Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence

New Expressive Languages for Ontological Query Answering

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

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

{*georg.gottlob,andreas.pieris*}*@comlab.ox.ac.uk*
andrea@dcs.bbk.ac.uk

Abstract

Ontology-based data access is a powerful form of extending database technology, where a classical extensional database (EDB) is enhanced by an ontology that generates new intensional knowledge which may contribute to answer a query. Recently, the Datalog$^\pm$ family of ontology languages was introduced; in Datalog$^\pm$, rules are tuple-generating dependencies (TGDs), i.e., Datalog rules with the possibility of having existentially-quantified variables in the head. In this paper we introduce a novel Datalog$^\pm$ language, namely *sticky* sets of TGDs, which allows for a wide class of joins in the body, while enjoying at the same time a low query-answering complexity. We establish complexity results for answering conjunctive queries under sticky sets of TGDs, showing, in particular, that ontological conjunctive queries can be compiled into first-order and thus SQL queries over the given EDB instance. We also show some extensions of sticky sets of TGDs, and how functional dependencies and so-called negative constraints can be added to a sticky set of TGDs without increasing the complexity of query answering. Our language thus properly generalizes both classical database constraints and most widespread tractable description logics.

Introduction

Ontological Database Management Systems. We are currently witnessing the rise of a new type of database management systems equipped with advanced reasoning and query answering mechanisms. The necessity of combining ontological reasoning and description logics (DLs) with database techniques has emerged in both the DL and database communities. In ontology-enhanced database systems, an extensional database D (also called ABox) is combined with an ontological theory Σ (also called TBox) describing rules and constraints which derive new intensional data from the extensional data. Queries are not just answered against the database D, but against the logical theory $D \cup \Sigma$. Thus, given a *conjunctive query (CQ)* of the form $q(\mathbf{X}) \leftarrow body(\mathbf{X}, \mathbf{Y})$, with output variables $\mathbf{X}$, its answer in the ontological database consists of all tuples $\mathbf{t}$ of constants such that $D \cup \Sigma \models \exists \mathbf{u}\, body(\mathbf{t}, \mathbf{u})$.

Interestingly, the well-known notion of *chase* (Maier, Mendelzon, and Sagiv 1979; Johnson and Klug 1984; Fagin et al. 2005; Deutsch, Nash, and Remmel 2008) of a database D w.r.t. an ontology Σ, denoted as $chase(D, \Sigma)$, is a useful tool for query answering. In particular, the answers to a query q against $D \cup \Sigma$ coincide with the answers to q over $chase(D, \Sigma)$ (see, e.g., (Deutsch, Nash, and Remmel 2008)). Roughly speaking, the chase adds new tuples to D (possibly with labeled nulls to represent unknown values) until $chase(D, \Sigma)$ satisfies all the constraints of Σ.

Research Challenges. A critical issue is that the chase of a database w.r.t. an ontology is (in general) infinite, and thus not explicitly computable.

Example 1 *Consider a database* $D = \{person(john)\}$*, and the ontological theory* Σ *consisting of the constraints*

$$\begin{array}{rcl} person & \rightarrow & \exists Y\, father(Y, X), \\ father(X, Y) & \rightarrow & person(X), \end{array}$$

stating that every person has a father, who is himself a person. Then, $chase(D, \Sigma)$ *is the infinite set of atoms* $D \cup \{father(z_1, john), person(z_1), father(z_1, z_2), person(z_2), father(z_2, z_3), pesron(z_3), \ldots\}$*, where each* z_i *is a labeled null value.* ■

Procedures for effectively answering queries, even when the chase is infinite, were first developed in the database context for the class of *inclusion dependencies (IDs)* (Johnson and Klug 1984). However, IDs alone are not expressive enough to capture some popular ontological constraints. Currently, there is a trend towards highly scalable procedures for query answering over ontologies. A significant step forward in this direction was the introduction of the *DL-Lite* family of DLs (Calvanese et al. 2007; Poggi et al. 2008).

Recently, the Datalog$^\pm$ family (Calì, Gottlob, and Lukasiewicz 2009) has been proposed, with the purpose of providing tractable query answering algorithms for more general ontology languages. Datalog$^\pm$ languages are based on Datalog rules that allow for the existential quantification of variables in the head of the rules, in the same fashion as Datalog with *value invention* (Mailharrow 1998; Cabibbo 1998). As observed in (Patel-Schneider and Horrocks 2007), the lack of value invention makes plain Datalog not very well suited for ontological reasoning. The basic Datalog$^\pm$ rules are so-called *tuple-generating dependencies (TGDs)* (Beeri and Vardi 1981).

