Advanced Processing for Ontological Queries

Andrea Calì[3,2]

[1]Computing Laboratory
University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3QD
United Kingdom

Georg Gottlob[1,2]

[2]Oxford-Man Institute of
Quantitative Finance
University of Oxford
Eagle House, Walton Well Road
Oxford OX2 6ED
United Kingdom

Andreas Pieris[1]

[3]Department of Information
Systems and Computing
Brunel University
St John's Building
Uxbridge UB8 3PH
United Kingdom

{andrea.cali, georg.gottlob, andreas.pieris}@comlab.ox.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. The ontological integrity constraints for generating this intensional knowledge can be specified in description logics such as DL-Lite. It was recently shown that these formalisms allow for very efficient query-answering. They are, however, too weak to express simple and useful integrity constraints that involve joins. In this paper we introduce a more expressive formalism that takes joins into account, while still enjoying the same low query-answering complexity. In our framework, ontological constraints are expressed by sets of rules that are so-called *tuple-generating dependencies (TGDs)*. We propose the language of *sticky* sets of TGDs, which are sets of TGDs with a restriction on multiple occurrences of variables (including joins) in the rule bodies. 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 how sticky sets of TGDs can be combined with functional dependencies. In summary, we obtain a highly expressive and effective ontological modeling language that unifies and generalizes both classical database constraints and important features of the most widespread tractable description logics.

1. 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 processing mechanisms. The necessity of combining ontological reasoning and description logics (DLs) with database techniques has emerged in both the DL community and in the database world. The marriage of techniques arising from both contexts is indeed seen as a great commercial opportunity. This is not surprising, given that the raw data on top of which ontological reasoning tasks are executed reside, in the business world, in enterprise databases. Making such enterprise data available to ontological reasoning is thus a key step towards the commercial breakthrough of DL and Semantic Web technology. Both small and large database technology providers have recognized this need and have recently started to build ontological reasoning modules on top of their existing software.

Oracle Inc. offers a DBMS enhanced by modules performing ontological reasoning tasks in order to provide semantic technologies for enterprises. Their product *Oracle Spatial 11g Semantic Technologies* [2] has applications in several areas of ontological data processing, among which we list: enterprise information integration, intelligence, law enforcement, knowledge mining, threat analysis, finance, web and social network solutions, life science research, in particular, bio-pathway analysis and protein interaction. A recent example of a new company dealing with ontological data access is Data-Grid Inc. [1], which develops *OWL-DBMS*, a DBMS offering semantic web technology and ontological querying. Integrating DLs and databases is also at the heart of several *research-based systems*, among which QuOnto [4] and FaCT [23]. Quonto is based on the DL-lite description logic family [15], and has interfaces to Oracle, DB2, and SQL Server; it takes advantage of the optimization capabilities of the underlying DBMS.

In ontology-enhanced database systems, an extensional relational database D (also referred-to as ABox in the description logic community) is combined with an *ontological theory* Σ (also called TBox) describing rules and constraints which derive new intensional data from the extensional data. A query is not just answered against the database D, but against the logical theory $D \cup \Sigma$. Thus, for a Boolean conjunctive query (CQ) q, one checks whether $D \cup \Sigma \models q$ rather than just checking whether $D \models q$. Similarly, if $q(\mathbf{X}) = \exists \mathbf{Y}\, body(\mathbf{X}, \mathbf{Y})$ is a conjunctive query with output variables $\mathbf{X}$, then its answer in the ontological database consists of all tuples $\mathbf{t}$ of constants such that $D \cup \Sigma \models \exists \mathbf{u}\, body(\mathbf{t}, \mathbf{u})$. Moreover, answering a conjunctive query q against $D \cup \Sigma$ has been shown to be equivalent to answer the same query q against the chase-expansion of D

Proceedings of the VLDB Endowment, Vol. 3, No. 1

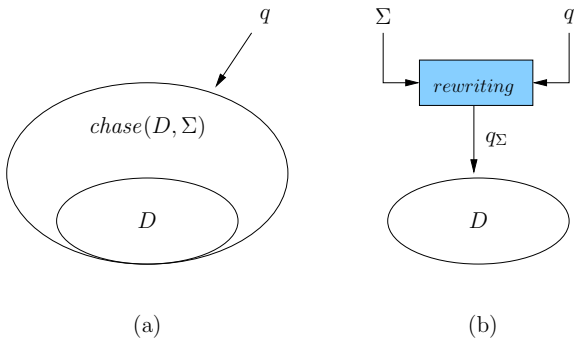

Figure 1: Query answering: chasing vs. rewriting.

according to Σ, which we denote by $chase(D, \Sigma)$ [24, 14, 22, 20]. This expansion can be obtained via the well-known *chase* algorithm [25, 24, 20], which we will review in Section 2. Informally, the chase adds new tuples to D (possibly involving null values) until the final result $chase(D, \Sigma)$ satisfies all constraints in Σ. A picture of ontological querying via the chase is given in Figure 1 (a).

Research Challenges. A particularly critical issue is that the chase expansion $chase(D, \Sigma)$ may be infinite, and thus not explicitly computable. This may be the case even for extremely small databases D and very simple ontological theories Σ. The following simple example illustrates this and also exhibits a Boolean query q that evaluates to true over $chase(D, \Sigma)$, while it is false over D alone.

Example 1. Consider a database $D = \{person(john)\}$ that contains a single fact stating that john is a person, and the ontological theory Σ stating that every person has a father, who is himself a person:

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

$chase(D, \Sigma)$ is then the infinite atom set $\{person(john)$, $father(z_1, john)$, $person(z_1)$, $father(z_2, z_1)$, $person(z_2)$, $father(z_3, z_2)$, $person(z_3), \ldots\}$, where all the z_i are (labeled) null values. Clearly, the Boolean conjunctive query

$$q \;=\; \exists X \exists Y \exists Z\, father(X, Y), father(Y, Z)$$

is positively answered on this infinite expansion, and indeed, $D \cup \Sigma \models q$; however, $D \not\models q$. Note that the two constraints in the ontological theory Σ are actually the inclusion dependencies $person[1] \subseteq father[2]$ and $father[1] \subseteq person[1]$. ■

Procedures for effectively answering queries even when the chase does not terminate were first developed in the database context by Johnson and Klug [24] for the special case where Σ contains inclusion dependencies only. In short, they showed that for each Boolean conjunctive query q, a finite initial part C_0 of $chase(D, \Sigma)$ can be computed, whose size depends on the size of q and Σ only, such that for each database D, $C_0 \models q$ iff $chase(D, \Sigma) \models q$ iff $D \cup \Sigma \models q$.

Inclusion dependencies alone, however, are not powerful enough to capture some popular ontological constraints. For this reason, a recent, important research direction is to find more expressive formalisms, for which query answering under constraints is still decidable. Moreover, to be able to work with very large databases, it is desirable not only that query answering is decidable, but also tractable in data complexity (i.e., when both Σ and q are fixed), and possibly feasible by use of relational query processors.

A significant step forward in this direction was the introduction of the *DL-lite* description logic family by Calvanese et al. in [15, 27]. Conjunctive query answering in DL-lite has the advantage of being *first-order (FO) rewritable*. In other terms, a pair $\langle \Sigma, q \rangle$, where Σ is a DL-lite ontology and q a conjunctive query defined as $q(\mathbf{X}) = \exists \mathbf{Y}\, body(\mathbf{X}, \mathbf{Y})$, can be rewritten as a first order query q_Σ, defined as $q_\Sigma(\mathbf{X}) = \exists \mathbf{Y}\, body_\Sigma(\mathbf{X}, \mathbf{Y})$, such that for every possible answer tuple $\mathbf{t}$ (of constants) it holds $D \cup \Sigma \models \exists \mathbf{u}\, body(\mathbf{t}, \mathbf{u})$ iff $D \models \exists \mathbf{u}\, body_\Sigma(\mathbf{t}, \mathbf{u})$. Given that each first order query can be equivalently written in SQL, in practical terms this means that a conjunctive query q based on an ontology Σ can be rewritten as an SQL query over the original database D. Figure 1 (b) depicts the notion of rewriting.

DL-lite was more recently embedded into a slightly more expressive language called *linear Datalog*$^\pm$ [10], which is in turn a sub-formalism of a noticeably more expressive formalism called *guarded Datalog*$^\pm$. Such languages are part of the Datalog$^\pm$ family, whose languages extend the well known Datalog language [3] by allowing existential quantifiers in rule heads, thus using *tuple generating dependencies* (TGDs) instead of classical Datalog rules. This feature is also known as *value invention* (see, e.g., [26, 8]). Guarded Datalog$^\pm$ restricts rule bodies to be *guarded*, which means that each rule body has a *guard* atom, which has among its arguments all the body variables. Linear Datalog$^\pm$ further restricts rule bodies to contain a single atom only (which is then automatically a guard). Linear Datalog$^\pm$ is FO-rewritable, while guarded Datalog$^\pm$ is not. However, query answering under guarded Datalog$^\pm$ was shown to be tractable in case the ontology Σ and the query q are assumed to be fixed (data complexity). For more details see [10].

Unfortunately, none of the above formalisms is expressive enough to be able to model real-life cases such as the one in the example below.

Example 2. Consider the following relational schema, which shall be used as our running example.

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

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

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

The following TGD expresses the fact that each employee works on some project that falls into his/her area of specialization ran by his/her department.

