: (i) $\theta(H) \subseteq chase^{\perp}(\Sigma, D)$, and (ii) $\theta(T) \subseteq chase^{+}(\Sigma, D)$.*

Clouds and the complexity of query answering under WGTGDs

To study the complexity of query answering under WGTGDs, we introduce the notion of *cloud*.

Definition 27. *Let Σ be a set of WGTGDs on a schema R and D an instance for $\mathcal{R}$. For every atom $\underline{a}$ of $chase(\Sigma, D)$ the* cloud *of $\underline{a}$ with respect to Σ and D, denoted $cloud(\Sigma, D, \underline{a})$, is the set of all atoms in $chase(\Sigma, D)$ whose arguments are in $dom(\underline{a}) \cup dom(D)$. More formally, $cloud(\Sigma, D, \underline{a}) = \{\underline{b} \in chase(\Sigma, D) \mid dom(\underline{b}) \subseteq dom(\underline{a}) \cup dom(D)\}$. Notice that for every atom $\underline{a} \in$*

$chase(\Sigma, D)$ *we have* $D \subseteq cloud(\Sigma, D, \underline{a})$. *Moreover, we define*

$$clouds(\Sigma, D) = \{cloud(\Sigma, D, \underline{a})\}_{\underline{a} \in chase(\Sigma, D)}$$
$$clouds^{+}(\Sigma, D) = \{(\underline{a}, cloud(\Sigma, D, \underline{a})) \mid \underline{a} \in chase(\Sigma, D)\}$$

A set $S \subseteq cloud(\Sigma, D, \underline{a})$ *is called a* subcloud *of* $\underline{a}$ *(with respect to* Σ *and* D*). The set of all subclouds of an atom* $\underline{a}$ *is denoted by* $subclouds(\Sigma, D, \underline{a})$. *Finally, we define* $subclouds^{+}(\Sigma, D) = \{(\underline{a}, C) \mid \underline{a} \in chase(\Sigma, D) \wedge C \subseteq cloud(\Sigma, D, \underline{a})\}$

Definition 28. *Let* D *be an instance for a schema* $\mathcal{R}$. *Let* α *and* β *be two constructs consisting each of one atom of* $HB(D)$*, or a set of atoms of* $HB(D)$*, or an atom paired with a set of atoms of* $HB(D)$. *We say that* α *and* β *are* D-isomorphic*, denoted* $\alpha \simeq_D \beta$*, or simply* $\alpha \simeq \beta$ *in case* D *is understood, iff there exists a bijection (i.e., a bijective homomorphism)* $f : dom(\alpha) \rightarrow dom(\beta)$ *such that* $f(\alpha) = f(\beta)$ *(notice that* f *is the identity on* $dom(D)$*).*

Example 5. If $a, b \in dom(D)$ and $\zeta_1, \zeta_2, \zeta_3, \zeta_4 \notin dom(D)$, we have: $P(a, \zeta_1, \zeta_2) \simeq P(a, \zeta_3, \zeta_4)$ and $(P(a, \zeta_3), \{Q(a, \zeta_3), Q(\zeta_3, \zeta_3), R(\zeta_3)\}) \simeq (P(a, \zeta_1), \{Q(a, \zeta_1), Q(a, \zeta_1), R(\zeta_1)\})$. Differently, $P(a, \zeta_1, \zeta_2) \not\simeq P(a, \zeta_1, \zeta_1)$ and $P(a, \zeta_1, \zeta_2) \not\simeq P(\zeta_3, \zeta_1, \zeta_1)$, $\alpha \not\simeq \beta$ meaning $\alpha \simeq \beta$ does not hold. □

Theorem 29. *Let* Σ *be a set of WGTGDs,* D *an instance, and* Q *a Boolean conjunctive query. Determining whether* $D \cup \Sigma \models Q$*, or, equivalently, whether* $chase(\Sigma, D) \models Q$*, is decidable in* EXPTIME *in case of bounded arities, and in 2-*EXPTIME *in general.*

Proof (sketch). This proof sketch is a short version of the proof that can be found in the report (Calì, Gottlob, and Kifer 2008). We first give some preliminary definitions.