Given that CQ answering under TGDs is undecidable (Beeri and Vardi 1981), some syntactic restrictions are needed. Two fundamental restriction paradigms have been studied so far for ensuring decidability, and in the case of *data complexity*, i.e., the complexity w.r.t. the data only, also tractability of query answering: *weak acyclicity* (Fagin et al. 2005) and *guardedness* (Calì, Gottlob, and Kifer 2008). The chase under weakly-acyclic sets of TGDs always terminates, and thus a finite instance C is constructed. Obviously query answering over C is decidable. The decidability of query answering under guarded TGDs, that is, TGDs with an atom in their body, the so-called *guard*, that contains all the body-variables, follows from the fact that the (possibly infinite) instance constructed by the chase has bounded treewidth.

Summary of Contributions. Our goal is to identify new expressive fragments of Datalog$^{\pm}$, which are based on some new decidability paradigm. As a central new class we propose the class of *sticky* sets of TGDs, which are sets of TGDs with a restriction on multiple occurrences of variables in the rule bodies. An informal explanation of stickiness, based on the chase, is as follows. First, stickiness requires that every TGD σ that has a double occurrence of a variable V in its body, V occurs in every head-atom of σ. Furthermore, whenever such a TGD is triggered during the chase and a new atom $\underline{a}$ is obtained that has a value v in place of the variable V, then the value v occurs in every atom obtained by a chase derivation that involves $\underline{a}$. In other words, every value that arises in a new atom $\underline{a}$ through a join in a TGD-body must be present in all further atoms derived from $\underline{a}$. We will define stickiness formally by an efficiently testable condition involving variable marking.

Moreover, we briefly discuss some extended languages obtained by combining stickiness with known decidability paradigms, in particular with *linearity* (Calì, Gottlob, and Lukasiewicz 2009) (linear TGDs are rules with just one body-atom, and thus trivially guarded) and weak acyclicity, without altering the complexity of query answering. By combining sticky sets of TGDs with linear TGDs (resp., weakly-acyclic sets of TGDs) we obtain the class of *sticky-join* (resp. *weakly-sticky*) sets of TGDs.

We then discuss how we can combine TGDs with *negative constraints* of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\bot$ is the truth constant *false*, and *functional dependencies* (see, e.g., (Abiteboul, Hull, and Vianu 1995)), without increasing the complexity of query answering. This allows us to show that our work properly generalizes the main DL-Lite languages, i.e., DL-Lite$_{\mathcal{F}}$, DL-Lite$_{\mathcal{R}}$ and DL-Lite$_{\mathcal{A}}$ (Calvanese et al. 2007; Poggi et al. 2008). Moreover, it is possible to show that these DLs can be extended with *concept product*, a construct introduced in (Rudolph, Krötzsch, and Hitzler 2008) which, through rules of the form $p(X), q(Y) \rightarrow r(X, Y)$, expresses the cartesian product of two concepts (unary relations) p and q, without altering the complexity of query answering.

Our main complexity results are summarized in Figure 1. Recall that the data complexity of query answering is calculated taking only the database as input, while the query and the set of TGDs are considered fixed. The *combined complexity* is the complexity calculated considering as part of the input, together with the database, also the query and the set of TGDs. The results in this paper appeared in (Calì, Gottlob, and Pieris 2010a) and (Calì, Gottlob, and Pieris 2010b).

Language	**Data Complexity**	**Comb. Complexity**
STGDs	in AC$_0$	EXPTIME-complete
SJTGDs	in AC$_0$	EXPTIME-complete
WSTGDs	PTIME-complete	2EXPTIME-complete

Figure 1: Complexity of query answering. STGDs, SJTGDs and WSTGDs are abbreviations for sticky, sticky-join and weakly-sticky sets of TGDs, respectively.

Preliminaries

We introduce the following pairwise disjoint sets of symbols: *(i)* an infinite set Γ of *constants*, which constitute the "normal" domain of a database, *(ii)* an infinite set Γ_N of *labeled nulls*, which will be used as "fresh" Skolem terms, and *(iii)* a set Γ_V of *variables*, used in queries and dependencies. Different nulls may represent the same value, and therefore can be seen as variables. However, different constants represent different objects (*unique name assumption*). Sets of variables (or sequences, with a slight abuse of notation) are denoted as $\mathbf{X}$, with $\mathbf{X} = X_1, \ldots, X_k$, for some $k > 0$.