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

The fact that for each project run by some department there exists an external controller, specialized on the area of the project, that works on it can be expressed by the TGD

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

Notice first that the above TGDs do not guarantee the termination of the chase for all initial databases. Moreover,

the third TGD of Example 2 contains a *join* over variable X in its rule body; but precisely such joins cannot be expressed by guarded TGDs, let alone by DL-lite. Query answering with TGDs involving joins is generally undecidable (see, e.g., [7], and [9] for examples of very tight undecidable classes). We observed, however, that this undecidability result is due to rather "contorted" cases that are unlikely to occur in practice. The main research challenge underlying our work is thus to find expressive classes of constraints that: *(1)* generalize DL-lite and other prominent ontology formalisms; *(2)* allow for joins in rule bodies (with some realistic restrictions to ensure decidability), and *(3)* still keep the most beneficial property of FO-rewritability, or at least of tractability of query answering when data complexity is considered. Clearly, such new classes must be based on some new decidability paradigm, that significantly differs from the paradigm of rule guardedness.

Results. As a novel Datalog$^{\pm}$ language, we introduce the class of *sticky* sets of TGDs. Stickiness is formally defined in Section 3.1 by an efficiently testable condition involving variable marking. In Section 3.2 we give a number of compelling reasons for the adoption of sticky sets of TGDs in data modeling and ontology querying.

In Section 3.3 we first tackle the data complexity of the CQ answering problem under sticky sets of TGDs. An AC_0 upper bound is proved by showing that sticky sets of TGDs enjoy the *bounded derivation-depth property (BDDP)* [10], which ensures that a number δ of chase steps, where δ depends on the query and the ontolology but not on the size of the initial instance, are sufficient for query answering. The BDDP ensures FO-rewritability, which in turns implies membership in AC_0. In Section 3.3 we also tackle the *combined complexity* of the CQ answering problem under sticky sets of TGDs, i.e., the complexity in the case where the input is constituted by the query, the constraints, and the data. We show that the problem in this case is EXPTIME-complete. The hardness is proved by showing the EXPTIME-hardness of the fact inference problem for *lossless Datalog*, whose rules are a special case of sticky sets of TGDs. Membership in EXPTIME is proved by exhibiting an alternating PSPACE algorithm which guesses a proof of the query.

In Section 4 we address the problem of the interaction between TGDs and *functional dependencies (FDs)*, a special case of equality-generating dependencies (EGDs). We provide a sufficient, syntactic condition (the *non-conflict* condition) that ensures the *separability* between TGDs and FDs [14, 10]. Separability guarantees that queries can be answered by considering only the TGDs, apart from an initial check. Therefore, all our results on TGDs extend to the case where FDs are present, that do not conflict with the TGDs.

Finally, in Section 5, we show that we can incorporate negative constraints in a knowledge base constituted by sticky sets of TGDs without increasing the complexity of query answering. This allows us to show that our work properly generalizes several prominent and tractable formalisms for ontology reasoning, in particular the DL-lite family [15, 27].

Notice that all our complexity results, derived for Boolean CQs, carry over, as usual, to the (decision) query answering problem for general (non-boolean) CQs (see, e.g., [9]), as well as to the CQ containment problem.

Our study aims at investigating the fundamental properties of our newly-introduced language, bringing to light the primary sources of the decidability and complexity issues underlying them. Given that sticky sets of TGDs, in general, neither enjoy the chase termination property, nor the bounded treewidth property, let alone guardedness, we had to look for new decision procedures way beyond the state of the art. We do believe that our work paves the way to the development of efficient query answering techniques applicable in real-world cases, and scalable to large data sets.

For most proofs in this paper, and for more details, we refer the interested reader to a report available online [11]. Additional results extending sticky sets of TGDs are published in [13].

2. DEFINITIONS AND BACKGROUND

2.1 Technical definitions

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

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

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

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

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

A *conjunctive query (CQ)* q of arity n over a schema $\mathcal{R}$, written as q/n, has the form $q(\mathbf{X}) = \exists \mathbf{Y} \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables or constants in Γ, and the length of $\mathbf{X}$ is n. $\varphi(\mathbf{X}, \mathbf{Y})$ is called the *body* of q, denoted as $body(q)$.

A *Boolean CQ (BCQ)* is a CQ of zero arity. The *answer* to a CQ q/n over a database D, denoted as $q(D)$, is the set of all n-tuples $\mathbf{t} \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \rightarrow \Gamma \cup \Gamma_f$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $h(\mathbf{X}) = \mathbf{t}$. A BCQ has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that has positive answer. Formally, a BCQ has *positive* answer over D, denoted as $D \models q$, iff $\langle\rangle \in q(D)$, or, equivalently, $q(D) \neq \varnothing$.

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

We now define the notion of *query answering* under TGDs. Given a database D for $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all databases B such that $B \models D \cup \Sigma$, which means that $B \supseteq D$ and B satisfies Σ. The *answer* to a CQ q w.r.t. D and Σ, denoted as $ans(q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in q(B)$ for each $B \in mods(D, \Sigma)\}$. The *answer* to a BCQ q w.r.t. D and Σ is positive, denoted as $D \cup \Sigma \models q$, iff $ans(q, D, \Sigma) \neq \varnothing$. Recall that query answering under general TGDs is undecidable [7], even when the schema and the set of TGDs are fixed [9].

We recall that the two problems of CQ and BCQ evaluation under TGDs are LOGSPACE-equivalent. Moreover, it is easy to see that the query output tuple problem (as a decision version of CQ evaluation) and BCQ evaluation are AC_0-reducible to each other. Henceforth, we thus focus only on the BCQ evaluation problem. All complexity results carry over to the other problems.

The TGD Chase. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [25], and later for checking query containment [24]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. We shall use the term chase interchangeably for both the procedure and its result. The chase works on an instance through the so-called TGD *chase rule*. The TGD chase rule comes in two different equivalent fashions: *oblivious* and *restricted* [9], where the restricted one repairs TGDs only when they are not satisfied. In the sequel, we focus on the oblivious one for better technical clarity (unless explicitly stated otherwise). The TGD chase rule defined below is the building block of the chase.

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

Given a database D and set of TGDs Σ, the chase algorithm for D and Σ consists of an exhaustive application of the TGD chase rule in a breadth-first fashion, which leads as result to a (possibly infinite) chase for D and Σ, denoted as $chase(D, \Sigma)$. A formal definition is given in Appendix A.

The (possibly infinite) chase for D and Σ is a *universal model* of D w.r.t. Σ, i.e., for each database $B \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to B [22, 20]. Using this fact it can be shown that for a BCQ q, $D \cup \Sigma \models q$ iff $chase(D, \Sigma) \models q$.

2.2 Ontology Languages and Rewriting

DL-lite is a family of languages based on the notions of *concepts* (sets of objects) and *roles* (binary relations between concepts). DL-lite languages include several constructs, including: functional and mandatory participation constraints (where the participation is that of a concept to a role), role and concept inclusion, and others. We do not give the syntax and the semantics of such constructs here, and we refer the reader, for instance, to [15, 27]. Interestingly, DL-lite languages can be translated into inclusion dependencies, with the addition of non-conflicting key constraints as in [14] and of negative constraints as in [12]. As an example, let us retake Example 1. First, notice that the two inclusion dependencies can be expressed in DL-lite with the assertions: $person \sqsubseteq \exists father^-$ and $\exists father \sqsubseteq person$, where $\exists father$ (resp. $\exists father^-$) denotes the set of objects that participate at least once to the relation *father* as first argument (resp. as second argument), i.e., fathers of some object (resp. sons of some object). Consider now the Boolean query $q = \exists X \exists Y \exists Z\, father(X, Y), father(Y, Z)$ asking whether there is any grandfather. The query evaluates to true iff the initial instance is nonempty, and a possible rewriting of q is $q_\Sigma = \exists X \exists Y\, father(X, Y) \vee \exists Z\, person(Z)$, which is expressible in SQL, being a first-order query.

DL-lite, in spite of its good capability of modeling ontologies, cannot capture any of the TGDs in Example 2 – its extension DLR-lite [16], which admits roles with arbitrary arity, can express the first two. One fundamental limitation of DL-lite languages (as well as of DLR-lite) is that there is no way they can express joins, as in the TGD

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

of Example 2. Joins are beyond the expression capabilities of DL-lite. Nevertheless, sticky sets of TGDs, though more expressive, are FO-rewritable, like DL-languages.

As a simple example of rewriting, let us consider the CQ q defined as $q(Y) = \exists X\, external(X, toys, Y)$, asking for projects in the area of toys for which there are external controllers. Intuitively, due to the above TGD, not only we have to query *external*, but we also need to look for projects that are run by a department, as such projects will have an external controller. The rewriting q_Σ will thus be the logical *union* of q and of the query $q_1(Y) = \exists W\, runs(W, Y) \wedge in\_area(Y, toys)$; q_Σ is a first-order query, and thus expressible in SQL:

```
SELECT E.project_id FROM external E
WHERE E.area='toys'
UNION
SELECT R.project_id FROM runs R, in_area I
WHERE I.area='toys' AND R.project_id=I.project_id
```

In the following section we introduce a novel class of FO-rewritable ontologies (sets of TGDs in this case).

3. STICKY SETS OF TGDS

In this section we introduce *sticky* sets of TGDs. We show that query answering under sticky sets of TGDs is highly

tractable in data complexity, specifically in AC_0, and it is EXPTIME-complete in combined complexity.