Definition 30. *Let* $\underline{a}$ *be an atom. The* canonical renaming $can_{\underline{a}} : dom(\underline{a}) \cup dom(D) \rightarrow \Delta_{\underline{a}} \cup dom(D)$*, where* $\Delta_{\underline{a}}$ *is a set* $\{\xi_1, \ldots, \xi_h\}$ *of labelled nulls, disjunct from* Δ_N*, is a substitution that maps each element of* $dom(D)$ *into itself and maps the* i*-th argument value in lexicographic order of* a *which is not in* $dom(D)$ *to* ξ_i*, for all* i *such that* $1 \leq i \leq h$*, where* h *is the number of values in* $\underline{a}$ *that are not in* $dom(D)$*. If* $S \subseteq cloud(\Sigma, D, \underline{a})$ *(i.e., if* $S \in subclouds(\Sigma, D, \underline{a})$*), then* $can_{\underline{a}}(S)$ *is well-defined and we denote by* $can(\underline{a}, S)$ *the pair* $(can_{\underline{a}}(\underline{a}), can_{\underline{a}}(S))$.

Example 6. If $\underline{a} = G(d, \alpha_1, \alpha_2, \alpha_1)$ where $d \in dom(D)$ and $\alpha_1, \alpha_2 \notin dom(D)$, and if $S = \{P(\alpha_1), R(\alpha_2, \alpha_2), S(\alpha_1, \alpha_2, b)\}$, where $b \in dom(D)$, then $can_{\underline{c}}(\underline{c}) = G(d, \xi_1, \xi_2, \xi_1)$, and $can_c(S) = \{P(\xi_1), R(\xi_2, \xi_2), S(\xi_1, \xi_2, b)\}$. □

We now list a number of important results regarding clouds. Let $|\mathcal{R}|$ be the number of predicates of the schema $\mathcal{R}$, and w the maximum arity of a predicate in $\mathcal{R}$.

Fact 1. $|cloud(\Sigma, D, \underline{a})| \leq |\mathcal{R}| \cdot (|dom(D)| + w)^w$, hence $cloud(\Sigma, D, \underline{a})$ is polynomial in size in case the arity w is fixed, and exponential otherwise.

Fact 2. Let $\underline{a}$ be an atom in the guarded join forest $GCF(\Sigma, D)$. Then each atom $\underline{b}$ of the subtree ${}^{\downarrow}\underline{a}$ of $GCF(\Sigma, D)$ rooted at $\underline{a}$ is obtained by a sequence of chase steps that involve only atoms of $cloud(\Sigma, D, \underline{a})$ and ${}^{\downarrow}\underline{a}$.

Fact 3. If D is an instance for a schema $\mathcal{R}$, Σ a set of WGTDs, $\underline{a}, \underline{b} \in chase(\Sigma, D)$, and $(\underline{a}, cloud(\Sigma, D, \underline{a})) \simeq (\underline{b}, cloud(\Sigma, D, \underline{b}))$, then ${}^{\downarrow}\underline{a} \cup cloud(\Sigma, D, \underline{a}) \simeq {}^{\downarrow}\underline{b} \cup cloud(\Sigma, D, \underline{b})$.

Fact 4. The quotient set $Z = \{(\underline{a}, cloud(\Sigma, D, \underline{a})) \mid \underline{a} \in chase(\Sigma, D)\}/_{\simeq}$ is finite. Its size is at most doubly exponential in $|D \cup \Sigma|$ in case of unbounded arities and at most singly exponential in $|D \cup \Sigma|$ in case of bounded arities.

The above facts allow us to design alternating algorithms for query answering, by simulating the infinite chase by operating on the finite quotient set Z rather than on an infinite number of atoms and their associated clouds. In particular, the equivalence class of each pair $(\underline{a}, cloud(\Sigma, D, \underline{a}))$ will be represented by the canonized pair $can_{\underline{a}}(\underline{a}, cloud(\Sigma, D, \underline{a}))$.

First, we describe an alternating algorithm Acheck(Σ, D, Q), which is executed given the following: *(1)* a set Σ of WGTGDs; *(2)* an instance D; *(3)* an atomic query Q of the form $\exists y_1, \ldots, y_\ell \, P(t_1, t_2, \ldots, t_r)$ where P is a predicate symbol, and the $t_1, \ldots, t_r$, with $r \geq \ell$, are terms (constants or variables) in $dom(D) \cup \{y_1, y_2, \ldots, y_\ell\}$. Acheck decides whether $chase(\Sigma, D) \models Q$, or, equivalently, $D \cup \Sigma \models Q$.