We will refer to a relational schema $\mathcal{R}$, assuming that database instances (or simply databases), queries and dependencies use predicates of $\mathcal{R}$. We assume the reader is familiar with the relational model. We denote 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. *Boolean CQs (BCQs)* are those with no variables in the head (i.e., with arity zero). Database instances will be constructed with values from $\Gamma \cup \Gamma_N$, and they will be possibly infinite.

An *atom* is an atomic formula of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate (also called relation name), and each term t_i is either a constant or a variable. An atom is called *ground* if all of its terms are constants of Γ. A *position* $p[i]$ of a relational schema is identified by a relational predicate p and its i-th attribute, where the integer i can take values between zero and the arity of p.

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

A major issue in this work are database dependencies. In the relational model, one of the most important classes of dependencies are *tuple-generating dependencies (TGDs)*, which are a generalization of inclusion dependencies. A TGD σ over a schema $\mathcal{R}$ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called *body* and *head* of the TGD, respectively. Such a dependency is satisfied by a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\varphi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there exists an extension h' of h (i.e., $h' \supseteq h$) that maps the atoms of $\psi(\mathbf{X}, \mathbf{Z})$ to D. To simplify the notation, we will omit the universal quantifiers in TGDs.

We now define the notion of *query answering* under TGDs. Given a database D and a set Σ of TGDs, we define the set of instances B such that $B \models D \cup \Sigma$ as the set of *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$. The *answers* to a CQ q on D w.r.t. Σ, denoted as $ans(q, D, \Sigma)$, is the set of tuples of constants $\mathbf{t}$ such that for every $B \in mods(D, \Sigma)$, it holds that $\mathbf{t} \in q(B)$. For a BCQ q, if the empty tuple $\langle\rangle$ belongs to $ans(q, D, \Sigma)$, then we write $D \cup \Sigma \models q$.

The *chase* procedure was introduced in order to enable checking implication of dependencies (Maier, Mendelzon, and Sagiv 1979), and later for checking query containment (Johnson and Klug 1984). Informally, the chase procedure is a process of repairing a database w.r.t. a set of database dependencies, by adding tuples that may contain labeled nulls to denote unknown values. By abuse of terminology, with "chase" we refer both to the chase procedure and to its output. We do not describe the chase in detail here, and we refer the reader, for instance, to (Calì, Gottlob, and Pieris 2010a). The (possibly infinite) chase of a database D w.r.t. a set Σ of TGDs, denoted as $chase(D, \Sigma)$, is a *universal model* of D w.r.t. Σ, i.e., for each instance $B \in mods(D, \Sigma)$, there exists a homomorphism that maps $chase(D, \Sigma)$ to B (Fagin et al. 2005; Deutsch, Nash, and Remmel 2008). Using this fact it can be shown that for each BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

We recall that BCQ answering is LOGSPACE-equivalent to CQ answering (Calì, Gottlob, and Kifer 2008). Moreover, it is easy to see that the query output tuple problem (as a decision version of CQ answering) and BCQ answering are mutually AC_0-reducible. We shall henceforth consider BCQ answering only.

Sticky Datalog$^{\pm}$

In this section we present a new decidability paradigm called *stickiness*. More precisely, the class of *sticky* sets of TGDs (which forms the language *sticky Datalog*$^{\pm}$) is introduced.

Formal Definition. The definition of sticky sets of TGDs is based heavily on a variable-marking procedure called SMarking. This procedure accepts as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of Σ. Formally, SMarking(Σ) works as follows. First, we apply the so-called *initial marking* step: for each TGD $\sigma \in \Sigma$, and for each variable V in $body(\sigma)$, if there exists an atom $\underline{a}$ in $head(\sigma)$ such that V does not appear in $\underline{a}$, then we mark each occurrence of V in $body(\sigma)$. Then, we apply exhaustively (i.e., until a fixpoint is reached) the *propagation* step: for each pair of TGDs $\langle\sigma, \sigma'\rangle \in \Sigma \times \Sigma$ (including the case $\sigma = \sigma'$), if a universally quantified variable V occurs in $head(\sigma)$ at positions $\pi_1, \ldots, \pi_m$, for $m \geqslant 1$, and there exists an atom $\underline{a} \in body(\sigma')$ such that at each position $\pi_1, \ldots, \pi_m$ a marked variable occurs, then we mark each occurrence of V in $body(\sigma)$. We are now ready to give the formal definition of sticky sets of TGDs.

