Rewriting Ontological Queries into Small Nonrecursive Datalog Programs*

Georg Gottlob[1] and Thomas Schwentick[2]

[1] Department of Computer Science, University of Oxford gottlob@cs.ox.ac.uk
[2] Fakultät für Informatik, TU Dortmund thomas.schwentick@udo.edu

Abstract. We consider the setting of ontological database access, where an A-box is given in form of a relational database D and where a Boolean conjunctive query q has to be evaluated against D modulo a T-box Σ formulated in DL-Lite or Linear Datalog$^{\pm}$. It is well-known that (Σ, q) can be rewritten into an equivalent nonrecursive Datalog program P that can be directly evaluated over D. However, for Linear Datalog$^{\pm}$ or for DL-Lite versions that allow for role inclusion, the rewriting methods described so far result in a nonrecursive Datalog program P of size exponential in the joint size of Σ and q. This gives rise to the interesting question of whether such a rewriting necessarily needs to be of exponential size. In this paper we show that it is actually possible to translate (Σ, q) into a polynomially sized equivalent nonrecursive Datalog program P.

1 Introduction

This paper is about query rewriting in the context of ontological database access. Query rewriting is an important new optimization technique specific to ontological queries. The essence of query rewriting, as will be explained in more detail below, is to compile a query and an ontological theory (usually formulated in some description logic or rule-based language) into a target query language that can be directly executed over a relational database management system (DBMS). The advantage of such an approach is obvious. Query rewriting can be used as a preprocessing step for enabling the exploitation of mature and efficient existing database technology to answer ontological queries. In particular, after translating an ontological query into SQL, sophisticated query-optimization strategies can be used to efficiently answer it. However, there is a pitfall here. If the translation inflates the query excessively and creates from a reasonably sized ontological query an enormous exponentially sized SQL query (or SQL DDL program), then the best DBMS may be of little use.

Main results. We show that polynomially sized query rewritings into nonrecursive Datalog exist in specific settings. Note that nonrecursive Datalog can be efficiently translated into SQL with view definitions (SQL DDL), which, in turn, can be directly executed over any standard DBMS. Our results are — for the time being — of theoretical nature and we do not claim that they will lead to better practical algorithms. This will be studied via implementations in the next future. Our main result applies to the

* Future improvements and extended versions of this paper will be published in arXive-CORR at http://arxiv.org/abs/1106.3767

setting where ontological constraints are formulated in terms of *tuple-generating dependencies (tgds)*, and we make heavy use of the well-known *chase* procedure [17, 14]. For definitions, see Section 2. The result after chasing a tgd set Σ over a database D is denoted by $chase(D, \Sigma)$.

Consider a set Σ of tgds and a database D over a joint signature $\mathcal{R}$. Let q be a Boolean conjunctive query (BCQ) issued against (D, Σ). We would like to transform q into a nonrecursive Datalog query P such that $(D, \Sigma) \models q$ iff $D \models P$. We assume here that P has a special propositional goal $goal$, and $D \models P$ means that $goal$ is derivable from P when evaluated over D. Let us define an important property of classes of tgds.

Definition 1. Polynomial witness property (PWP). *The PWP holds for a class $\mathcal{C}$ of tgds if there exists a polynomial γ such that, for every finite set $\Sigma \subseteq \mathcal{C}$ of tgds and each BCQ q, the following holds: for each database D, whenever $(D, \Sigma) \models q$, then there is a sequence of at most $\gamma(|\Sigma|, |q|)$ chase steps whose atoms already entail q.*

Our main technical result, which is more formally stated in Section 3, is as follows.

Theorem 1. *Let Σ be a set of tgds from a class $\mathcal{C}$ enjoying the PWP. Then each BCQ q can be rewritten in polynomial time into a nonrecursive Datalog program P of size polynomial in the joint size of q and Σ, such that for every database D, $(D, \Sigma) \models q$ if and only if $D \models P$. Moreover, the arity of P is $\max(a + 2, 9)$, where a is the maximum arity of any predicate symbol occurring in Σ, in case a sufficiently large linear order can be accessed in the database, or otherwise by $O(\max(a + 2, 9) \cdot \log m)$, where m is the joint size of q and Σ.*

Other Results. From this result, and from already established facts, a good number of further rewritabliity results for other formalisms can be derived. In particular, we can show that conjunctive queries based on other classes of tgds or description logics can be efficiently translated into nonrecursive Datalog. Among these formalisms are: linear tgds, originally defined in [5] and equivalent to inclusion dependencies, various major versions of the well-known description logic DL-Lite [9, 20], and sticky tgds [8] as well as sticky-join tgds [6, 7]. For space reasons, we will just give an overview and very short explanations of how each of these rewritability results follows from our main theorem.