Alternating Algorithm Acheck**.** The alternating algorithm uses existential moves in order to successively guess the vertices of a path in $GCF(\Sigma, D)$ from some atom in D to some atom $\underline{b}$ which is a homomorphic instance of the query atom $P(t_1, t_2, \ldots, t_r)$ to $\underline{b}$ (and fails if such a guess cannot be made). For each intermediate atom $\underline{a}$ of this path, a subcloud S of $\underline{a}$ is guessed as well as a linear ordering $\underline{s}_1 < \underline{s}_2 < \cdots < \underline{s}_k$ on the elements of S. To prove that all atoms of the guessed subcloud S are effectively in $chase(\Sigma, D)$, in universal moves, k auxiliary configurations are generated, where the i-th auxiliary configuration assumes $\underline{s}_1 \ldots \underline{s}_{i-1}$ is already derived and starts an alternating subroutine deriving $\underline{s}_i$. The details of these auxiliary computation are tricky and are described in more detail in (Calì, Gottlob, and Kifer 2008). Whenever an atom $\underline{a}$ and its cloud $\mathcal{C}$ are generated, instead of using $(\underline{a}, \mathcal{C})$, the algorithm works with $can_{\underline{a}}(\underline{a}, \mathcal{C}) = (can_{\underline{a}}(\underline{a}), can_{\underline{a}}(\mathcal{C}))$ and applies $can_{\underline{a}}$ also to the linear order and all other data structures used with each configuration.

In case of bounded arities, canonized clouds are of polynomial size, and thus each configuration uses polynomial space only. Given that APSPACE = EXPTIME, answering atomic queries is in EXPTIME. In case the arity is unbounded, each configuration requires at most exponential space: the problem is then feasible in Alternating EXPSPACE, which is equal to double exponential time, or 2-EXPTIME.

From this complexity bound, we easily derive :

Fact 5. Let Σ be a set of WGTGDs, and let D be an instance for a schema $\mathcal{R}$. Then, computing $chase^{\perp}(\Sigma, D)$ can be done in exponential time in case of bounded arity, and in double exponential time otherwise.

We now show that answering general conjunctive queries is of the same complexity. To this end, we design a nondeterministic algorithm Qcheck such that Qcheck(Σ, D, Q) outputs "true" iff $D \cup \Sigma \models Q$, or, equivalently, iff $chase(\Sigma, D) \models Q$. The algorithm heavily relies on the concept of squid decompositions, and on Lemma 26.

Nondeterministic Algorithm Qcheck**.** Qcheck first computes $chase^{\perp}(\Sigma, D)$. Then it nondeterministically guesses a squid decomposition $\delta = (h, H, T)$ of Q, and a substitution $\theta_0 : dom(H) \rightarrow dom(chase(\Sigma, D))$, such that $\theta_0(H) \subseteq chase^{\perp}(\Sigma, D)$. Next Qcheck tests whether θ_0 can be extended to a homomorphism θ such that $\theta(T) \subseteq chase^{+}(\Sigma, D)$. Note that, by Lemma 26, this is equivalent to $chase(\Sigma, D) \models Q$. Such a θ exists iff for each connected subgraph t of $\theta(T)$, there is a homomorphism θ_t that leaves all elements of $dom(D)$ unaltered such that $\theta_t(t) \subseteq chase^{+}(\Sigma, D)$. The Qcheck algorithm thus identifies the connected components of $\theta(T)$. Each such component is an acyclic conjunctive query that can be written as a join tree t. For each such join tree t, Qcheck now tests whether there exists a homomorphism θ_t (which, we remind, is the identity on D) such that $\theta_t(t) \subseteq chase^{+}(\Sigma, D)$. This is done by the subroutine Tcheck, which takes as arguments the TGDs, the database instance, and the subgraph t of T; how Tcheck(Σ, D, t) is executed is described below. Qcheck succeeds iff the exit is positive for each component.

The correctness of Qcheck follows from Lemma 26. Given the nondeterministic guess of a squid decomposition, the complexity of Qcheck is in NP^X , i.e., NP with an oracle in X, where X is a complexity class that is sufficiently powerful for computing $chase^{\perp}(\Sigma, D)$, and performing the tests Tcheck(Σ, D, t). We finally describe Tcheck.