Definition 1 *Consider a set Σ of TGDs over a schema $\mathcal{R}$. Σ is* sticky *iff there is no TGD $\sigma \in$ SMarking(Σ) such that a marked variable occurs in $body(\sigma)$ more than once.*

Example 2 *Consider the following set of TGDs. We mark the body-variables, according to the* SMarking *procedure, with hat, e.g., $\hat{X}$:*

$$\begin{array}{rcl} p(\hat{X}, \hat{Y}) & \rightarrow & \exists Z\, p(Y, Z) \\ p(X, \hat{Y}) & \rightarrow & q(X) \\ q(X), q(Y) & \rightarrow & r(X, Y) \\ p(X, \hat{Y}), p(\hat{Z}, X) & \rightarrow & q(X). \end{array}$$

The only variable that occurs more than once in the body of a TGD, i.e., the variable X in the body of the last TGD, is non-marked. Therefore, Σ is a sticky set. ■

Consider the simple database $D = \{p(a, a)\}$ and the set Σ of TGDs given in the above example. It is easy to verify that $chase(D, \Sigma)$ is infinite, which implies that Σ is not weakly-acyclic. In fact, the first rule of Σ by itself violates weak acyclicity. Moreover, the extension of the relation r in $chase(D, \Sigma)$ is an infinite clique, and thus $chase(D, \Sigma)$ has infinite treewidth. This implies that Σ in non-guarded; in particular, the third rule of Σ is a prime example of non-guardedness. Actually, stickiness is incomparable to weak acyclicity and guardedness.

It is straightforward to see that the problem of identifying sticky sets of TGDs, that is, given a set Σ of TGDs, decide whether Σ is sticky, is feasible in polynomial time. This follows by observing that at each application of the propagation step, during the execution of the SMarking procedure, at least one body-variable is marked. Thus, after polynomially many steps the SMarking procedure terminates.

Sticky Property. It is interesting to see that the chase constructed under a sticky set of TGDs enjoys a syntactic property called *sticky property*. Before we proceed further let us introduce a useful technical notion. Given a database D for a schema $\mathcal{R}$ and a set Σ of TGDs over $\mathcal{R}$, we define the binary relation $\xrightarrow{D,\Sigma}$ as follows. Suppose that in the construction of $chase(D, \Sigma)$ we apply a TGD $\sigma \in \Sigma$, with homomorphism h, and the atom $\underline{a}$ is generated. Then, for each $\underline{b} \in body(\sigma)$, we have $h(\underline{b}) \xrightarrow{D,\Sigma} \underline{a}$.

Definition 2 *Consider a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. Suppose that in the construction of $chase(D, \Sigma)$ we apply $\sigma \in \Sigma$, with homomorphism h, that has a variable V appearing more than once in its body, and the atoms $\underline{a}_1, \ldots, \underline{a}_k$, for $k \geqslant 1$, are generated. We say that $chase(D, \Sigma)$ has the* sticky property *iff, for each atom $\underline{a} \in \{\underline{a}_1, \ldots, \underline{a}_k\}$, $h(V)$ occurs in $\underline{a}$, and also in every atom $\underline{b}$ such that $\langle\underline{a}, \underline{b}\rangle$ is in the transitive closure of $\xrightarrow{D,\Sigma}$.*

The following theorem implies that stickiness is a sufficient condition for the sticky property of the chase.

Theorem 1 *Consider a set Σ of TGDs over a schema $\mathcal{R}$. If Σ is sticky, then $chase(D, \Sigma)$ enjoys the sticky property, for every database D for $\mathcal{R}$.*

Usefulness of Sticky Sets of TGDs. Together with standard constructs such as keys and negative constraints, sticky sets of TGDs can express whatever is expressible in the well-known DL-Lite versions DL-Lite$_{\mathcal{A}}$, DL-Lite$_{\mathcal{R}}$, and DL-Lite$_{\mathcal{F}}$, and actually more. Therefore, wherever DL-Lite is appropriate for modeling ontological constraints, so are, *a fortiori*, sticky sets of TGDs.

