Optimizing Query Answering under Ontological Constraints

Giorgio Orsi

Institute for the Future of Computing
University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3JP
United Kingdom

Andreas Pieris

Department of Computer Science
University of Oxford
Wolfson Building, Parks Road
Oxford OX1 3JP
United Kingdom

{giorgio.orsi, andreas.pieris}@cs.ox.ac.uk

ABSTRACT

Ontological queries are evaluated against a database combined with ontological constraints. Answering such queries is a challenging new problem for database research. For many ontological modelling languages, query answering can be solved via query rewriting: given a conjunctive query and an ontology, the query can be transformed into a first-order query, called the *perfect rewriting*, that takes into account the semantic consequences of the ontology. Then, for every extensional database D, the answer to the query is obtained by evaluating the rewritten query against D. In this paper we present a new algorithm that computes the perfect rewriting of a conjunctive query w.r.t. a linear Datalog$^{\pm}$ ontology. Also, we provide an experimental comparison of our algorithm with existing rewriting techniques.

1. INTRODUCTION

Ontology-Based Data Access. In the recent years, initiatives such as the Linked Open Data[1], and the adoption of semantic data representation formats such as RDF(S) and OWL, pushed the academy and the industry to study Semantic Web data management techniques (see, e.g., [11]) to support efficient storage and querying of large-scale repositories of semantic data. In order to meet the efficiency requirements, current solutions often resort to a relational DBMS. Interestingly, such a trend inspired a related line of research in the database community, dealing with the enhancement of traditional database systems with advanced reasoning and query processing mechanisms.

In ontological database management systems, an extensional relational database D (also called 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 answered against the logical theory $D \cup \Sigma$, and not just against D. Formally, if $Q : q(\mathbf{X}) \leftarrow \phi(\mathbf{X}, \mathbf{Y})$ is a conjunctive query (CQ) with output variables $\mathbf{X}$, then its answer in the ontological database consists of all the tuples $\mathbf{t}$ of constants such that $D \cup \Sigma \models \exists \mathbf{u}\, \phi(\mathbf{t}, \mathbf{u})$, or, equivalently, $\mathbf{t}$ belongs to the answer of Q over I, for each instance I that contains D and satisfies Σ.

Answering a CQ against $D \cup \Sigma$ is equivalent to answering the same query against the chase-expansion of D w.r.t. Σ (see, e.g., [12]). This expansion can be constructed by applying the well-known *chase* procedure [18, 16], which is presented in Section 2. Roughly speaking, the chase adds new tuples to D (possibly with labelled nulls that represent unknown values) until the final instance, written $chase(D, \Sigma)$, satisfies Σ. However, the chase-expansion may be infinite, and hence not explicitly computable. As shown by the following example, taken from [6], this holds even for databases with just a single fact, and very simple ontologies.

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

$$person(X) \quad \rightarrow \quad \exists Y\, father(Y, X), person(Y)$$

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

Procedures for effectively answering queries in case of non-terminating chase-expansions were first proposed in the database context by Johnson and Klug [16] for the special case where the ontological theory contains *inclusion dependencies* only. However, inclusions dependencies are not powerful enough to represent ontological constraints. Thus, a recent research direction is to find more expressive formalisms, under which query answering is still decidable. Moreover, it is desirable that query answering is tractable in *data complexity* (i.e., when the query and the ontology are fixed), and possibly executable by relational query engines; this is needed in order to be able to deal with very large databases.