Nondeterministic Algorithm Tcheck**.** Tcheck works essentially like Acheck, but instead of nondeterministically constructing a main configuration path of the configuration tree such that eventually some atom matches the unique query atom, Tcheck nondeterministically constructs a main configuration subtree τ of the configuration tree, such that eventually all atoms of the join tree t will be consistently translated into some vertices of τ. In addition to the data structures of Acheck, the main configurations of Tcheck maintain a pointer π and a substitution θ. In the initial main configuration, π points to the root of t and θ is empty. In general, the pointer π of each main configuration C points to some atom $\underline{\pi}^{\uparrow}$ of t, which has not yet been matched. The algorithm attempts to expand this configuration by successively guessing a subtree of configurations mimicking a suitable subtree of $GCF(\Sigma, D)$ that satisfies the subquery of t rooted at $\underline{\pi}^{\uparrow}$. If Tcheck gets to an atom $\underline{a}$ such that there is a homomorphism σ such that $\sigma(\theta(\underline{\pi}^{\uparrow})) = \underline{a}^{\uparrow}$, then, in case $\underline{\pi}^{\uparrow}$ has no children, the current configuration turns into an accepting one. Otherwise, (via a universal move) for each child atom $\underline{d}$ of $\underline{a}^{\uparrow}$ in t, Tcheck creates a separate configuration with values $\underline{a}^{\uparrow} = \underline{d}$ and $\theta = \sigma \circ \theta$.

For the complexity of Qcheck, note that in case the arity is bounded, Tcheck runs in APSPACE = EXPTIME, and computing $chase^{\perp}(\Sigma, D)$ is in EXPTIME (Calì, Gottlob, and Kifer 2008). Thus, Qcheck runs in time NP$^{\text{EXPTIME}}$ = EXPTIME. In case of unbounded arities, both computing $chase^{\perp}(\Sigma, D)$ and running Tcheck are in 2-EXPTIME, therefore Qcheck runs in time NP$^{\text{2-EXPTIME}}$ = 2-EXPTIME. □

By combining Theorems 24 and 29 we get the following complexity characterization for reasoning under WGTGDs.

Theorem 31. *Let Σ be a set of WGTGDs, let D be an instance, and let Q be a Boolean conjunctive query. Determining whether $D \cup \Sigma \models Q$, or, 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* 2-EXPTIME *complete in general. The same completeness results hold for query containment under WGTGDs.*

Guarded TGDs

Theorem 32. *Let Σ be a set of GTGDs over a schema $\mathcal{R}$, and let D be an instance for $\mathcal{R}$. Let, moreover, w denote the maximum arity of any predicate appearing in $\mathcal{R}$, and let $|\mathcal{R}|$ denote the total number of predicate symbols. Then:*

(1) Computing $chase^{\perp}(\Sigma, D)$ can be done in PTIME *if both w and $|\mathcal{R}|$ are bounded, and thus also in case of a fixed set Σ. This problem is in* EXPTIME *in case w is bounded, and in 2-*EXPTIME *otherwise.*

(2) If Q is an atomic (Boolean) query, then checking whether $\Sigma \cup D \models Q$ or, equivalently, $chase(\Sigma, D) \models Q$, is PTIME*-complete in case both w and $|\mathcal{R}|$ are bounded, and remains* PTIME*-complete even in case Σ is fixed. This problem is* EXPTIME*-complete if w is bounded and 2-*EXPTIME*-complete in general. It remains 2-*EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

(3) If Q is a general Boolean conjunctive query, checking whether $\Sigma \cup D \models Q$ or, equivalently $chase(\Sigma, D) \models Q$ is NP*-complete in case both w and $|\mathcal{R}|$ are bounded, and thus also in case of a fixed set Σ. Checking whether $chase(\Sigma, D) \models Q$ is* EXPTIME*-complete if w is bounded and* 2-EXPTIME*-complete in general. It remains* 2-EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

(4) Query containment under GTGDs is NP*-complete if both w and $|\mathcal{R}|$ are bounded, and even in case the set Σ of GTGDs is fixed.*

(5) Query containment under GTGDs is EXPTIME*-complete if w is bounded and 2-*EXPTIME*-complete in general. It remains* 2-EXPTIME*-complete even when $|\mathcal{R}|$ is bounded.*