Sticky sets of TGDs can be used with relational database schemas of *arbitrary arity*. Note that the above mentioned DL-Lite languages are, as most description logics, usable for binary relations only. In this sense, sticky sets of TGDs are closer to the paradigm of database constraints, that make no assumption about arity. In particular, sticky sets of TGDs generalize the class of *inclusion dependencies*, while DL-Lite constraints generalize unary and binary inclusion dependencies only. DLR-Lite (Calvanese et al. 2006), a recent generalization of DL-Lite to arbitrary arities, is also strictly generalized by sticky sets of TGDs.

Sticky sets of TGDs can express constraints and rules involving *joins*. We are convinced that the overwhelming number of real-life situations involving such constraints can be effectively modeled by sticky sets of TGDs. Of course, since query answering under TGDs involving joins is undecidable in general, we somehow needed to restrict the interaction of TGDs, when joins are used. But we believe that the restriction imposed by stickiness is a very mild one. Only rather contorted TGDs that seem not to occur too often in real life violate it. For example, each singleton multivalued dependency is sticky, as are many realistic sets of multivalued dependencies.

Sticky sets of TGDs very significantly generalize some other constructs that were introduced to enhance DLs with joins. Noticeably, Rudolph et al. (Rudolph, Krötzsch, and Hitzler 2008) introduce the *concept product*, which, through rules of the form $p(X) \wedge q(Y) \rightarrow r(X,Y)$, expresses the cartesian product of two concepts (unary relations) p and q. This way one can, for instance, express that all elephants are bigger than all mice: $elephant(X) \wedge mouse(Y) \rightarrow bigger\_than(X,Y)$. Several convincing arguments are given in (Rudolph, Krötzsch, and Hitzler 2008) to support the introduction of the concept product in DLs. Note that concept products are very special cases of sticky sets of TGDs.

Data Complexity. Let us know study the data complexity of query answering under sticky sets of TGDs. A class $\mathcal{C}$ of TGDs is *first-order rewritable*, henceforth abbreviated as *FO-rewritable*, iff for every set Σ of TGDs in $\mathcal{C}$, and for every BCQ q, it is possible to construct a first-order query q_Σ such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$ (Calvanese et al. 2007). Since answering first-order queries is in the class AC_0[1] in data complexity (Vardi 1995), it immediately follows that for FO-rewritable TGDs, query answering is in AC_0 in data complexity.

To establish FO-rewritability of sticky sets of TGDs, we first prove that they enjoy the so-called *bounded derivation-depth property (BDDP)* (Calì, Gottlob, and Lukasiewicz 2009). The *derivation depth* of an atom in $chase(D,\Sigma)$ is defined inductively as follows. The atoms of D have derivation depth zero. Let $\underline{a}$ be an atom obtained by the atoms $\underline{a}_1,\dots,\underline{a}_n$ through the application of a TGD $\sigma \in \Sigma$. Let also d be the maximum depth of an atom among $\underline{a}_1,\dots,\underline{a}_n$. Then, the derivation depth of $\underline{a}$ is $d+1$. We denote by $chase^k(D,\Sigma)$ the initial part of the chase constituted by atoms of derivation depth at most k.

Definition 3 *A class $\mathcal{C}$ of TGDs enjoys the BDDP iff for every BCQ q, for every instance D, and for every set $\Sigma \in \mathcal{C}$, if $D \cup \Sigma \models q$, then $chase^k(D,\Sigma) \models q$, where k depends only on q and Σ, i.e., k is constant w.r.t. the size of the data.*

As shown in (Calì, Gottlob, and Pieris 2010a), the class of sticky sets of TGDs enjoys the BDDP. It is well-known that the BDDP is a sufficient condition for FO-rewritability (Calì, Gottlob, and Lukasiewicz 2009). The desired result follows immediately.

Theorem 2 *BCQ answering under sticky sets of TGDs is in* AC_0 *in data complexity.*

Combined Complexity. Let us know study the combined complexity of query answering under sticky sets of TGDs. First, observe that BCQ answering under a fixed sticky set of TGDs is NP-hard. This is derived from the NP-hardness of CQ containment (which in turn is polynomially equivalent to CQ answering) without constraints (Chandra and Merlin 1977). BCQ answering under (general) sticky sets of TGDs is EXPTIME-hard. This is established by showing the EXPTIME-hardness of the fact inference problem for *lossless Datalog*. Lossless Datalog programs, which are trivially sticky sets of TGDs, are Datalog programs where each rule enjoys the following property: all variables in the body occur also in the head.