Structure of the Paper. The rest of the paper is structured as follows. In Section 2 we state a few preliminaries and simplifying assumptions. In Section 3, we explain the idea of the proof of the main result. Section 4, contains the other results following from the main result. A brief overview of related work concludes the paper in Section 5.

2 Preliminaries and Assumptions

We assume the reader to be familiar with the terminology of relational databases and the concepts of *conjunctive query (CQ)* and *Boolean conjunctive query (BCQ)*. For simplicity, we restrict our attention to Boolean conjunctive queries q. However, our results can easily be reformulated for queries with output, see the extended version of this paper [13]).

Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (tgd)* σ is a first-order formula of the form $\forall \boldsymbol{X} \forall \boldsymbol{Y}\, \Phi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\, \Psi(\boldsymbol{X}, \boldsymbol{Z})$, where $\Phi(\boldsymbol{X}, \boldsymbol{Y})$ and $\Psi(\boldsymbol{X}, \boldsymbol{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted $body(\sigma)$

and $head(\sigma)$, respectively. We usually omit the universal quantifiers in tgds. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\Phi(\boldsymbol{X}, \boldsymbol{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\boldsymbol{X}, \boldsymbol{Z})$ to atoms of D. All sets of tgds are finite here. We assume in the rest of the paper that every tgd has exactly one atom and at most one existentially quantified variable in its head. A set of tgds is in *normal form* if the head of each tgd consists of a single atom. It was shown in [4, Lemma 10] that every set Σ of TGDs can be transformed into a set Σ' in normal form of size at most quadratic in $|\Sigma|$, such that Σ and Σ' are equivalent with respect to query answering. The normal form transformation shown in [4] can be achieved in logarithmic space. It is, moreover, easy to see that this very simple transformation preserves the polynomial witness property.

For a database D for $\mathcal{R}$, and a set of tgds Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $mods(D, \Sigma)$, is the set of all (possibly infinite) databases B such that (i) $D \subseteq B$ and (ii) every $\sigma \in \Sigma$ is satisfied in B. The set of *answers* for a CQ q to D and Σ, denoted $ans(q, D, \Sigma)$, is the set of all tuples $\underline{a}$ such that $\underline{a} \in q(B)$ for all $B \in mods(D, \Sigma)$. The *answer* for a BCQ q to D and Σ is *yes* iff the empty tuple is in $ans(q, D, \Sigma)$, also denoted as $D \cup \Sigma \models q$.

Note that, in general, query answering under tgds is undecidable [2], even when the schema and tgds are fixed [4]. Query answering is, however, decidable for interesting classes of tgds, among which are those considered in the present paper.

The *chase* procedure [17, 14] uses the following *oblivious* chase rule.

TGD CHASE RULE. Consider a database D for a relational schema $\mathcal{R}$, and a tgd σ on $\mathcal{R}$ of the form $\Phi(\boldsymbol{X}, \boldsymbol{Y}) \rightarrow \exists \boldsymbol{Z}\, \Psi(\boldsymbol{X},\, \boldsymbol{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Phi(\boldsymbol{X}, \boldsymbol{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \boldsymbol{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \boldsymbol{Z}$, $h_1(Z_j) = z_j$, where z_j is a fresh null value (i.e., a Skolem constant) different from all nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\boldsymbol{X}, \boldsymbol{Z}))$ if not already in D (which is possible when $\boldsymbol{Z}$ is empty). ■

The chase algorithm for a database D and a set of tgds Σ consists of an exhaustive application of the tgd chase rule in a breadth-first (level-saturating) fashion, which leads as result to a (possibly infinite) chase for D and Σ. Each atom from the database D is assigned a *derivation level*. Atoms in D have derivation level 0. If an atom has not already derivation level $\leq i$ but can be obtained by a single application of a tgd via the chase rule from atoms having derivation level $\leq i$, then its derivation level is $i+1$. The set of all atoms of derivation level $\leq k$ is denoted by $chase^k(D, \Sigma)$. The *chase* of D relative to Σ, denoted $chase(D, \Sigma)$, is then the limit of $chase^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase relative to tgds is a *universal model*, i.e., there exists a homomorphism from $chase(D, \Sigma)$ onto every $B \in mods(D, \Sigma)$ [11, 4]. This result implies that BCQs q over D and Σ can be evaluated on the chase for D and Σ, i.e., $D \cup \Sigma \models q$ is equivalent to $chase(D, \Sigma) \models q$.