Proof (sketch). The hardness results are obtained with a straightforward modification of the proof of Theorem 24. The membership results are proved exactly as those for WGTGDs (Theorem 29), except that, instead of using the notion of cloud, we use the similar notion of *restricted cloud*. The *restricted cloud* $rcloud(\Sigma, D, \underline{a})$ of an atom $\underline{a} \in chase(\Sigma, D)$ is the set of all atoms $\underline{b} \in chase(\Sigma, D)$ such that $dom(\underline{b}) \subseteq dom(\underline{a})$. We thus use algorithms that differ from the respective original algorithms only in that they use restricted clouds instead of clouds. A more detailed proof is given in (Calì, Gottlob, and Kifer 2008). □

Note that one of the main results of Johnson and Klug (Johnson and Klug 1984), namely, that query containment under inclusion dependencies of bounded arities is NP-complete, is a special case of Item *(3)* of Theorem 32.

Application

In this section we show the application of our subset of Datalog$^\exists$ to a formalism called F-logic Lite; we show that query answering and containment under F-logic Lite rules are NP-complete.

F-logic Lite is a smaller but expressive version of F-logic, a well-known formalism introduced for object-oriented deductive databases. We refer the reader to refer the reader to (Calì and Kifer 2006) for details about F-logic Lite. Roughly, with respect to F-Logic, F-logic Lite] excludes negation and default inheritance, and allows only a limited form of cardinality constraints.

We now briefly show how to encode F-logic Lite using Datalog$^\exists$ rules that we denote with Σ_{FLL}, with $\Sigma_{FLL} = \{\rho_i\}_{1 \leq i \leq 12}$.

(1) $\mathsf{member}(V, T) \leftarrow \mathsf{type}(O, A, T), \mathsf{data}(O, A, V)$.
(2) $\mathsf{sub}(C_1, C_2) \leftarrow \mathsf{sub}(C_1, C_3), \mathsf{sub}(C_3, C_2)$.
(3) $\mathsf{member}(O, C_1) \leftarrow \mathsf{member}(O, C), \mathsf{sub}(C, C_1)$.
(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.
(5) $\mathsf{data}(O, A, V) \leftarrow \mathsf{mandatory}(A, O)$.
Note that this is a TGD with an existential variable in the head (variable V; quantifiers are omitted).
(6) $\mathsf{type}(O, A, T) \leftarrow \mathsf{member}(O, C), \mathsf{type}(C, A, T)$.
(7) $\mathsf{type}(C, A, T) \leftarrow \mathsf{sub}(C, C_1), \mathsf{type}(C_1, A, T)$.
(8) $\mathsf{type}(C, A, T) \leftarrow \mathsf{type}(C, A, T_1),\ \mathsf{sub}(T_1, T)$.
(9) $\mathsf{mandatory}(A, C) \leftarrow \mathsf{sub}(C, C_1), \mathsf{mandatory}(A, C_1)$.
(10) $\mathsf{mandatory}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{mandatory}(A, C)$.
(11) $\mathsf{funct}(A, C) \leftarrow \mathsf{sub}(C, C_1),\ \mathsf{funct}(A, C_1)$.
(12) $\mathsf{funct}(A, O) \leftarrow \mathsf{member}(O, C), \mathsf{funct}(A, C)$.

It can be easlity shown that the only EGD in the above Datalog$^\exists$ rules does not actually interact with the TGDs, and therefore we can ignore it (Calì, Gottlob, and Kifer 2008).

We now prove the complexity results.

Theorem 33. *Conjunctive query answering under F-logic Lite rules is* NP*-hard.*

Proof (sketch). The proof is by reduction from the 3-COLORABILITY problem. Encode a graph $G = (V, E)$ as a conjunctive query Q which, for each edge (v_i, v_j) in E, has two atoms $\mathsf{data}(X, V_i, V_j)$ and $\mathsf{data}(X, V_j, V_i)$, where X is a unique, fixed variable. Let D be the instance $D = \{\mathsf{data}(o, r, g),\ \mathsf{data}(o, g, r),\ \mathsf{data}(o, r, b),\ \mathsf{data}(o, b, r),\ \mathsf{data}(o, g, b),\ \mathsf{data}(o, b, g)\}$. Then, G is three-colorable iff $D \models Q$, which is the case iff $D \cup \Sigma_{FLL} \models Q$. The transformation from G to (Q, D) is obviously polynomial. This proves the claim. □

Theorem 34. *Conjunctive query answering under F-logic Lite rules is in* NP.