The desired upper bounds are established by exhibiting an algorithm that runs in $\text{NP}^{\mathcal{C}}$, i.e., in non-deterministic polynomial time with an oracle $\mathcal{C}$, where $\mathcal{C} = \text{ALOGSPACE}$ (resp., APSPACE), when the set of TGDs is fixed (resp., arbitrary). Note that ALOGSPACE and APSPACE denote alternating LOGSPACE and alternating PSPACE, respectively. Since ALOGSPACE = PTIME and APSPACE = EXPTIME, the following complexity characterization follows immediately.

Theorem 3 *BCQ answering under sticky sets of TGDs is* NP*-complete, if the set of TGDs is fixed, and is* EXPTIME*-complete, in general.*

Extending Stickiness

Sticky-Join Datalog$^{\pm}$. Sticky sets of TGDs are arguably a very relevant and applicable modeling tool. However, they are not expressive enough for being able to model simple cases such as the linear TGD $r(X,Y,X) \rightarrow \exists Z\, p(Y,Z)$; clearly, the variable X is marked, and thus stickiness is violated. The question that comes up is whether a FO-rewritable class can be defined that captures both sticky sets of TGDs and linear TGDs. At the first glance it may seen that it could be sufficient to allow a marked variable V to occur more than once in the body of a TGD, as long as V appears only in one body-atom. The obtained class, which we call *atom-sticky* sets of TGDs, captures linear TGDs. Nevertheless, as shown by the following example, FO-rewritability is not preserved.

Example 3 *Consider the TGD σ : $r(X,Y), r(Y,Z) \rightarrow r(X,Z)$. Observe that this rule captures the transitive*

[1]This is the complexity class of recognizing words in languages defined by constant-depth Boolean circuits with an (unlimited fan-in) AND and OR gates.

closure of the binary relation r, which in general cannot be done using a finite number of first-order queries. Now, we transform σ into the set Σ of TGDs consisting of $r(X,Y), r(Y',Z) \rightarrow s(X,Y,Y',Z)$ and $s(X,Y,Y,Z) \rightarrow r(X,Z)$. Clearly, Σ is not sticky since in the body of the second rule the marked variable Y occurs more than once. However, Y occurs in one atom only, and thus Σ is atom-sticky. Notice that, given a database D, the set of atoms with predicate r in $chase(D,\{\sigma\})$ and $chase(D,\Sigma)$, respectively, coincide. This implies that if atom-sticky sets of TGDs are FO-rewritable, then transitivity can be captured using a finite number of first-order queries which is a contradiction. ■

It turns out that the class of atom-sticky sets of TGDs is not only non-FO-rewritable, but is undecidable. This can be shown by employing the same principle as in the Example 3 to transform an arbitrary set of TGDs into an atom-sticky set. Notice that such transformation was employed in (Calì, Gottlob, and Pieris 2010b) to show that BCQ answering under *joinless* TGDs, i.e., TGDs where every body-variable occurs in at most one atom, or, equivalently, is not in a join, which is a special case of the proposed class, is undecidable.

A more restrictive condition than atom-stickiness, that gives rise to the class of *sticky-join* sets of TGDs (which forms the language *sticky-join Datalog*$^{\pm}$) was proposed in (Calì, Gottlob, and Pieris 2010b). The chase constructed under a sticky-join set of TGDs enjoys the so-called *sticky-join* property.

Definition 4 *Consider a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$. Suppose that in the construction of $chase(D,\Sigma)$ we apply $\sigma \in \Sigma$, with homomorphism h, that has a variable V in its body which is in a join, and the atoms $\underline{a}_1, \ldots, \underline{a}_k$, for $k \geqslant 1$, are generated. We say that $chase(D,\Sigma)$ has the* sticky-join property *iff, for each atom $\underline{a} \in \{\underline{a}_1, \ldots, \underline{a}_k\}$, $h(V)$ occurs in $\underline{a}$, and also in every atom $\underline{b}$ such that $\langle \underline{a}, \underline{b} \rangle$ is in the transitive closure of $\xrightarrow{D,\Sigma}$.*

Observe that the only difference between the sticky and the sticky-join property is that the latter applies stricter criteria whether a symbol (either a constant or a labeled null) must occur in every atom of a certain part of the chase. Clearly, if the chase has the sticky property, then enjoys also the sticky-join property; however, the converse is not true. The next result establishes that the chase constructed under a sticky-join set of TGDs fulfills the sticky-join property.