A *chase sequence* of length n based on D and Σ is a sequence of n atoms such that each atom is either from D or can be derived via a single application of some rule in Σ from previous atoms in the sequence. If S is such a chase sequence and q a conjunctive query, we write $S \models q$ if there is a homomorphism from q to the set of atoms of S.

We assume that every database has two constants, 0 and 1, that are available via the unary predicates $Zero$ and One, respectively. Moreover, each database has a binary predicate Neq such that $\mathrm{Neq}(a, b)$ is true precisely if a and b are distinct values.

We finally define *N-numerical databases*. Let D be a database whose domain does not contain any natural numbers. We define D_N as the extension of D by adding the natural numbers $0, 1, \ldots, N$ to its domain, a unary relation Num that contains exactly the numbers $1, \ldots, N$, binary order relations $Succ$ and $<$ on $0, 1, \ldots, N$, expressing the natural successor and "$<$" orders on N, respectively. [3] We refer to D_N as the *N-numerical extension* of D, and, a so extended database as *N-numerical database*. We denote the total domain of a numerical database D_N by $\mathrm{dom}_N(D)$ and the non-numerical domain (still) by $\mathrm{dom}(D)$. Standard databases can always be considered to be N-numerical, for some large N by the standard type *integer*, with the $<$ predicate (and even arithmetic operations). A number $maxint$ corresponding to N can be defined.

3 Main Result

Our main result is more formally stated as follows:

Theorem 1. *Let $\mathcal{C}$ be a class of tgds in normal form, enjoying the polynomial witness property and let γ be the polynomial bounding the number of chase steps (with $\gamma(n_1, n_2) \geq \max(n_1, n_2)$, for all naturals n_1, n_2). For each set $\Sigma \subseteq \mathcal{C}$ of tgds and each Boolean CQ q, one can compute in polynomial time a nonrecursive Datalog program P of polynomial size in $|\Sigma|$ and $|q|$, such that, for every database D it holds $D, \Sigma \models q$ if and only if $D \models P$. Furthermore:*

- *(a) For N-numerical databases D, where $N \geq \gamma(|\Sigma|, |q|)$, the arity of P is $\max(a + 2, 9)$, where a is the maximum arity of any predicate symbol occurring in Σ;*
- *(b) otherwise (for non-numerical databases), the arity of P is $\mathcal{O}(\max(a + 2, 9) \cdot \log \gamma(|\Sigma|, |q|))$, where a is as above.*

We note that N is polynomially bounded in $|\Sigma|$ and $|q|$ by the polynomial γ that only depends on $\mathcal{C}$. The rest of this section explains the basic ideas of the proof of this result. A more detailed proof is given in [13].

High-level idea of the proof. We first describe the high level idea of the construction of the Datalog program P. It checks whether there is a chase sequence $S = t_1, \ldots, t_N$ with respect to D and Σ and a homomorphism h from q to (the set of atoms of) S. To this end, P consists of one large rule r_{goal} of polynomial size in N and some shorter rules that define auxiliary relations and will be explained below.

The aim of r_{goal} is to guess the chase sequence S *and* the homomorphism q at the same time. We recall that N does not depend on the size of D but only on $|\Sigma|$ and $|q|$ and thus r_{goal} can well be as long as the chase sequence and q together. One of the advantages of this approach is that we only have to deal with those null values that are actually relevant for answering the query. Thus, at most N null values need to be represented.

[3] Of course, if $\mathrm{dom}(D)$ already contains some natural numbers we can add a fresh copy of $\{0, 1, \ldots, N\}$ instead.

One might try to obtain r_{goal} by just taking one atom A_i for each tuple t_i of S and one atom for each atom of q and somehow test that they are consistent. However, it is not clear how consistency could possibly be checked in a purely conjunctive fashion.[4] There are two ways in which disjunctive reasoning is needed. First, it is not a priori clear on which previous tuples, tuple t_i will depend. Second, it is not a priori clear to which tuples of S the atoms of q can be mapped.

To overcome these challenges we use the following basic ideas.

(1) We represent the tuples of S (and the required tuples of D) in a symbolic fashion, utilizing the numerical domain.
(2) We let P compute auxiliary predicates that allow us to express disjunctive relationships between the tuples in S.