Proof (sketch). As mentioned before, we can ignore the only EGD in Σ_{FLL}, since it does not interfere with query answering; details are found in the full version (Calì, Gottlob, and Kifer 2008). Let us denote with Σ'_{FLL} the set of Datalog$^\exists$ resulting from Σ_{FLL} by eliminating rule ρ_4, i.e., let $\Sigma'_{FLL} = \Sigma_{FLL} - \{\rho_4\}$. To establish membership in NP, it is sufficient to show that:

(1) Σ'_{FLL} is weakly guarded.
(2) Σ'_{FLL} is such that, for every instance D, there are, up to D-isomorphisms, polynomially many clouds; more precisely, for every instance D there exists a polynomial pol such that $|clouds(\Sigma, D)/_{\simeq}| \leq pol(|D|)$.
(3) There is a polynomial $pol'(\cdot)$ such that for each instance D and for each atom a: *(3.1)* if $\underline{a} \in D$, then $cloud(\Sigma, D, \underline{a})$ can be computed in time $pol'(|D|)$, and *(3.2)* if $\underline{a} \notin D$, then $cloud(\Sigma, D, \underline{a})$ can be computed in time $pol(|D|)$ from D, a, and $cloud(\Sigma, D, \underline{b})$, where $\underline{b}$ is the predecessor of $\underline{a}$ in $GCF(\Sigma, D)$.

(1) is readily seeen: the affected positions are the following: $\mathsf{data}[3]$, $\mathsf{member}[1]$, $\mathsf{type}[1]$, $\mathsf{mandatory}[2]$, $\mathsf{funct}[2]$, $\mathsf{data}[1]$. It is easy to see that every rule of Σ'_{FLL} is weakly guarded, and thus Σ_{FLL} is weakly guarded.

Now let us sketch *(2)*. Let $\Sigma^{full}_{FLL} = \Sigma'_{FLL} - \{\rho_5\}$, i.e., the set of all TGDs of Σ'_{FLL} but ρ_5. These are all full TGDs and their application does not alter the domain. We have $chase(\Sigma'_{FLL}, D) = chase(\Sigma'_{FLL}, chase(\Sigma^{full}_{FLL}, D))$. Let us now have a closer look at $D^+ = chase(\Sigma^{full}_{FLL}, D)$. Clearly, $dom(D^+) = dom(D)$. For each predicate symbol P, let $Rel(P)$ denote the relation consisting of all P-tuples in D^+. Let Ω be the family of all relations that can be obtained from any of the relations $Rel(P)$ by performing an arbitrary selection followed by some projection (we forbid disjunctions in the selection predicate). For example, assume $c, d \in dom(D)$; then, $Rel(\mathsf{data})$ will give rise to relations $\pi_{1,2}(\sigma\{1 = c\}Rel(\mathsf{data}))$, and to $\pi_2(\sigma\{1 = d \wedge 3 = c\}Rel(\mathsf{data}))$, and so on, where the numbers are attribute identifiers (the notation here should be self-explanatory). Given that D^+ is of size polynomial in D and that the maximum arity of any relation $Rel(p)$ is 3, the set Ω is of size polynomial in D^+ and thus polynomial in D. It can now be shown that Ω is preserved in a precise sense, when going to the final result $chase(\Sigma'_{FLL}, D^+)$. In particular, for each relation $Rel'(P)$ corresponding to predicate P in the final chase result, when performing a selection on $Rel'(P)$ that assigns fixed values $\notin dom(D)$ to one or more attributes, and projecting on the other columns, the set of all tuples of $dom(D)$-elements in the result is a relation in Ω. For example, assume that v_5 is a specific labeled null, then the set of all $T \in dom(D)$ such that $\mathsf{member}(v_5, T)$ is an element of the final result is a set in Ω; similarly, if v_7 and v_8 are new values, the set of all values A such that $\mathsf{data}(v_7, A, v_8)$ is a relation in Ω. It is easy to see that from this it follows that Σ'_{FLL} satisfies *(2)*. In fact, all possible clouds are determined by the polynomially many ways of choosing at most three elements of Ω for each predicate. The proof of the preservation property can be done by induction on the i-th new labeled null added. Roughly, for each such labeled null, created by rule ρ_5, we just analyze which sets of values (or tuples) are attached to it via rules ρ_4, then ρ_6, ρ_7, ρ_8, ρ_{10}, and so on, and conclude that these sets were all already present at the next lower level, and thus, by induction hypothesis, are in Ω.