3.1 Definition of Sticky Sets of TGDs

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

Example 3. Consider the relational schema $\mathcal{R}$ given in Example 2, and let Σ be the set of TGDs over $\mathcal{R}$ in Example 2. According to the marking procedure in Definition 1, we mark the variables as follows (we mark variables with a cap, e.g., $\hat{X}$):

$$dept(\hat{V}, \hat{W}) \rightarrow \exists X \exists Y \exists Z\, emp(W, X, Y, Z),$$
$$emp(\hat{V}, \hat{W}, \hat{X}, \hat{Y}) \rightarrow \exists Z\, dept(W, Z), runs(W, Y), in\_area(Y, X),$$
$$runs(\hat{W}, X), in\_area(X, Y) \rightarrow \exists Z\, external(Z, Y, X).$$

Clearly, for each TGD $\sigma \in \Sigma$, there is no marked variable that occurs in $body(\sigma)$ more than once. Therefore, Σ is a sticky set of TGDs. It is important to observe that the set Σ is neither weakly-acyclic [22], nor guarded [9], nor weakly-guarded [9]. In fact, the class of sticky sets of TGDs is incomparable to the above three known classes. ■

It is interesting to see that an equivalent criterion for stickiness, based on the chase, can be defined. Given a database D for a schema $\mathcal{R}$, and a set Σ of TGDs over $\mathcal{R}$, we define the (binary) relation $\stackrel{D,\Sigma}{\longrightarrow}$ 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 atom $\underline{b} \in body(\sigma)$, we have that $h(\underline{b}) \stackrel{D,\Sigma}{\longrightarrow} \underline{a}$. We now define when the chase enjoys the so-called sticky property.

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 a TGD $\sigma \in \Sigma$, with homomorphism h, that has a variable V appearing more than once in its body, and the atom $\underline{a}$ is generated. We say that $chase(D, \Sigma)$ has the *sticky property* iff $h(V)$ occurs in $\underline{a}$, and in every atom $\underline{b}$ such that $\langle \underline{a}, \underline{b} \rangle$ is in the transitive closure of $\stackrel{D,\Sigma}{\longrightarrow}$.

The following result implies that the sticky property characterizes the class of sticky sets of TGDs; the proof is found in Appendix B (Theorem B.3)

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

3.2 Usefulness of Sticky Sets of TGDs

Sticky sets of TGDs are arguably a very relevant and applicable modeling tool:

- Together with standard constructs such as keys and negative constraints (see Sections 4 and 5), 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 (Theorem 10). 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, 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 with 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 (MVD) is sticky, as are many realistic sets of MVDs.
- Sticky sets of TGDs very significantly generalize some other constructs that were introduced to enhance description logics with joins. Noticeably, Rudolph and Krötsch [29] 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 [29] to support the introduction of the concept product in DLs. Note that concept products are very special cases of sticky sets of TGDs; moreover, the addition of a concept product to a sticky set of TGDs can never make the resulting set non-sticky.
- *Weakly-sticky* sets of TGDs, a natural extension of sticky sets of TGDs, are able to capture both Datalog and weakly-acyclic sets of TGDs, by paying of course a price in complexity. Roughly, in a weakly-sticky set of TGDs, 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 chase. Weakly-sticky sets of TGDs are formally defined in Appendix D, and we refer the reader to [11] for the complexity results.

3.3 Algorithms and Complexity

In this subsection we study the data and combined complexity of query answering under sticky sets of TGDs. To this aim we first exhibit an alternating algorithm, which guesses a proof of the given query, and establish its soundness and completeness. We recall that query answering under (general) TGDs is equivalent to query answering under

TGDs with just one atom in their heads [9]. This is established by providing a LOGSPACE transformation from (general) TGDs to TGDs with singleton atoms in their heads. Since this transformation preserves the syntactic condition of sticky sets of TGDs, henceforth we assume w.l.o.g. that every TGD has just one atom in its head. In addition, for technical reasons, we do not allow repetition of variables in the head-atom. Note that most realistic examples do not have repeated head-variables.

Algorithm QAns. QAns has as input a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a sticky set Σ of TGDs over $\mathcal{R}$, and outputs "Accept" iff $D \cup \Sigma \models q$, or, equivalently, $chase(D, \Sigma) \models q$. The algorithm works as follows:

1. Guess which variables in $var(q)$, during the evaluation of q over the $chase(D, \Sigma)$, are mapped onto a constant of $dom(D)$ (and which constant), and also which are mapped onto the same null. Let S be the set of variables that are mapped onto a null.
2. Guess, for each variable $V \in S$, an atom $\underline{a}_V$ (possibly containing constant values) that represents the atom in which z_V is invented during the chase, where z_V is the null onto which V is mapped.
3. Halt and "Accept" iff for each variable $V \in S$, each occurrence of V is mapped onto the same null. This is done by applying an alternating procedure which uses the guessed atoms for the variables in S.

For more details on the algorithm QAns we refer the reader to Appendix B. The formal definition of QAns is found in [11]. The following result implies that the algorithm QAns is sound and complete for query answering under sticky sets of TGDs.

THEOREM 2. *Let $\mathcal{R}$ be a relational schema. Consider a sticky set Σ of TGDs over $\mathcal{R}$, a database D for $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. It holds that $D \cup \Sigma \models q$ iff QAns(q, D, Σ) outputs "Accept".*

Data Complexity. A class $\mathcal{C}$ of TGDs is *first-order rewritable*, henceforth abbreviated as *FO-rewritable*, iff for every set Σ of TGDs in $\mathcal{C}$, and for every BCQ q, there exists a first-order query q_Σ such that, for every database D, $D \cup \Sigma \models q$ iff $D \models q_\Sigma$ [27, 10]. Since answering first-order queries is in the class AC_0[1] in data complexity [30], 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)* [10]. Roughly speaking, $chase(D, \Sigma)$ can be decomposed in levels, where D has level 0, and an atom has level $\gamma + 1$ if it is obtained, during the chase, due to atoms with maximum level γ. We refer to the part of the chase up to level γ as $chase^\gamma(D, \Sigma)$. For the formal definitions see Appendix A.

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 $chase^\gamma(D, \Sigma) \models q$, where γ depends only on q and Σ, i.e., γ is constant w.r.t. the size of the data. To establish this result we exploit the alternating algorithm QAns described above.

[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.

THEOREM 3. *The class of sticky sets of TGDs enjoys the BDDP.*

PROOF IDEA. Roughly, the alternating algorithm QAns described above, given a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a sticky set Σ of TGDs over $\mathcal{R}$, constructs a finite portion of the chase $P \subseteq chase^\gamma(D, \Sigma)$ such that $P \models q$ iff $D \cup \Sigma \models q$. The claim follows by observing that γ depends solely on $|var(q)|$ and Σ, but does not depend on D. For more details see [11]. □

It is known that if a class of TGDs enjoys the BDDP, then it is FO-rewritable [10]. Thus, we get the desired result.

COROLLARY 4. *BCQ answering under sticky sets of TGDs is in AC_0.*

Combined Complexity. First, observe that query answering under a fixed sticky set of TGDs is NP-hard. This is derived from NP-hardness of query containment (which in turn is polynomially equivalent to query answering) without constraints [17]. We continue to show that query answering under (general) sticky sets of TGDs is EXPTIME-hard. Before we proceed further we give some preliminary definitions.

A *lossless Datalog rule* is a Datalog rule such that the variables that appear in the body occur also in the head. A *lossless Datalog program* $\mathcal{P}$ is a set of lossless Datalog rules. The *Datalog fact inference problem* is the following: given a database D for a schema $\mathcal{R}$, a Datalog program $\mathcal{P}$ over $\mathcal{R}$, and a ground atom $\underline{a}$, decide whether $D \cup \mathcal{P} \models \underline{a}$. The same problem, if we consider lossless Datalog programs, is EXPTIME-hard; the proof is found in Appendix B (Theorem B.4).

THEOREM 5. *The lossless Datalog fact inference problem is EXPTIME-hard.*

Since each lossless Datalog program is a sticky set of TGDs, from Theorem 5 we immediately get the following result.

COROLLARY 6. *BCQ answering under sticky sets of TGDs is NP-hard, if the set of TGDs is fixed, and it is EXPTIME-hard in general.*

We now establish that query answering under sticky sets of TGDs is in NP in case that the set of constraints is fixed, and in EXPTIME in general. We prove this result by using the alternating algorithm QAns presented above.

THEOREM 7. *BCQ answering under sticky sets of TGDs is in NP, if the set of TGDs is fixed, and it is in EXPTIME in general.*

PROOF IDEA. The alternating algorithm QAns runs in $NP^{ALOGSPACE} = NP$ (resp., $NP^{APSPACE} = EXPTIME$) in case the set of TGDs is fixed (resp., is part of the input), where ALOGSPACE and APSPACE denote alternating LOGSPACE and alternating PSPACE, respectively. We refer the interested reader to [11] for a full proof. □

The following complexity characterization follows immediately from Corollary 6 and Theorem 7.

COROLLARY 8. *BCQ answering under sticky sets of TGDs is NP-complete, if the set of TGDs is fixed, and it is EXPTIME-complete in general.*

4. ADDING EGDS

So far, we have dealt only with TGDs. In this section we also consider equality-generating dependencies (EGDs). The interaction of general TGDs and EGDs has been proved to lead to undecidability of query answering. In fact, this is true even in simple cases such that of functional and inclusion dependencies [18]. Thus, we cannot hope to extend the results established in the previous section to cover also EGDs. We are looking for suitable syntactic restrictions which would guarantee decidability of query answering.

An *equality-generating dependency* over a schema $\mathcal{R}$ is a first-order formula $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow X_i = X_j$, where $\varphi(\mathbf{X})$ is a conjunction of atoms over $\mathcal{R}$, and $X_i, X_j \in \mathbf{X}$. Such an EGD is satisfied by a database D iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq D$, then $h(X_i) = h(X_j)$. In the context of description logics, general EGDs cannot be formulated, but only functional dependencies (FDs) [3]; thus, we restrict our attention on FDs.