Example 1. We illustrate the proof idea with a very simple running example, shown in Figure 1.

(a) Σ :

σ_1: $R_1(X,Y) \rightarrow \exists Z\ R_4(X,Y,Z)$
σ_2: $R_2(Y,Z) \rightarrow \exists X\ R_4(X,Y,Z)$
σ_3: $R_3(X,Z) \rightarrow \exists Y\ R_4(X,Y,Z)$
σ_4: $R_4(X_1,Y_1,Z_1), R_4(X_2,Y_2,Z_2) \rightarrow R_5(X_1,Z_2)$

(b) $q : R_5(X,Y), R_3(Y,X)$

(c) D :

R_1	
a	b
c	d

R_2	
e	g

R_3	
g	a
g	h

Fig. 1. Simple example with (a) a set Σ of tgds, (b) a query q and (c) a database D.

A possible chase sequence in this example is shown in Figure 2(a). The mapping $X \mapsto a$ and $Y \mapsto g$, maps $R_5(X,Y)$ to t_5 and $R_3(Y,X)$ to t_6, thus satisfying q.

(a)

- t_1: $R_1(a,b)$
- t_2: $R_4(a,b,\perp_2)$
- t_3: $R_2(e,g)$
- t_4: $R_4(\perp_4,e,g)$
- t_5: $R_5(a,g)$
- t_6: $R_3(g,a)$

(b)

- t_1: $R_1(a,b,a)$
- t_2: $R_4(a,b,\perp_2)$
- t_3: $R_2(e,g,e)$
- t_4: $R_4(\perp_4,e,g)$
- t_5: $R_5(a,g,a)$
- t_6: $R_3(g,a,g)$

(c)

i	r_i	f_i	x_{i1}	x_{i2}	x_{i3}	s_i	c_{i1}	c_{i2}
1	1	0	a	b	a	0	0	0
2	4	1	a	b	2	1	1	1
3	2	0	e	g	e	0	0	0
4	4	1	4	e	g	2	3	3
5	5	1	a	g	a	4	2	4
6	3	0	g	a	g	0	0	0

Fig. 2. (a) Example chase sequence, (b) its extension and (c) its encoding. t_2 is obtained by applying σ_1 to t_1. Likewise t_4 and t_5 are obtained by applying σ_2 to t_3 and σ_4 to t_2 and t_4, respectively.

Notation and conventions. Let $\mathcal{C}$ be a class of tgds enjoying the PWP, let Σ be a set of tgds from $\mathcal{C}$, and let q be a BCQ. Let $R_1, \ldots R_m$ be the predicate symbols occurring in Σ or in q. We denote the number of tgds in Σ by ℓ.

[4] Furthermore, of course, there are no relations to which the atoms A_i could possible be matched.

Let $N := \gamma(|\Sigma|, |q|)$ where γ is as in Definition 1, thus N is polynomial in $|\Sigma|$ and $|q|$. By definition of N, if $(D, \Sigma) \models q$, then q can be witnessed by a chase sequence Γ of length $\leq N$. Our assumption that $\gamma(n_1, n_2) \geq \max(n_1, n_2)$, for every n_1, n_2, guarantees that N is larger than (i) the number of predicate symbols occurring in Σ, (ii) the cardinality $|q|$ of the query, and (iii) the number of rules in Σ.

For the sake of a simpler presentation, we assume that all relations in Σ have the same arity a and all rules use the same number k of tuples in their body. The latter can be easily achieved by repeating tuples, the former by filling up shorter tuples by repeating the first tuple entry. Furthermore, we only consider chase sequences of length N. Shorter sequences can be extended by adding tuples from D.

Example 2. Example 1 thus translates as illustrated in Figure 3. The (extended) chase sequence is shown in Figure 2 (b). The query q is now satisfied by the mapping $X \mapsto a$, $Y \mapsto g$, $U \mapsto g$, $V \mapsto a$, thus mapping $R_5(X, Y, X)$ to t_5 and $R_3(Y, X, Y)$ to t_6.