Theorem 4 *Consider a set Σ of TGDs over a schema $\mathcal{R}$. If Σ is sticky-join, then $chase(D,\Sigma)$ enjoys the sticky-join property, for every database D for $\mathcal{R}$.*

Similarly to sticky sets, sticky-join sets of TGDs are defined by a testable condition based on variable-marking. However, the marking procedure for this new class is more sophisticated, and for this reason the problem of identifying whether a set of TGDs is sticky-join is much harder than the one of identifying sticky sets; in particular, is PSPACE-complete. Due to lack of space, the formal definition of sticky-join sets of TGDs is omitted, and we refer the interested reader to (Calì, Gottlob, and Pieris 2010b).

Interestingly, the alternating algorithm proposed for query answering under sticky sets of TGDs, can be also used for query answering under sticky-join sets of TGDs. This allows us to show that the extension from sticky to sticky-join sets can be done without paying a price in the complexity of query answering.

Theorem 5 *BCQ answering under sticky-join sets of TGDs is in* AC_0 *in data complexity. Also, is* NP*-complete, if the set of TGDs is fixed, and is* EXPTIME*-complete, in general.*

Weakly-Sticky Datalog$^{\pm}$. A more general class, that generalizes both stickiness and weak acyclicity, called *weakly-sticky* sets of TGDs (and forms the language *weakly-sticky Datalog*$^{\pm}$), is proposed in (Calì, Gottlob, and Pieris 2010b).

Roughly, in a weakly-sticky set, the variables that occur more than once in the body of a TGD are non-marked or occur at positions where a finite number of symbols can appear during the construction of the chase. Since the chase under a weakly-acyclic set over a schema $\mathcal{R}$ always terminates, at every position of $\mathcal{R}$ only finitely many values can appear during the construction of the chase. Hence, every weakly-acyclic set of TGDs is indeed weakly-sticky.

Analogously to weakly-sticky sets, the class of *weakly-sticky-join* sets of TGDs, that generalizes both sticky-join and weakly-acyclic sets, can be defined (Calì, Gottlob, and Pieris 2010b).

The desired upper bounds of the complexity of query answering under weakly-sticky(-join) sets of TGDs are obtained by extending the alternating algorithm employed for sticky sets. The PTIME-hardness of data complexity follows immediately from the PTIME-hardness of fact inference in Datalog programs (since a Datalog program is trivially a weakly-sticky set of TGDs). Finally, the 2EXPTIME-hardness of combined complexity is established by simulating the behavior of a 2EXPTIME Turing machine using a weakly-sticky set of TGDs; in fact, this can be done using a weakly-acyclic set of TGDs.

Theorem 6 *BCQ answering under weakly-sticky(-join) sets of TGDs is* PTIME*-complete in data complexity. Also, is* NP*-complete, if the set of TGDs is fixed, and is* 2EXPTIME*-complete, in general.*

Additional Features

Negative Constraints. A *negative constraint (NC)* is a first-order formula of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\bot$ denotes the truth constant *false*. Using NCs we can state, for example, that professors and students are disjoint sets: $\forall X\ professor(X), student(X) \rightarrow \bot$. As established in (Calì, Gottlob, and Lukasiewicz 2009), checking whether a set of constraints is satisfied by a database and a set of TGDs is tantamount to query answering. Consequently, we obtain that the addition of NCs does not increase the complexity of BCQ answering under the classes of TGDs presented in the previous sections.

Functional Dependencies. While the addition of NCs can be done effortless, from a computation perspective, the addition of functional dependencies (FDs) (Abiteboul, Hull, and Vianu 1995), with which we assume the reader is familiar, is tricky. In the presence of TGDs and FDs, the chase also needs to repair according to FDs, which is done by *unifying* values. When the chase attempts at unifying (equating) two constants, there is a *hard violation*, and the chase fails as the theory is inconsistent. In certain favorable cases, a controlled interaction of TGDs and FDs is achieved, so that FDs do not increase the complexity of query answering. This is captured by the notion of *separability* (Calì, Gottlob, and Lukasiewicz 2009), which, roughly speaking, states that if there is no chase failure, then we can ignore the FDs and proceed with the TGDs only. It is possible to show that, if TGDs and FDs are separable, then the complexity of BCQ answering is the same as in the case with TGDs alone. A sufficient syntactic condition for separability of sets of TGDs and FDs is given in (Calì, Gottlob, and Pieris 2010a); sets that satisfy this condition are called *non-conflicting*.