A *functional dependency* over a schema $\mathcal{R}$ is an assertion of the form $r : \mathbf{A} \rightarrow \mathbf{B}$, where $r \in \mathcal{R}$ and $\mathbf{A}, \mathbf{B}$ are sets of attributes of r. Such a FD is satisfied by a database D iff, whenever there exist two (distinct) tuples $\mathbf{t_1}, \mathbf{t_2} \in r(D)$ such that $\mathbf{t}_1[\mathbf{A}] = \mathbf{t}_2[\mathbf{A}]$, where $\mathbf{t}[\mathbf{A}]$ is the projection of tuple $\mathbf{t}$ over $\mathbf{A}$, then $\mathbf{t}_1[\mathbf{B}] = \mathbf{t}_2[\mathbf{B}]$. Observe that FDs can be identified with sets of EGDs. For example, consider the FD $\phi = r : \{1\} \rightarrow \{2, 4\}$, defined on a predicate $r/4$. Clearly, ϕ can be identified by the set of EGDs $\Sigma_E = \{r(X, Y_2, Y_3, Y_4), r(X, Z_2, Z_3, Z_4) \rightarrow Y_i = Z_i\}_{i \in \{2,4\}}$. It is easy to verify that for every database D it holds that D satisfies ϕ iff D satisfies Σ_E.

As for TGDs, a FD chase rule can be defined. Roughly, during the application of the FD chase rule, we unify symbols in order to satisfy the violated FD. If two constants are unified, the we have a so-called *hard violation* and the chase *fails*; for the formal definition see Appendix C.

Given a database D and a set $\Sigma = \Sigma_T \cup \Sigma_F$, where Σ_T and Σ_F are sets of TGDs and FDs, respectively, the chase of D relative to Σ is computed by iteratively applying: *(i)* a single TGD once, and *(ii)* the FDs, as long as they are applicable (i.e., until a fixpoint is reached); for an example see Appendix C (Example C.1).

We now recall the notion of separability which formulates a controlled interaction of TGDs and FDs, so that FDs do not increase the complexity of query answering [14, 10].

Definition 3. Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_F$ over $\mathcal{R}$, where Σ_T and Σ_F are sets of TGDs and FDs, respectively. Σ is *separable* iff for every database D for $\mathcal{R}$ the following conditions are satisfied: *(i)* if $chase(D, \Sigma)$ fails, then D violates Σ_F, and *(ii)* if there is no chase failure then, for every BCQ q over $\mathcal{R}$ it holds that $chase(D, \Sigma) \models q$ iff $chase(D, \Sigma_T) \models q$.

Non–Conflicting Sets of TGDs and FDs. In what follows we provide a sufficient syntactic condition for separability. For a TGD $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$, we define the set $\mathbf{U}_\sigma$ as the set of attributes of r at which we have a universally quantified variable. The following definition generalizes the notion of *non-key-conflicting TGDs* [10], which in turn generalizes the notion of *non-key-conflicting inclusion dependencies* [14].

Definition 4. Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_F$ over $\mathcal{R}$, where Σ_T and Σ_F are sets of TGDs and FDs, respectively. Σ is *non-conflicting* if for each pair $\langle \sigma, \phi \rangle \in \Sigma_T \times \Sigma_F$, where $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$ and $\phi = r : \mathbf{A} \rightarrow \mathbf{B}$, $\mathbf{U}_\sigma$ is not a strict superset of $\mathbf{A}$.

The next result shows that non-conflicting sets of TGDs and FDs are separable. The proof is found in Appendix C (Theorem C.1).

THEOREM 9. *Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_F$ over $\mathcal{R}$, where Σ_T and Σ_F are sets of TGDs and FDs, respectively. If Σ is non-conflicting, then it is separable.*

The above result implies that, in the non-conflicting case, FDs do not increase the data and combined complexity of query answering.

5. ADDING NEGATIVE CONSTRAINTS

We now come to negative constraints. A *negative constraint* is a first-order sentence of the form $\forall \mathbf{X}\, \phi(\mathbf{X}) \rightarrow \bot$, where $\bot$ denotes the truth constant *false* (the universal quantification is usually omitted for brevity).

Example 4. Let $\mathcal{R}$ be the relational schema given in Example 2. The fact that departmental managers cannot be project managers can be represented by the negative constraint: $dept(U, V), proj\_mgr(V, W) \rightarrow \bot$. Moreover, the fact that projects which fall in the area of security cannot be assigned to external controller can be represented by the negative constraint: $dept(U, security, V) \rightarrow \bot$. Observe that in the body of a negative constraint we can have both variables and constants. ■

In [10], it is shown that checking negative constraints is tantamount to query answering. In particular, given an instance D, a set $\Sigma_\bot$ of negative constraints, and a sticky set Σ of TGDs, for each negative constraint ν of the form $\phi(\mathbf{X}) \rightarrow \bot$, we answer the BCQ $q_\nu = \exists \mathbf{X}\, \phi(\mathbf{X})$. If at least one of such queries answers positively, then $D \cup \Sigma \cup \Sigma_\bot \models \bot$ (i.e., the theory is inconsistent), and therefore for every BCQ q it holds $D \cup \Sigma \cup \Sigma_\bot \models q$; otherwise, given a BCQ q, we have $D \cup \Sigma \cup \Sigma_\bot \models q$ iff $D \cup \Sigma \models q$, i.e., we can answer q by ignoring the negative constraints.

The following result can be proved exactly as the analogous result in [10] for linear TGDs by observing that a set of inclusion dependencies is trivially sticky.

THEOREM 10. *Non-conflicting sticky sets of TGDs and FDs, with the addition of negative constraints, are strictly more expressive than DL-lite$_\mathcal{A}$, DL-lite$_\mathcal{R}$ and DL-lite$_\mathcal{F}$.*

6. CONCLUSIONS

In this paper, we have presented a complete picture, with data and combined complexity bounds, of the computational complexity of conjunctive query answering under sticky sets of TGDs. The new ontology language we have introduced, which is a member of the Datalog$^\pm$ family, is based on a completely novel paradigm called stickiness. We have also shown that all results extend when we combine TGDs and FDs, as long as the FDs do not interact with TGDs. This is particularly important since, in ontological reasoning, FDs are essential to represent, for instance, functional participation constraints of a class (concept) to a relationship (role). The addition of negative constraints also does not increase the complexity of query answering.

A more general class, which we call *weakly-sticky* sets of TGDs, and which constitute weakly-sticky Datalog$^{\pm}$, is discussed in Appendix D.

Interestingly, stickiness is a sufficient syntactic property that ensures that the TGDs are a so-called FINITE UNIFICATION SET (FUS). A FUS is semantically characterized as a set of TGDs that enjoy the following property: for every conjunctive query q, the rewriting q_Σ of q obtained by backward-chaining through unification, according to the rules in Σ, terminates. We refer the reader to [5] for a formal definition. Notice that under certain conditions, as specified in [5], a FUS can be combined with a BOUNDED TREEWIDTH SET (BTS), i.e., a set of TGDs such that the chase under such TGDs has bounded treewidth, while retaining decidability of query answering. Therefore, query answering under sticky sets of TGDs and *weakly-guarded* sets of TGDs (a generalization of guarded TGDs [9]) is decidable, providing that the conditions specified in [5] are fulfilled.

In the present paper we have considered entailment under arbitrary (finite or infinite) models; when this coincides with entailment under finite models only, it is said that *finite controllability* [24, 28, 6] holds. Whether sticky sets of TGDs are finitely controllable is an open problem, which we plan to solve. Another direction we intend to pursue is finding a decidable class of TGDs that generalizes both guarded TGDs and sticky sets of TGDs.

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

7. REFERENCES

[1] Data grid inc. http://dt123.com/DataGrid/DataGridWebsiteV1a/.

[2] Oracle semantic technology center. http://www.oracle.com/technology/tech/semantic_technologies/.

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

[4] A. Acciarri, D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, M. Palmieri, and R. Rosati. QuOnto: Querying ontologies. In *Proc. of AAAI*, 2005.

[5] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. Extending decidable cases for rules with existential variables. In *Proc. of IJCAI*, 2009.

[6] V. Barany, G. Gottlob, and M. Otto. Querying the guarded fragment. In *Proc. of LICS*, 2010.

[7] C. Beeri and M. Y. Vardi. The implication problem for data dependencies. In *Proc. of ICALP*, 1981.

[8] L. Cabibbo. The expressive power of stratified logic programs with value invention. *Inf. Comput.*, 147(1), 1998.

[9] A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proc. of KR*, 2008.

[10] A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS*, 2009.

[11] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. Unpublished manuscript. Available at: http://benner.dbai.tuwien.ac.at/staff/gottlob/CGP.pdf.

[12] A. Calì, G. Gottlob, and A. Pieris. Tractable query answering over conceptual schemata. In *Proc. of ER*, 2009.

[13] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+/-. In *Proc. of RR*, 2010.

[14] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of PODS*, 2003.

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

[16] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Data complexity of query answering in description logics. In *Proc. of KR*, 2006.

[17] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of STOCS*, 1977.

[18] A. K. Chandra and M. Y. Vardi. The implication problem for functional and inclusion dependencies. *SIAM Journal of Computing*, 14, 1985.

[19] E. Dantsin, T. Eiter, G. Georg, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3), 2001.

[20] A. Deutsch, A. Nash, and J. B. Remmel. The chase revisisted. In *Proc. of PODS*, 2008.