Condition *(3)* can straightforwardly proved by similar arguments. □

From Theorems 33 and 34 we immediately get:

Corollary 35. *Conjunctive query answering under F-logic Lite rules is* NP*-complete.*

Conclusions

In this paper we identified a large and non-trivial class of relational constraints, namely *tuple-* and *equality-generating* dependencies, for which the problems of conjunctive query answering and containment are decidable, and provided the relevant complexity results. Applications of our results include databases and Knowledge Representation. For instance, our results subsume the classical work of Johnson and Klug (Johnson and Klug 1984) as well as (Calì and Kifer 2006).

Future work. We intend to investigate query answering (and containment) under WGTGDs in the case of finite models (*finite implication problem*). Some interesting results (Rosati 2006) exist in this respect, and they may carry over to GTGDs or WGTGDs,

A related previous approach to guarded logic programming is *guarded open answer set programming* (Heymans, Nieuwenborgh, and Vermeir 2005). It is easy to see that a set of GTGDs can be interpreted as a guarded answer set program as defined in (Heymans, Nieuwenborgh, and Vermeir 2005), but that guarded answer set programs are, in general, more expressive than GTGDs, for example, because they allow for negation. Investigating the decidability and complexity of query answering (and containment) under more expressive classes of constraints, capable of subsuming, for instance, the results of (Calì 2007) and (Heymans, Nieuwenborgh, and Vermeir 2005), is the subject of our future work. We also plan to investigate the same problem in the case of finite models.

References

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

Beeri, C.; Fagin, R.; Maier, D.; Mendelzon, A. O.; Ullman, J. D.; and Yannakakis, M. 1981. Properties of acyclic database schemes. In *STOC*, 355–362.

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

Calì, A.; Gottlob, G.; and Kifer, M. 2008. Extending datalog for terminological reasoning. Unpublished technical report, available from `http://www.andreacali.com`.

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

Calì, A. 2007. Querying incomplete data with logic programs: ER strikes back. In *ER 2007*, 245–260.

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.; De Giacomo, G.; and Lenzerini, M. 1998. On the decidability of query containment under constraints. In *PODS 1998*, 149–158.

Calvanese, D.; De Giacomo, G.; and Lenzerini., M. 2002. Description logics for information integration. In *Computational Logic: Logic Programming and Beyond*, volume 2408 of *LNCS*. Springer. 41–60.

Chandra, A., and Merlin, P. 1977. Optimal implementation of conjunctive queries in relational data bases. In *STOC 1977*, 77–90.

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., and Tannen, V. 2003. Reformulation of xml queries and constraints. In *ICDT*, 225–241.

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.

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2008. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. *J. of Artificial Intelligence Research* 31:151–198.

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

Gottlob, G., and Nash, A. 2006. Data exchange: computing cores in polynomial time. In *PODS*, 40–49.

Gottlob, G.; Leone, N.; and Scarcello, F. 2002. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.* 64(3):579–627.

Heymans, S.; Nieuwenborgh, D. V.; and Vermeir, D. 2005. Guarded open answer set programming. In *LPNMR 2005*, 92–104.

Johnson, D., and Klug, A. 1984. Testing containment of conjunctive queries under functional and inclusion dependencies. *JCSS* 28:167–189.

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

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

Millstein, T.; Levy, A.; and Friedman, M. 2000. Query containment for data integration systems. In *PODS 2000*, 67–75.

Nash, A.; Deutsch, A.; and Remmel, J. 2006. Data exchange, data integration, and chase. Technical Report CS2006-0859, UCSD.

Qian, X. 1996. Query folding. In *ICDE*, 48–55.

Rabin, M. 1969. Decidability of Second-Order Theories and Automata on Infinite Trees. *Transactions of the American Mathematical Society* 141(1-35):4.

Rosati, R. 2006. On the decidability and finite controllability of query processing in databases with incomplete information. In *PODS 2006*, 356–365.

Rosati, R. 2007. On conjunctive query answering in EL. In *20th International Workshop on Description Logics (DL-2007)*. CEUR Electronic Workshop Proceedings.

Simkus, M., and Eiter, T. 2007. DNC: Decidable nonmonotonic disjunctive logic programs with function symbols. In *LPAR 2007*, 514–530.