Ontology Querying

Non-conflicting sticky sets of TGDs and FDs, with the addition of NCs, are strictly more expressive than DL-Lite$_{\mathcal{F}}$, DL-Lite$_{\mathcal{R}}$ and DL-Lite$_{\mathcal{A}}$ (Calvanese et al. 2007; Poggi et al. 2008).

Example 4 *The DL-Lite assertions*

$$\begin{array}{rcl} \mathit{Professor} & \sqsubseteq & \mathit{Member}, \\ \mathit{Phd\_student} & \sqsubseteq & \mathit{Member}, \\ \mathit{Member} & \sqsubseteq & \exists \mathit{worksIn}, \\ \mathit{Professor} & \sqsubseteq & \exists \mathit{leaderOf}, \\ \exists \mathit{worksIn}^- & \sqsubseteq & \mathit{Group}, \\ \exists \mathit{leaderOf}^- & \sqsubseteq & \mathit{Group}, \\ \mathit{Professor} & \sqsubseteq & \neg \mathit{Phd\_student}, \end{array}$$

state that professors and phd students work in research groups, professors are leaders of research groups, and no phd student is also a professor (and vice-versa). Also, the functionality assertion $(\mathsf{funct}\ \mathit{leaderOf})$ *states that every professor is the leader of at most one research group.*

The above TBox is translated into the following set of TGDs, NCs and FDs

$$\begin{array}{l} \mathit{Professor}(X) \rightarrow \mathit{Member}(X), \\ \mathit{Phd\_student}(X) \rightarrow \mathit{Member}(X), \\ \mathit{Member}(X) \rightarrow \exists Y\ \mathit{worksIn}(X,Y), \\ \mathit{Professor}(X) \rightarrow \exists Y\ \mathit{leaderOf}(X,Y), \\ \mathit{worksIn}(X,Y) \rightarrow \mathit{Group}(Y), \\ \mathit{leaderOf}(X,Y) \rightarrow \mathit{Group}(Y), \\ \mathit{Professor}(X), \mathit{Phd\_student}(X) \rightarrow \bot, \\ \mathit{leaderOf}\ :\ \{1\} \rightarrow \{2\}. \end{array}$$

Notice that the set of TGDs given above is sticky (in fact, they are inclusion dependencies). ■

By observing that concept products (Rudolph, Krötzsch, and Hitzler 2008), that is, rules of the form $p(X), q(Y) \rightarrow r(X,Y)$ which express the cartesian product of two concepts (unary relations) p and q, are very special cases of sticky sets of TGDs, it is possible to show that the above DL-Lite languages can be extended with concept product, without increasing the complexity of query answering.

Acknowledgements

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.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Beeri, C., and Vardi, M. Y. 1981. The implication problem for data dependencies. In *Proc. of ICALP*, 73–85.

Cabibbo, L. 1998. The expressive power of stratified logic programs with value invention. *Inf. Comput.* 147(1):22–56.

Calì, A.; Gottlob, G.; and Kifer, M. 2008. 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. 2010a. Advanced processing for ontological queries. *Proc. of VLDB* 3(1):554–565.

Calì, A.; Gottlob, G.; and Pieris, A. 2010b. Query answering under non-guarded rules in Datalog+/-. In *Proc. of RR*, 1–17.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2006. Data complexity of query answering in description logics. In *Proc. of KR*, 260–270.

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.

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

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

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.

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.

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

Mailharrow, D. 1998. A classification and constraint-based framework for configuration. *Artif. Intell. for Eng. Design, Anal. and Manuf.* 12(4):383–397.

Patel-Schneider, P. F., and Horrocks, I. 2007. A comparison of two modelling paradigms in the semantic web. *J. Web Semantics* 5(4):240–250.

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.

Rudolph, S.; Krötzsch, M.; and Hitzler, P. 2008. All elephants are bigger than all mice. In *Proc. of DL*.

Vardi, M. Y. 1995. On the complexity of bounded-variable queries. In *Proc. of PODS*, 266–276.