(a) Σ :

σ_1: $R_1(X, Y, X), R_1(X, Y, X) \rightarrow \exists Z\ R_4(X, Y, Z)$

σ_2: $R_2(Y, Z, Y), R_2(Y, Z, Y) \rightarrow \exists X\ R_4(X, Y, Z)$

σ_3: $R_3(X, Z, X), R_3(X, Z, X) \rightarrow \exists Y\ R_4(X, Y, Z)$

σ_4: $R_4(X_1, Y_1, Z_1), R_4(X_2, Y_2, Z_2) \rightarrow R_5(X_1, Z_2, X_1)$

(b) $q : R_5(X, Y, U), R_3(Y, X, V)$

(c) D :

R_1		
a	b	a
c	d	c

R_2		
e	g	e

R_3		
g	a	g
g	h	g

Fig. 3. Modified example with (a) a set Σ of tgds, (b) a query q and (c) a database D.

Proof idea (continued). On an abstract level, the atoms that make up the final rule r_{goal} of P can be divided into three groups serving three different purposes. That is, r_{goal} can be considered as a conjunction $r_{\text{tuples}} \wedge r_{\text{chase}} \wedge r_{\text{query}}$. Each group is "supported" by a sub-program of P that defines relations that are used in r_{goal}, and we refer to these three subprograms as P_{tuples}, P_{chase} and P_{query}, respectively.

- The purpose of r_{tuples} is basically to lay the ground for the other two. It consists of N atoms that allow to guess the symbolic encoding of a sequence $S = t_1, \ldots, t_N$.
- The atoms of r_{chase} are designed to verify that S is an actual chase sequence with respect to D.
- Finally, r_{query} checks that there is a homomorphism from q to S.

$P_{\textbf{tuples}}$ **and** $r_{\textbf{tuples}}$**.** The symbolic representation of the tuples t_i of the chase sequence S uses numerical values to encode null values, predicate symbols R_i (by i), tgds $\sigma_j \in \Sigma$ (by j) and the number of a tuple t_i in the sequence (that is: i).

In particular, the symbolic encoding uses the following numerical parameters.[5]

- r_i to indicate the relation R_{r_i} to which the tuple belongs;
- f_i to indicate whether t_i is from D ($f_i = 0$) or yielded by the chase ($f_i = 1$);

[5] We use the names of the parameters as variable names in r_{goal} as well.

- Furthermore, $x_{i1}, \ldots, x_{ia}$ represent the attribute values of t_i as follows. If the j-th attribute of t_i is a value from $\text{dom}(D)$ then x_{ij} is intended to be that value, otherwise it is a null represented by a numeric value.

Since each rule of Σ has at most one existential quantifier in its head, at each chase step, at most one new null value can be introduced. Thus, we can unambiguously represent the null value (possibly) introduced in the j-th step of the chase by the number j.

The remaining parameters s_i and $c_{i1}, \ldots, c_{ik}$ are used to encode information about the tgd and the tuples (atoms) in S that are used to generate the current tuple. More precisely, s_i is intended to be the number of the applied tgd σ_{s_i} and $c_{i1}, \ldots, c_{ik}$ are the tuple numbers of the k tuples that are used to yield t_i. In the example, e.g., t_5 is obtained by applying σ_4 to t_2 and t_4. The encoding of our running example can be found in Figure 2 (c).

We use a new relational symbol T of arity $a + k + 4$ not present in the schema of D for the representation of the tuples from S. Thus, r_{tuples} is just:

$$T(1, r_1, f_1, x_{11}, \ldots, x_{1a}, s_1, c_{11}, \ldots, c_{1k}), \ldots, \\ T(N, r_N, f_N, x_{N1}, \ldots, x_{Na}, s_N, c_{N1}, \ldots, c_{Nk}).$$

The sub-program P_{tuples} is intended to “fill” T with suitable tuples. Basically, T contains all encodings of tuples in D (with $f_i = 0$) and all syntactically meaningful tuples corresponding to possible chase steps (with $f_i = 1$).

P_{chase} and r_{chase}. The following kinds of conditions have to be checked to ensure that the tuples “guessed” by r_{tuples} constitute a chase sequence.

(1) For every i, the relation R_{r_i} of a tuple t_i has to match the head of its rule σ_{s_i}.
  - In the example, e.g., r_4 has to be 4 as the head of σ_2 is an R_4-atom.

(2) Likewise, for each i and j the relation number of tuple $t_{c_{ij}}$ has to be the relation number of the j-th atom of σ_{s_i}.
  - In the example, e.g., r_2 must be 4, as $c_{5,1} = 2$ and the first atom of $\sigma_{s_5} = \sigma_4$ is an R_4-atom.

(3) If the head of σ_{s_i} contains an existentially quantified variable, the new null value is represented by the numerical value i.
  - This is illustrated by t_4 in the example: the first position of the head of rule 2 has an existentially quantified variable and thus $x_{4,1} = 4$.