[21] A. Deutsch and V. Tannen. Reformulation of XML queries and constraints. In *Proc. of ICDT*, 2003.

[22] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theor. Comput. Sci.*, 336(1), 2005.

[23] I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of KR*, 1998.

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

[25] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4), 1979.

[26] D. Mailharrow. A classification and constraint-based framework for configuration. *Artif. Intell. for Engineering Design, Analysis and Manufacturing*, 12(4), 1998.

[27] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10, 2008.

[28] R. Rosati. On the decidability and finite controllability of query processing in databases with incomplete information. In *Proc. of PODS*, 2006.

[29] S. Rudolph, M. Krötzsch, and P. Hitzler. All elephants are bigger than all mice. In *Description Logics*, 2008.

[30] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS*, 1995.

APPENDIX

A. THE TGD CHASE

The notion of the *(derivation) level* of an atom in a TGD chase is defined as follows. Let D be the *initial* database from which the chase is constructed. The atoms in D have zero level. Now, let a TGD $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X},\mathbf{Z})$ be applied at some point in the construction of the chase, and let h and h' be as in the TGD chase rule. If the atom with highest level among those in $h(\varphi(\mathbf{X},\mathbf{Y}))$ has level k, then every atom in $h'(\psi(\mathbf{X},\mathbf{Z}))$ has level $k+1$.

Let D be a database for a schema $\mathcal{R}$, and let Σ be a set of TGDs over $\mathcal{R}$. The *chase* of D relative to Σ, denoted as $chase(D,\Sigma)$, is the database built by an iterative application of the TGD chase rule as follows. Let $I_1,\ldots,I_k$ be all possible images of bodies of TGDs in Σ relative to some homomorphism, and $\underline{a}_i$ be the atom with highest level in I_i; let M be such that $level(\underline{a}_M) = \min_{1\leqslant i\leqslant k}\{level(\underline{a}_i)\}$. 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. For brevity, the application of the chase rule with a TGD σ on a database D is called application of σ on D.

Example A.1. Let $\mathcal{R} = \{r,s\}$. Consider the set Σ of TGDs over $\mathcal{R}$ constituted by the TGDs

$$\sigma_1 = r(X,Y), s(Y) \rightarrow \exists Z\, r(Z,X),$$
$$\sigma_2 = r(X,Y) \rightarrow s(X).$$

Let D be the database for $\mathcal{R}$ consisting of the two atoms $r(a,b)$ and $s(b)$. During the construction of $chase(D,\Sigma)$ we first apply σ_1, and we add the atom $r(z_1,a)$, where z_1 is a "fresh" null. Moreover, σ_2 is applicable and we add the atom $s(a)$. Now, σ_1 is applicable and the atom $r(z_2,z_1)$ is obtained, where z_2 is a "fresh" null. Also, σ_2 is applicable and the atom $s(z_1)$ is generated. It is clear that there is no finite chase. Satisfying both σ_1,σ_2 would require to construct the infinite instance $D \cup \{r(z_1,a), s(a), r(z_2,z_1), s(z_1), r(z_3,z_2), s(z_2), \ldots\}$. ■

The *chase of level up to* $k \geqslant 0$ for D and Σ, denoted as $chase^k(D,\Sigma)$, is the set of all atoms in $chase(D,\Sigma)$ of level at most k. We denote as $chase^{[k]}(D,\Sigma)$ the *initial segment* of the chase for D and Σ obtained by applying $k \geqslant 0$ times the TGD chase rule.

B. STICKY SETS OF TGDS

In the definition of sticky sets of TGDs, we consider a variable in the body of a TGD as marked, during the initial step of the marking procedure, if there exists some atom in the head in which it does not occur (and not if it does not exist in the head). This fact is crucial since without it, we would instead define a class that captures *lossless TGDs*, i.e., TGDs where every variable appearing in the body appears also in the head, for which query answering is undecidable.

THEOREM B.1. *BCQ answering under lossless TGDs is undecidable. The problem remains undecidable even in the case of atomic BCQs.*

PROOF. The proof is by reduction from the more general problem of BCQ evaluation problem under (general) TGDs which is undecidable. Let $\mathcal{R}$ be a relational schema. Consider a database D for $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. We construct from Σ a set Σ' of lossless TGDs such that Σ and Σ' are equivalent w.r.t. query answering. For each TGD $\sigma = \varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X},\mathbf{Z}) \in \Sigma$, if $\mathbf{Y}$ is not empty, then replace σ with the TGD $\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X},\mathbf{Z}), r_\sigma(\mathbf{Y})$, where r_σ is an auxiliary predicate symbol of arity m, where m is the number of variables in $\mathbf{Y}$. It is easy to see that Σ' is a set of lossless TGDs. Observe that, except for the atoms with an auxiliary predicate symbol, $chase(D,\Sigma)$ and $chase(D,\Sigma')$ coincide. The auxiliary predicates, being introduced only during the above construction, do not match any predicate symbol in q. Therefore, $chase(D,\Sigma) \models q$ iff $chase(D,\Sigma') \models q$. The claim follows. □

Moreover, according to the definition of sticky sets of TGDs, it is not allowed to have a marked variable that occurs more than once in the body of a TGD, even if it occurs only in one atom, i.e., it is not in a join; a variable in the body of a TGD is in a join, if it occurs more than once, but not in the same atom. Again this fact is critical since without it, we would instead define a class that captures *joinless TGDs*, i.e., TGDs where every variable in their body is not in a join, for which query answering is undecidable.

THEOREM B.2. *BCQ answering under joinless TGDs is undecidable. The problem remains undecidable even in the case of atomic BCQs.*

PROOF. The proof is by reduction from the more general problem of BCQ evaluation problem under (general) TGDs which is undecidable. Let $\mathcal{R}$ be a relational schema. Consider a database D for $\mathcal{R}$, a set Σ of TGDs over $\mathcal{R}$, and a BCQ q over $\mathcal{R}$. We construct from Σ a set Σ' of joinless TGDs such that Σ and Σ' are equivalent w.r.t. query answering. For each TGD $\sigma \in \Sigma$ (that does not have the desired syntactic property), if $\{X_1,\ldots,X_k\}$, for $k \geqslant 1$, is the set of all $\forall$–variables in $body(\sigma)$ that are in a join, and $\{Y_1,\ldots,Y_\ell\}$, for $\ell \geqslant 0$, is the set of all $\forall$–variables in $body(\sigma)$ that are not in a join, then replace σ with the following TGDs:

$$body(\sigma)' \rightarrow r_\sigma(X_1^1,\ldots,X_1^{n_1},\ldots,X_k^1,\ldots,X_k^{n_k},Y_1,\ldots,Y_\ell),$$
$$r_\sigma(\underbrace{X_1,\ldots,X_1}_{n_1},\ldots,\underbrace{X_k,\ldots,X_k}_{n_k},Y_1,\ldots,Y_\ell) \rightarrow head(\sigma),$$

where $body(\sigma)'$ is obtained from $body(\sigma)$ by replacing the j-th occurrence of X_i with X_i^j, r_σ is an auxiliary predicate, and n_i is the number of occurrences of X_i in $body(\sigma)$. For example, the TGD $\sigma = r(X,Y,Z,Y), s(X,Z) \rightarrow \exists W p(Y,W)$, will be replaced with the TGDs

$$r(X^1,Y,Z^1,Y), s(X^2,Z^2) \rightarrow r_\sigma(X^1,X^2,Z^1,Z^2,Y),$$
$$r_\sigma(X,X,Z,Z,Y) \rightarrow \exists W p(Y,W).$$

It is easy to see that Σ' is a set of joinless TGDs. Observe that, except for the atoms with an auxiliary predicate symbol, $chase(D,\Sigma)$ and $chase(D,\Sigma')$ coincide. The auxiliary predicates, being introduced only during the above construction, do not match any predicate symbol in q. Therefore, $chase(D,\Sigma) \models q$ iff $chase(D,\Sigma') \models q$. The claim follows. □

THEOREM B.3. *Consider a set Σ of TGDs over a schema $\mathcal{R}$. Σ is sticky iff for every database D for $\mathcal{R}$, $chase(D,\Sigma)$ has the sticky property.*

PROOF. Suppose first that Σ is sticky. Consider an arbitrary database D for $\mathcal{R}$. Let $\sigma \in \Sigma$ be a TGD such that

there exists a variable V which occurs in $body(\sigma)$ more than once. Note that if there is no such a TGD in Σ, then trivially $chase(D,\Sigma)$ has the sticky property. Suppose that during the construction of $chase(D,\Sigma)$ the TGD σ is applicable with homomorphism h, and the atom $\underline{a}$ is generated. Let A be the set of atoms

$$\{\underline{a}\} \cup \left\{\underline{a}' \mid \langle \underline{a}, \underline{a}' \rangle \text{ is in the transitive closure of } \xrightarrow{D,\Sigma}\right\}.$$

We need to show that the symbol $h(V)$ occurs in every atom $\underline{b} \in A$. The depth of an atom $\underline{b} \in A$ is defined as the cardinality of the *maximal* subset of $\xrightarrow{D,\Sigma}$ of the form

$$\left\{\langle \underline{a}, \underline{a}_1 \rangle, \langle \underline{a}_1, \underline{a}_2 \rangle, \ldots, \langle \underline{a}_{n-2}, \underline{a}_{n-1} \rangle, \langle \underline{a}_{n-1}, \underline{b} \rangle\right\},$$

where $n \geqslant 1$; the depth of $\underline{a}$ is zero. We denote as A_d, for $d \geqslant 0$, the subset of A that contains the atoms up to depth d. The proof is by induction on d. *Base Step.* Clearly, in A_0 we have only the atom $\underline{a}$. The symbol $h(V)$ occurs necessarily in $\underline{a}$, otherwise the variable V in $body(\sigma)$ is marked which is a contradiction since Σ is sticky. *Inductive Step.* Suppose that the symbol $h(V)$ occurs in every atom of the set $A_0 \cup \ldots \cup A_{d-1}$. We are going to show that occurs also in every atom of A_d. Consider an arbitrary atom $\underline{b} \in A_d$, obtained by applying some TGD $\sigma' \in \Sigma$ with homomorphism h'. Suppose that $h(V)$ does not occur in $\underline{b}$. This implies, by induction hypothesis, that there exists a variable W in $body(\sigma')$ such that $h(V) = h'(W)$, but W does not occur in $head(\sigma')$. Thus, the variable W in $body(\sigma')$ is marked. Hence, the variable V in $body(\sigma)$ is also marked, which is a contradiction since Σ is sticky. We conclude that $h(V)$ occurs also in $\underline{b}$.

Conversely, suppose that Σ is not sticky. We need to show that there exists a database D for $\mathcal{R}$ such that $chase(D,\Sigma)$ has not the sticky property. Clearly, since Σ is not sticky, there exists a TGD $\sigma \in \Sigma$ such that a variable V occurs in $body(\sigma)$ more than once, but V does not occur in $head(\sigma)$. Now, consider a database D for $\mathcal{R}$ such that there exists a homomorphism h from $body(\sigma)$ to D, and for any other variable W in $body(\sigma)$ it holds that $h(V) \neq h(W)$. Therefore, during the construction of $chase(D,\Sigma)$, the TGD σ is applicable with homomorphism h, and the atom $\underline{a}$ is obtained. Observe that the symbol $h(V)$ does not occur in $\underline{a}$. This implies that $chase(D,\Sigma)$ does not have the sticky property. The proof is now complete. □

THEOREM B.4. *The lossless datalog fact inference problem is* EXPTIME*-hard.*

PROOF. The proof is by reduction from the more general problem of datalog fact inference. We recall that this problem is EXPTIME-complete, even if we restrict our attention on databases over the domain $\{0,1\}$ (see, e.g., [19]). Consider an instance $\langle \mathcal{R}, D, \mathcal{P}, \underline{a} \rangle$ of the datalog fact inference problem, where D is a database for $\mathcal{R}$, $\mathcal{P}$ is a datalog program over $\mathcal{R}$, and $\underline{a}$ is a ground atom. Let $dom(D) = \{0,1\}$. We construct an instance $\langle \mathcal{R}^\star, D^\star, \mathcal{P}^\star, \underline{a}^\star \rangle$ of the lossless datalog fact inference problem as follows.

- For each predicate $r/m \in \mathcal{R}$, we add in $\mathcal{R}^\star$ a predicate $r^\star$ of arity $m+n+2$, where $n \geqslant 1$ is the maximum number of variables in the body of any rule in $\mathcal{P}$.
- $D^\star = \{r^\star(c_1,\ldots,c_k,\mathbf{0},0,1) \mid r(c_1,\ldots,c_k) \in D\}$, where $\mathbf{0} = \{0\}^n$ (sequence of n zeros).
- For each rule $\sigma = r(\mathbf{X}) \leftarrow r_1(\mathbf{X}_1),\ldots,r_m(\mathbf{X}_m)$ in $\mathcal{P}$ we add in $\mathcal{P}^\star$ the rule

$$r^\star(\mathbf{X},Y_1,\ldots,Y_n,Z_0,Z_1) \leftarrow r_1^\star(\mathbf{X}_1,\mathbf{Z_0},Z_0,Z_1),\ldots, r_m^\star(\mathbf{X}_m,\mathbf{Z_0},Z_0,Z_1),$$

  where $\mathbf{Z_0} = \{Z_0\}^n$, and if $\{V_1,\ldots,V_\ell\}$, for $1 \leqslant \ell \leqslant n$, is the set of variables in the body of σ (the order is not relevant), then $Y_i = V_i$, for each $i \in \{1,\ldots,\ell\}$, and $Y_j = V_1$, for each $j \in \{\ell+1,\ldots,n\}$. Intuitively, by adding $\sigma^\star$ we actually convert the rule σ to a lossless datalog rule. Note that the variable Z_0 (resp., Z_1) is a placeholder for the value 0 (resp., 1). Clearly, each atom generated during the inference process due to $\sigma^\star$ can be of the form $r^\star(c_1,\ldots,c_k,\mathbf{B},0,1)$, where $\mathbf{B} \in \{0,1\}^n$. In case that $\mathbf{B} \neq \mathbf{0}$, then we need to ensure that the atom $r^\star(c_1,\ldots,c_k,\mathbf{0},0,1)$ will be generated. This can be achieved by adding the following rules. For each predicate $r^\star \in \mathcal{R}^\star$, if m is the arity of $r \in \mathcal{R}$, then, for each $i \in \{1,\ldots,n\}$, we add in $\mathcal{P}^\star$ the rule

$$r^\star(X_1,\ldots,X_m,Y_1,\ldots,Y_{i-1},Z_0,Y_{i+1},\ldots,Y_n,Z_0,Z_1) \leftarrow r^\star(X_1,\ldots,X_m,Y_1,\ldots,Y_{i-1},Z_1,Y_{i+1},\ldots,Y_n,Z_0,Z_1).$$

- If $\underline{a} = r(c_1,\ldots,c_k)$, then $\underline{a}^\star = r^\star(c_1,\ldots,c_k,\mathbf{0},0,1)$.

It is straightforward to see that $\mathcal{P}^\star$ is indeed a lossless datalog program. Furthermore, it is clear that the above transformation can be done in polynomial time. In what follows we show that for any ground atom $\underline{b} = r(c_1,\ldots,c_k)$, $D \cup \mathcal{P} \models \underline{b}$ iff $D^\star \cup \mathcal{P}^\star \models \underline{b}^\star$, or, equivalently, $chase(D,\mathcal{P}) \models \underline{b}$ iff $chase(D^\star,\mathcal{P}^\star) \models \underline{b}^\star$, where $\underline{b}^\star = r^\star(c_1,\ldots,c_k,\mathbf{0},0,1)$.

Suppose first that $chase(D,\mathcal{P}) \models \underline{b}$. We are going to show that $chase(D^\star,\mathcal{P}^\star) \models \underline{b}^\star$. We proceed by induction on the number of applications of the TGD chase rule during the construction of $chase(D,\mathcal{P})$. *Base Step.* Clearly, since $chase^{[0]}(D,\mathcal{P}) = D$, the claim follows immediately by construction. *Inductive Step.* Suppose that $chase^{[i]}(D,\mathcal{P}) \models \underline{b}$. If $\underline{b} \in chase^{[i-1]}(D,\mathcal{P})$, then the claim follows by induction hypothesis. Now, suppose that $\underline{b}$ is the atom generated during the i-th application of the TGD chase rule due to the TGD $\sigma = r_1(\mathbf{X}_1),\ldots,r_m(\mathbf{X}_m) \rightarrow r(\mathbf{X})$. This implies that there exists a homomorphism h such that $h(body(\sigma)) \subseteq chase^{[i]}(D,\mathcal{P})$, and $h(head(\sigma)) = \underline{b}$. Since the set of ground atoms $\{r_j(h(\mathbf{X}_j))\}_{1 \leqslant j \leqslant m}$ is a subset of $chase^{[i-1]}(D,\mathcal{P})$, by induction hypothesis, the set of atoms $\{r_j^\star(h(\mathbf{X}_j),\mathbf{0},0,1)\}_{1 \leqslant j \leqslant m}$ is a subset of $chase(D^\star,\mathcal{P}^\star)$. Moreover, by construction, the TGD $\sigma^\star$

$$r_1^\star(\mathbf{X}_1,\mathbf{Z_0},Z_0,Z_1),\ldots, r_m^\star(\mathbf{X}_m,\mathbf{Z_0},Z_0,Z_1) \rightarrow r^\star(\mathbf{X},Y_1,\ldots,Y_n,Z_0,Z_1)$$

is in $\mathcal{P}^\star$. Let $h^\star = h \cup \{Z_0 \rightarrow 0, Z_1 \rightarrow 1\}$. Observe that $h^\star(body(\sigma^\star)) = \{r_j^\star(h(\mathbf{X}_j),\mathbf{0},0,1)\}_{1 \leqslant j \leqslant m} \subseteq chase(D^\star,\mathcal{P}^\star)$, and thus $h^\star(head(\sigma^\star)) = r^\star(h(\mathbf{X}),h(Y_1),\ldots,h(Y_n),0,1) \in chase(D^\star,\mathcal{P}^\star)$. If $h(Y_j) = 0$, for each $j \in \{1,\ldots,n\}$, then $\underline{b}^\star \in chase(D^\star,\mathcal{P}^\star)$, as needed. Now, suppose that there exists $j \in \{1,\ldots,n\}$ such that $h(Y_j) = 1$. By construction, if m is the arity of $r \in \mathcal{R}$, then the TGD

$$r^\star(X_1,\ldots,X_m,Y_1,\ldots,Y_{j-1},Z_1,Y_{j+1},\ldots,Y_n,Z_0,Z_1) \rightarrow r^\star(X_1,\ldots,X_m,Y_1,\ldots,Y_{j-1},Z_0,Y_{j+1},\ldots,Y_n,Z_0,Z_1)$$

is in $\mathcal{P}^\star$. Hence, the atom

$$r^\star(h(\mathbf{X}),h(Y_1),\ldots,h(Y_{j-1}),0,h(Y_{j+1}),\ldots,h(Y_n),0,1)$$

occurs in $chase(D^\star, \mathcal{P}^\star)$. It is straightforward to see that eventually the atom $\underline{b}^\star = r^\star(h(\mathbf{X}), \mathbf{0}, 0, 1)$ is generated, and the claim follows.