(4) If a variable occurs at two different positions in σ_{s_i} then the corresponding positions in the tuples used to produce t_i carry the same value.

(5) If a variable in the body of σ_{s_i} also occurs in the head of σ_{s_i} then the values of the corresponding positions in the body tuple and in t_i are equal.
  - Z_2 occurs in position 3 of the second atom of the body of σ_4 and in position 2 of its head. Therefore, $x_{4,3}$ and $x_{5,2}$ have to coincide (where the 4 is determined by $c_{5,2}$.

It turns out that all these tests can be done by r_{chase}, given some relations that are precomputed by P_{chase}. More precisely, we let P_{chase} specify a 4-ary predicate $\text{IfThen}(X_1, X_2, U_1, U_2)$ that is intended to contain all tuples fulfilling the condition: if $X_1 = X_2$ then $U_1 = U_2$. Similar predicates are defined for conditions with two and three conjuncts in the IF-part. Their definition by Datalog rules is straightforward.

P_{query} **and** r_{query}**.** Finally, we explain how it can be checked that there is a homomorphism from q to S. We explain the issue through the little example query $R_3(x, y) \wedge R_4(y, z)$. To evaluate this query, r_{query} makes use of two additional variables q_1 and q_2, one for each atom of q. The intention is that these variables bind to the numbers of the tuples that the atoms are mapped to. We have to make sure two kinds of conditions. First, the tuples need to have the right relation symbol and second, they have to obey value equalities induced by the variables of q that occur more than once.

The first kind of conditions is checked by adding atoms IfThen$(q_1, i, r_i, 3)$ and IfThen$(q_2, i, r_i, 4)$ to r_{query}, for every $i \leq N$. The second condition is checked similarly. As we do not need any further auxiliary predicates, P_{query} is empty.

This completes the description of P. Note that P is nonrecursive, and has polynomial size in the size of q and Σ. Furthermore, the arity of P is as required. This proves part (a) of Theorem 1.

In order to prove part (b), we must get rid of the numeric domain (except for 0 and 1). This is actually very easy. We just replace each numeric value by a logarithmic number of bits (coded by our 0 and 1 domain elements), and extend the predicate arities accordingly. As a matter of fact, this requires an increase of arity by a factor of $\log N = \mathcal{O}(\log |q|)$. This concludes our explanation of the proof ideas underlying Theorem 1.

Remark 1. Note that the evaluation complexity of the Datalog program obtained for case (b) is not significantly higher than the evaluation complexity of the program P constructed for case (a). For example, in the most relevant case of bounded arities, both programs can be evaluated in NPTIME combined complexity over a database D. In fact, it is well-known that the combined complexity of a Datalog program of bounded arity is in NPTIME (see [10]). But it is easy to see that if we expand the signature of such a program (and of the underlying database) by a logarithmic number of Boolean-valued argument positions (attributes), nothing changes, because the possible values for such vectorized arguments are still of polynomial size. It is just a matter of coding. In a similar way, the data complexity in both cases (a) and (b) is the same (PTIME).

Remark 2. It is easy to generalize this result to the setting where q is actually a union of conjunctive queries (UCQ).

4 Further Results Derived From the Main Theorem

We wish to mention some interesting consequences of Theorem 1 that follow easily from the above result after combining it with various other known results.

4.1 Linear TGDs

A linear tgd [5] is one that has a single atom in its rule body. The class of linear tgds is a fundamental one in the Datalog$^{\pm}$ family. This class contains the class of *inclusion dependencies*. It was already shown in [14] for inclusion dependencies that classes of linear tgds of bounded (predicate) arities enjoy the PWP. That proof carries over to linear tgds.

By Theorem 1, we then conclude:

Theorem 2. *Conjunctive queries under linear tgds of bounded arity are polynomially rewritable as nonrecursive Datalog programs in the same fashion as for Theorem 1. So are sets of inclusion dependencies of bounded arity.*

4.2 DL-Lite

A pioneering and highly significant contribution towards tractable ontological reasoning was the introduction of the *DL-Lite* family of description logics (DLs) by Calvanese et al. [9, 20]. *DL-Lite* was further studied and developed in [1].

A DL-lite theory (or TBox) $\Sigma = (\Sigma^-, \Sigma^+)$ consists of a set of negative constraints Σ^- such as key and disjointness constraints, and of a set Σ^+ of positive constraints that resemble tgds. As shown in [9], the negative constraints Σ^- can be compiled into a polymomially sized first-order formula (actually a union of conjunctive queries) of the same arity as Σ^- such that for each database and BCQ q, $(D, \Sigma) \models q$ iff $D \not\models \Sigma^-$ and $(D, \Sigma^+) \models q$. In (the full version of) [5] it was shown that for the main DL-Lite variants defined in [9], each Σ^+ can be immediately translated into an equivalent set of linear tgds of arity 2. By virtue of this, and the above we obtain the following theorem.

Theorem 3. *Let q be a CQ and let $\Sigma = (\Sigma^-, \Sigma^+)$ be a DL-Lite theory expressed in one of the following DL-Lite variants: DL-Lite$_{\mathcal{F},\sqcap}$, DL-Lite$_{\mathcal{R},\sqcap}$, DL-Lite$^+_{\mathcal{A},\sqcap}$, DLR-Lite$_{\mathcal{F},\sqcap}$, DLR-Lite$_{\mathcal{R},\sqcap}$, or DLR-Lite$^+_{\mathcal{A},\sqcap}$. Then Σ^+ can be rewritten into a nonrecursive Datalog program P such that for each database D, $(D, \Sigma^+) \models q$ iff $D \models P$. Regarding the arities of P, the same bounds as in Theorem 1 hold.*

4.3 Sticky and Sticky Join TGDs

Sticky tgds [6] and sticky-join tgds [6] are special classes of tgds that generalize linear tgds but allow for a limited form of join (including as special case the cartesian product). They allow one to express natural ontological relationships not expressible in DLs such as OWL. For space reasons, we do not define these classes here, and refer the reader to [8]. By results of [8], which will also be discussed in detail in a future extended version [13] of the present paper, both classes enjoy the Polynomial Witness Property. By Theorem 1, we thus obtain the following result:

Theorem 4. *Conjunctive queries under sticky tgds and sticky-join tgds over a fixed signature $\mathcal{R}$ are rewritable into polynomially sized nonrecursive Datalog programs of arity bounded as in Theorem 1.*

5 Related Work on Query Rewriting

Several techniques for query-rewriting have been developed. An early algorithm, introduced in [9] and implemented in the QuOnto system[6], reformulates the given query into a union of CQs (UCQs) by means of a backward-chaining resolution procedure.

[6] http://www.dis.uniroma1.it/ quonto/

The size of the computed rewriting increases exponentially w.r.t. the number of atoms in the given query. This is mainly due to the fact that unifications are derived in a "blind" way from every unifiable pair of atoms, even if the generated rule is superfluous. An alternative resolution-based rewriting technique was proposed by Peréz-Urbina et al. [19], implemented in the Requiem system[7], that produces a UCQs as a rewriting which is, in general, smaller (but still exponential in the number of atoms of the query) than the one computed by QuOnto. This is achieved by avoiding the useless unifications, and thus the redundant rules obtained due to these unifications. This algorithm works also for more expressive non-first-order rewritable DLs. In this case, the computed rewriting is a (recursive) Datalog query. Following a more general approach, Calì et al. [3] proposed a backward-chaining rewriting algorithm for the first-order rewritable Datalog$^{\pm}$ languages mentioned above. However, this algorithm is inspired by the original QuOnto algorithm, and inherits all its drawbacks. In [12], a rewriting technique for linear Datalog$^{\pm}$ into unions of conjunctive queries is proposed. This algorithm is an improved version of the one already presented in [3]. However, the size of the rewriting is still exponential in the number of query atoms.

Of more interest to the present work are rewritings into nonrecursive Datalog. In [15, 16] a polynomial-size rewriting into nonrecursive Datalog is given for the description logics DL-Lite$^{\mathcal{F}}_{horn}$ and DL-Lite$_{horn}$. For DL-Lite$^{\mathcal{N}}_{horn}$, a DL with counting, a polynomial rewriting involving aggregate functions is proposed. It is, moreover, shown in (the full version of) [15] that for the description logic DL-Lite$_{\mathcal{F}}$ a polynomial-size pure first-order query rewriting is possible. Note that neither of these logics allows for role inclusion, while our approach covers description logics with role inclusion axioms. Other results in [15, 16] are about *combined rewritings* where both the query and the database D have to be rewritten. A recent very interesting paper discussing polynomial size rewritings is [22]. Among other results, [22] provides complexity-theoretic arguments indicating that without the use of special constants (e.g, 0 and 1, or the numerical domain), a polynomial rewriting such as ours may not be possible. Rosati et al. [21] recently proposed a very sophisticated rewriting technique into nonrecursive Datalog, implemented in the Presto system. This algorithm produces a non-recursive Datalog program as a rewriting, instead of a UCQs. This allows the "hiding" of the exponential blow-up inside the rules instead of generating explicitly the disjunctive normal form. The size of the final rewriting is, however, exponential in the number of non-eliminable existential join variables of the given query; such variables are a subset of the join variables of the query, and are typically less than the number of atoms in the query. Thus, the size of the rewriting is exponential in the query size in the worst case. Relevant further optimizations of this method are given in [18].