Conversely, assume that $chase(D^\star, \mathcal{P}^\star) \models \underline{b}^\star$. We are going to show that $chase(D, \mathcal{P}) \models \underline{b}$. The proof is by induction on the number of applications of the TGD chase rule during the construction of $chase(D^\star, \mathcal{P}^\star)$. *Base Step.* Clearly, since $chase^{[0]}(D^\star, \mathcal{P}^\star) = D^\star$, the claim follows immediately by construction. *Inductive Step.* Suppose that $chase^{[i]}(D^\star, \mathcal{P}^\star) \models \underline{b}^\star$. If $\underline{b}^\star \in chase^{[i-1]}(D^\star, \mathcal{P}^\star)$, then the claim follows by induction hypothesis. Now, assume that $\underline{b}^\star$ is the atom generated during the i-th application of the TGD chase rule. There exists an atom $\underline{c} \in chase^{[i]}(D^\star, \mathcal{P}^\star)$ of the form $r^\star(c_1, \ldots, c_k, \mathbf{B}, 0, 1)$ obtained due to a TGD $\sigma^\star$ of the form

$$r_1^\star(\mathbf{X}_1, \mathbf{Z_0}, Z_0, Z_1), \ldots, \\ r_m^\star(\mathbf{X}_m, \mathbf{Z_0}, Z_0, Z_1) \rightarrow r^\star(\mathbf{X}, Y_1, \ldots, Y_n, Z_0, Z_1),$$

such that either $\underline{c} = \underline{b}^\star$ or the set of pairs

$$\left\{\langle \underline{c}, \underline{c}_1\rangle, \langle \underline{c}_1, \underline{c}_2\rangle, \ldots, \langle \underline{c}_{j-2}, \underline{c}_{j-1}\rangle, \langle \underline{c}_{j-1}, \underline{b}^\star\rangle\right\}$$

is a subset of $\xrightarrow{D,\Sigma}$, and each pair occurs in $\xrightarrow{D,\Sigma}$ due to a TGD of the form

$$r^\star(X_1, \ldots, X_k, Y_1, \ldots, Y_{j-1}, Z_1, Y_{j+1}, \ldots, Y_n, Z_0, Z_1) \rightarrow \\ r^\star(X_1, \ldots, X_k, Y_1, \ldots, Y_{j-1}, Z_0, Y_{j+1}, \ldots, Y_n, Z_0, Z_1),$$

for some $j \in \{1, \ldots, n\}$. This implies that there exists a homomorphism h such that $h(body(\sigma^\star)) \subseteq chase^{[i-1]}(D^\star, \mathcal{P}^\star)$, and $h(head(\sigma^\star)) = \underline{c}$. Since the set of ground atoms $\{r_j^\star(h(\mathbf{X}_j), \mathbf{0}, 0, 1)\}_{1 \leqslant j \leqslant m}$ is a subset of $chase^{[i-1]}(D^\star, \mathcal{P}^\star)$, by induction hypothesis, the set of atoms $\{r_j(h(\mathbf{X}_j))\}_{1 \leqslant j \leqslant m}$ is a subset of $chase(D, \mathcal{P})$. Furthermore, by construction, the TGD $\sigma = r_1(\mathbf{X}_1), \ldots, r_m(\mathbf{X}_m) \rightarrow r(\mathbf{X})$ occurs in $\mathcal{P}$. Clearly, $h(body(\sigma)) = \{r_j(h(\mathbf{X}_j))\}_{1 \leqslant j \leqslant m} \subseteq chase(D, \mathcal{P})$. Since $chase(D, \mathcal{P})$ satisfies all the TGDs in $\mathcal{P}$, it follows that the atom $h(head(\sigma)) = r(h(\mathbf{X})) = \underline{b} \in chase(D, \mathcal{P})$, as needed. The proof is now complete. □

Algorithm QAns. QAns has as input a BCQ q over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, a sticky set Σ of TGDs over $\mathcal{R}$, and outputs "Accept" iff $D \cup \Sigma \models q$, or, equivalently, $chase(D, \Sigma) \models q$. The algorithm works as follows:

1. Non-deterministically guess a substitution $\mu : S_1 \rightarrow S_2 \cup dom(D)$, where $S_1 \subseteq var(q)$ and $S_2 \subset var(q)$ are disjoint sets. Let $S = var(q) \setminus S_1$. Roughly, the algorithm guesses which of the variables in the body of q, during the evaluation of q over the $chase(D, \Sigma)$, are mapped onto a constant of $dom(D)$, and which constant, and also which of these variables are mapped onto the same null.
2. For each variable $V \in S$, QAns non-deterministically guesses an atom $\underline{a}_V$. Let $G = \{\langle V, \underline{a}_V\rangle\}_{V \in S}$. Intuitively, QAns guesses, for each variable $V \in S$, an atom $\underline{a}_V$ (possibly containing constant values) that represents the atom in which z_V is invented during the chase, where z_V is the null onto which V is mapped.
3. Halt and "Accept" iff $\mathsf{Check}(q, D, \Sigma, \mu, S, G)$ accepts.

Check. Intuitively, the subroutine Check verifies, for each variable V in the body of q that is mapped onto some null during the evaluation of q over the $chase(D, \Sigma)$, that each occurrence of V is mapped onto the same null.

1. If $\mu(body(q)) \subseteq D$, then halt and output "Accept"; otherwise, universally select all atoms $\underline{a}$ in $h(body(q))$ that do not occur in D.
2. Let $G_{\underline{a}} = \{\langle V, \underline{a}_V\rangle \mid V \in dom(\underline{a})\} \subseteq G$.
3. Existentially guess a TGD $\sigma \in \Sigma$ such that there exists a homomorphism h from the head of σ to $\underline{a}$. We replace with "$\star$" the variables that occur in the body of σ but not in the head. Note that each of these variables occur only once in $body(\sigma)$, since Σ is sticky.
4. If at some position π in $\underline{a}$ some variable V occurs, and at the same position π in $head(\sigma)$ we have an $\exists$–variable then, if $\underline{a}_V$ can be obtained from $\underline{a}$ by replacing "$\star$" (if any) with other terms, then halt and output "Accept"; otherwise, halt and output "Reject".
5. If $h(\sigma)$, after replacing "$\star$" with constants, is a subset of D, then halts and "Accept"; otherwise, universally select all atoms $\underline{a}$ in $h(\sigma)$ from which we cannot obtain an atom in D by substituting "$\star$" with constants, and goto step (2).

C. ADDING EGDS

FD Chase Rule: Consider a database D for a schema $\mathcal{R}$, and a FD ϕ of the form $r : \mathbf{A} \rightarrow \mathbf{B}$ over $\mathcal{R}$. If ϕ is *applicable* to D, i.e., there exist two (distinct) tuples $\mathbf{t_1}, \mathbf{t_2} \in r(D)$ such that $\mathbf{t_1}[\mathbf{A}] = \mathbf{t_2}[\mathbf{A}]$ and $\mathbf{t_1}[\mathbf{B}] \neq \mathbf{t_2}[\mathbf{B}]$, then for each attribute $i \in \mathbf{B}$ such that $\mathbf{t_1}[i] \neq \mathbf{t_2}[i]$: *(i)* if $\mathbf{t_1}[i]$ and $\mathbf{t_2}[i]$ are both constants of Γ, then there is a *hard violation* of ϕ and the chase *fails*, *(ii)* replace each occurrence of $\mathbf{t_2}[i]$ with $\mathbf{t_1}[i]$, if $\mathbf{t_1}[i]$ precedes $\mathbf{t_2}[i]$ in the lexicographic order, or vice-versa otherwise.

In Section 4, for technical reasons, we make use of the *restricted* TGD chase rule. The difference between the oblivious and the restricted TGD chase rules is that the latter introduces a new atom only when the TGD is not satisfied. Formally, given a database D for a schema $\mathcal{R}$, and a TGD $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$, σ is *applicable* to D iff there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, and there is no extension h' of h such that $h'(r(\mathbf{X}, \mathbf{Z})) \in D$. Note that, under TGDs, both the restricted and the oblivious chase are universal models [9].