Acknowledgment G. Gottlob's Work was funded by the EPSRC Grant EP/H051511/1 ExODA: Integrating Description Logics and Database Technologies for Expressive Ontology-Based Data Access. We thank the anonymous referees, as well as Roman Kontchakov, Carsten Lutz, and Michael Zakharyaschev for useful comments on an earlier version of this paper.

[7] http://www.comlab.ox.ac.uk/projects/requiem/home.html

References

1. Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev, *The dl-lite family and relations*, J. Artif. Intell. Res. (JAIR) **36** (2009), 1–69.
2. Catriel Beeri and Moshe Y. Vardi, *The implication problem for data dependencies*, Proc. of ICALP, 1981, pp. 73–85.
3. A. Calì, G. Gottlob, and A. Pieris, *Query rewriting under non-guarded rules*, Proc. AMW, 2010.
4. Andrea Calì, Georg Gottlob, and Michael Kifer, *Taming the infinite chase: Query answering under expressive relational constraints*, Proc. of KR, 2008, pp. 70–80.
5. Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz, *A general datalog-based framework for tractable query answering over ontologies*, Proc. of PODS, 2009, pp. 77–86.
6. Andrea Calì, Georg Gottlob, and Andreas Pieris, *Advanced processing for ontological queries*, PVLDB **3** (2010), no. 1, 554–565.
7. ———, *Query answering under non-guarded rules in datalog+/-*, Proc. of RR, 2010, pp. 175–190.
8. ———, *Towards more expressive ontology languages: The query answering problem*, Tech. report, University of Oxford, Department of Computer Science, 2011, Submitted for publication - available from the authors.
9. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati, *Tractable reasoning and efficient query answering in description logics: The DL-lite family*, J. Autom. Reasoning **39** (2007), no. 3, 385–429.
10. Evgeny Dantsin, Thomas Eiter, Gottlob Georg, and Andrei Voronkov, *Complexity and expressive power of logic programming*, ACM Comput. Surv. **33** (2001), no. 3, 374–425.
11. Alin Deutsch, Alan Nash, and Jeff B. Remmel, *The chase revisisted*, Proc. of PODS, 2008, pp. 149–158.
12. Georg Gottlob, Giorgio Orsi, and Andreas Pieris, *Ontological queries: Rewriting and optimization*, Proc. of ICDE, 2011.
13. Georg Gottlob and Thomas Schwentick, *Rewriting ontological queries into small nonrecursive datalog programs*, arXiv Computing Research Repository (CoRR) **arXiv:1106.3767** (2011), extended version, available at `http://arxiv.org/abs/1106.3767`.
14. David S. Johnson and Anthony C. Klug, *Testing containment of conjunctive queries under functional and inclusion dependencies*, J. Comput. Syst. Sci. **28** (1984), no. 1, 167–189.
15. Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev, *The combined approach to query answering in dl-lite*, KR (Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski, eds.), AAAI Press, 2010.
16. ———, *The combined approach to ontology-based data access*, IJCAI, 2011.
17. David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv, *Testing implications of data dependencies.*, ACM Trans. Database Syst. **4** (1979), no. 4, 455–469.
18. Giorgio Orsi and Andreas Pieris, *Optimizing query answering under ontological constraints*, PVLDB, 2011, to appear.
19. H. Pérez-Urbina, B. Motik, and I. Horrocks, *Tractable query answering and rewriting under description logic constraints*, Journal of Applied Logic **8** (2009), no. 2, 151–232.
20. Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati, *Linking data to ontologies*, J. Data Semantics **10** (2008), 133–173.
21. R. Rosati and A. Almatelli, *Improving query answering over DL-Lite ontologies*, Proc. KR, 2010.
22. R.Kontchakov S. Kikot, Carsten Lutz, and M. Zakharyaschev, *On (In)Tractability of OBDA with OWL2QL*, Proc. DL, 2011.