Example C.1. Let $\mathcal{R}$ be the schema given in Example 2, and Σ_T be the set of TGDs over $\mathcal{R}$ constituted by the TGDs:

$$\sigma_1 = runs(W, X), in\_area(X, Y) \rightarrow \exists Z\, emp(Z, W, Y, X),$$
$$\sigma_2 = emp(V, W, X, Y) \rightarrow \exists Z\, dept(W, Z).$$

The fact that each department has a unique id can be expressed by the FD:

$$\phi = dept : \{1\} \rightarrow \{2\}.$$

Let D be the database for $\mathcal{R}$ consisting of the atoms $runs(d, p)$, $in\_area(p, a)$ and $dept(d, m)$. Observe that during the first application of the TGD chase rule we get the atom $emp(z_1, d, a, p)$, where $z_1 \in \Gamma_f$. Then, σ_2 is applicable and we get the atom $dept(d, z_2)$, where $z_2 \in \Gamma_f$. Now, observe that the FD ϕ is applicable. Therefore, we substitute z_2 with m, and the atom $dept(d, z_2)$ is eliminated. ■

Theorem C.1. *Let $\mathcal{R}$ be a relational schema. Consider a set $\Sigma = \Sigma_T \cup \Sigma_F$ over $\mathcal{R}$, where Σ_T and Σ_F are sets of TGDs and FDs, respectively. If Σ is non-conflicting, then it is separable.*

PROOF. Let D be a database for $\mathcal{R}$ such that D satisfies Σ_F, otherwise the claim holds trivially. We need to show that *(i)* $chase(D,\Sigma)$ does not fail, and *(ii)* for every BCQ q over $\mathcal{R}$, $chase(D,\Sigma) \models q$ iff $chase(D,\Sigma_T) \models q$. To establish the latter statement it suffices to show that $chase(D,\Sigma)$ and $chase(D,\Sigma_T)$ are homomorphically equivalent, i.e., there exists a homomorphism from $chase(D,\Sigma)$ to $chase(D,\Sigma_T)$, and vice-versa. The existence of a homomorphism from $chase(D,\Sigma_T)$ to $chase(D,\Sigma)$ is trivial. It remains to show that $chase(D,\Sigma)$ does not fail, and also that there exists a homomorphism from $chase(D,\Sigma)$ to $chase(D,\Sigma_T)$.

The proof is by induction on the number of applications of the chase rule in the construction of $chase(D,\Sigma)$, i.e., a single application of a TGD, and then exhaustively applying FDs. We need to prove that for each $i \geqslant 0$, $chase^{[i]}(D,\Sigma)$ does not fail, and there exists a homomorphism h_i such that $h_i(chase^{[i]}(D,\Sigma)) \subseteq chase(D,\Sigma_T)$. *Base Step.* Since D satisfies Σ_F we get that $chase^{[0]}(D,\Sigma)$ does not fail. Moreover, since $D \subseteq chase(D,\Sigma_T)$ there exists trivially a homomorphism h_0 that maps $chase^{[0]}(D,\Sigma)$ to $chase(D,\Sigma_T)$; in fact, h_0 is the identity homomorphism. *Inductive Step.* Suppose that during the i-th application of the chase rule we apply the TGD $\sigma = \varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}\, r(\mathbf{X},\mathbf{Z})$. This implies that there exists a homomorphism λ that maps $\varphi(\mathbf{X},\mathbf{Y})$ to a set of atoms of $chase^{[i-1]}(D,\Sigma)$, and the atom $\underline{a} = \lambda'(r(\mathbf{X},\mathbf{Z}))$ is obtained, where λ' is an extension of λ as in the TGD chase rule. Before we proceed further, we establish the following auxiliary lemma.

LEMMA C.2. *There is no FD in Σ_F that is applicable to $chase^{[i-1]}(D,\Sigma) \cup \{\underline{a}\}$.*

PROOF. Consider an arbitrary FD $\phi \in \Sigma_F$ of the form $r : \mathbf{A} \rightarrow \mathbf{B}$. Suppose that $\underline{a} = r(\mathbf{t})$. It suffices to show that there is no tuple $\mathbf{t}' \in r(chase^{[i-1]}(D,\Sigma))$ such that $\mathbf{t}[\mathbf{A}] \neq \mathbf{t}'[\mathbf{A}]$. According to the Definition 4, we identity the following four cases:

- $\mathbf{U}_\sigma = \mathbf{A}$. Suppose, in the sake of contradiction, that $\mathbf{t}[\mathbf{A}] = \mathbf{t}'[\mathbf{A}]$. Since each existentially quantified variable in σ occurs just once, it follows that σ is not applicable with homomorphism λ which is a contradiction (recall that we consider the restricted chase). Therefore, $\mathbf{t}[\mathbf{A}] \neq \mathbf{t}'[\mathbf{A}]$.
- $\mathbf{U}_\sigma \subset \mathbf{A}$. Clearly, $\mathbf{A} \setminus \mathbf{U}_\sigma \neq \varnothing$. This implies that $\mathbf{t}[\mathbf{A}]$ contains nulls introduced during the i-th application of the chase rule, and thus $\mathbf{t}[\mathbf{A}] \neq \mathbf{t}'[\mathbf{A}]$.
- $\mathbf{U}_\sigma \cap \mathbf{A} \neq \varnothing$ with $\mathbf{U}_\sigma \setminus \mathbf{A}$. Necessarily $\mathbf{A} \setminus \mathbf{U}_\sigma \neq \varnothing$, otherwise $\mathbf{U}_\sigma$ is a strict superset of $\mathbf{A}$ which is not allowed by Definition 4. Hence, $\mathbf{t}[\mathbf{A}] \neq \mathbf{t}'[\mathbf{A}]$.
- $\mathbf{U}_\sigma \cap \mathbf{A} = \varnothing$. Obviously, $\mathbf{A} \setminus \mathbf{U}_\sigma \neq \varnothing$, and thus $\mathbf{t}[\mathbf{A}] \neq \mathbf{t}'[\mathbf{A}]$.

The claim follows. □

By Lemma C.2 we immediately get that $chase^{[i]}(D,\Sigma) = chase^{[i-1]}(D,\Sigma) \cup \{\underline{a}\}$; thus, $chase^{[i]}(D,\Sigma)$ does not fail. There exists a homomorphism $\mu = h_{i-1} \circ \lambda$ that maps $\varphi(\mathbf{X},\mathbf{Y})$ to $chase(D,\Sigma_T)$. Since $chase(D,\Sigma_T)$ satisfies Σ_T it follows that there exists $\mu' \supseteq \mu$ that maps $\mu'(r(\mathbf{X},\mathbf{Z}))$ to a set of atoms of $chase(D,\Sigma_T)$. Denoting $\mathbf{Z} = Z_1,\ldots,Z_m$, we define the substitution

$$h_i = h_{i-1} \cup \{\lambda'(Z_i) \rightarrow \mu'(Z_i)\}_{1 \leqslant i \leqslant m}.$$

Obviously h_i is a well-defined substitution. Observe that $h_i(\underline{a}) = h_i(\lambda'(r(\mathbf{X},\mathbf{Z}))) = r(h_{i-1}(\lambda(\mathbf{X})), h_i(\lambda'(\mathbf{Z}))) = r(\mu(\mathbf{X}),\mu'(\mathbf{Z})) = \mu'(r(\mathbf{X},\mathbf{Z})) \in chase(D,\Sigma_T)$. Therefore, $h_i(chase^{[i]}(D,\Sigma)) \subseteq chase(D,\Sigma_T)$, as needed.

Eventually the desired homomorphism from $chase(D,\Sigma)$ to $chase(D,\Sigma_T)$ is $h = \bigcup_{i=0}^{\infty} h_i$. □

D. WEAKLY-STICKY SETS OF TGDS

In this final section we give the definition of a class, called weakly-sticky sets of TGDs, that generalizes both sticky sets and weakly-acyclic sets of TGDs [21, 22], for which conjunctive query answering is decidable.

We first recall the notion of the dependency graph, as defined by Fagin et al. [22]. Given a set Σ of TGDs over a schema $\mathcal{R}$, the *dependency graph* of Σ is the directed graph constructed as follows. There exists a node for each position $r[i]$ in $\mathcal{R}$, where $r/n \in \mathcal{R}$ and $i \in \{1,\ldots,n\}$. For each TGD $\sigma \in \Sigma$, for each $\forall$–variable V in $head(\sigma)$, and for each occurrence of V in $body(\sigma)$ at position $r[i]$, apply the following two steps:

1. For each occurrence of V in $head(\sigma)$ at position $s[j]$, add an arc from $r[i]$ to $s[j]$ (if it does not already exist).
2. For each $\exists$–variable W, and for each occurrence of W in $head(\sigma)$ at position $t[k]$, add a *special* arc from $r[i]$ to $t[k]$ (if it does not already exist).

The *rank* of a position $r[i]$ is the maximum number of special arcs over all (finite or infinite) paths ending at $r[i]$. The set of positions in $\mathcal{R}$ can be partitioned into two sets $\Pi_F(\mathcal{R},\Sigma)$ and $\Pi_\infty(\mathcal{R},\Sigma)$, where $\Pi_F(\mathcal{R},\Sigma)$ (resp., $\Pi_\infty(\mathcal{R},\Sigma)$) is the set of positions with finite (resp., infinite) rank. Intuitively, $\Pi_F(\mathcal{R},\Sigma)$ (resp., $\Pi_\infty(\mathcal{R},\Sigma)$) is the set of position where a finite (resp., infinite) number of nulls can appear. We are now ready to define our extended version of sticky sets of TGDs.

Definition D.1. Consider a set Σ of TGDs over a schema $\mathcal{R}$. We say that Σ is *weakly-sticky* iff for each $\sigma \in \Sigma$ and for each variable V that occurs more than once in $body(\sigma)$, at least one of the following conditions holds: *(i)* V is a non-marked variable[2], or *(ii)* at least one occurrence of V in $body(\sigma)$ occurs at some position in $\Pi_F(\mathcal{R},\Sigma)$.

Recall that a set Σ of TGDs over a schema $\mathcal{R}$ is weakly-acyclic iff all positions of $\mathcal{R}$ are in $\Pi_F(\mathcal{R},\Sigma)$. This immediately implies that every weakly-acyclic set of TGDs is also weakly-sticky.

THEOREM D.1. *BCQ answering under weakly-sticky sets of TGDs is decidable.*

For complexity results on conjunctive query answering under this extended class we refer the reader to [11].

[2]Marked variables are defined as in the Definition 1.