A significant contribution in this direction has been the introduction of the *DL-Lite* family of description logics

[1] http://linkeddata.org/

Proceedings of the VLDB Endowment, Vol. 4, No. 11

(DLs) by Calvanese et al. [9, 20]. The DL-Lite family was recently embedded into an expressive framework called *Datalog*$^{\pm}$ (see, e.g., [5, 6]), whose languages extend the well-known Datalog language [1] by allowing existential quantifiers in rule-heads, thus using *tuple-generating dependencies (TGDs)* instead of plain Datalog rules; this feature is also known as *value invention* (see, e.g., [3]). Highly tractable languages in this framework are *linear Datalog*$^{\pm}$ [5], which is a sub-formalism of a more expressive (but still tractable) language called *guarded Datalog*$^{\pm}$ [5], *sticky Datalog*$^{\pm}$ [6], and *sticky-join Datalog*$^{\pm}$ [7], which captures both linear and sticky Datalog$^{\pm}$. In a nutshell, 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). Sticky(-join) Datalog$^{\pm}$ impose a restriction on multiple occurrences of variable in rule-bodies.

First-Order Rewritability. CQ answering under the above Datalog$^{\pm}$ languages (apart from guarded Datalog$^{\pm}$), as well as under DL-Lite, has the advantage of being *first-order rewritable*. This property implies that query answering can be performed as follows: compute the so-called *perfect rewriting* of the given query w.r.t. to the ontological theory, and then evaluate it over the given database; note that the perfect rewriting is a first-order query. More precisely, a pair $\langle Q, \Sigma \rangle$, where Q is a CQ of the form $q(\mathbf{X}) \leftarrow \phi(\mathbf{X}, \mathbf{Y})$, and Σ is an ontology, can be rewritten as a first-order query Q_Σ, defined as $q(\mathbf{X}) \leftarrow \phi_\Sigma(\mathbf{X}, \mathbf{Y})$, such that for every possible answer tuple $\mathbf{t}$ (of constants) it holds that $D \cup \Sigma \models \exists \mathbf{u}\, \phi(\mathbf{t}, \mathbf{u})$ iff $D \models \exists \mathbf{u}\, \phi_\Sigma(\mathbf{t}, \mathbf{u})$, for every database D.

Example 2. Consider the ontological theory Σ given in the Example 1, and let Q be the CQ $q(B) \leftarrow \mathit{father}(A, B)$ asking for persons who have a father. Intuitively, due to the rule in Σ, not only do we have to query *father*, but we also need to query *person*, since all the persons necessarily have a father. The perfect rewriting Q_Σ will thus be the logical *union* of the given query and of the query $q(B) \leftarrow \mathit{person}(B)$. ■

It is well-known that each first-order query can be equivalently written in SQL. Thus, a CQ based on an ontology can be rewritten as an SQL query over the original database (the translation of the perfect rewriting of Example 2 into SQL is given in Appendix A), which implies that it can be passed to a relational DBMS, and executed efficiently by exploiting all the DBMS's optimizations.

Existing Rewriting Techniques. Several techniques for computing the perfect rewriting of a query w.r.t. an ontology that falls in one of the first-order rewritable languages mentioned above have been proposed. Let us first discuss existing systems and algorithms for the DL-Lite family.

An early algorithm, introduced in [9] and implemented in the QuOnto system[2], reformulates the given query into a union of CQs (UCQs) by means of a backward-chaining resolution procedure. 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[3], 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 them. Note that this algorithm works also for more expressive non-first-order rewritable DLs. In this case, the computed rewriting is a (recursive) Datalog query.

Rosati et al. [21] recently proposed a very sophisticated rewriting technique, 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 explicitly generating the disjunctive normal form. The size of the final rewriting is exponential only 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.

Following a more general approach, Calì et al. [8] 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.

Gottlob et al. [15] recently proposed a rewriting technique for linear Datalog$^{\pm}$. In fact, this algorithm is an improved version of the one presented in [8], where the useless unifications are avoided, and also the atoms in the body of a rule that are logically implied (w.r.t. the ontological theory) by other atoms in the same rule are eliminated. This elimination implies the avoidance of the construction of redundant rules during the rewriting process. However, the size of the rewriting is still exponential in the number of atoms of the query.

Summary of Contributions. The goal of this paper is to improve the current rewriting techniques for linear Datalog$^{\pm}$ and DL-Lite. In particular, we present a new algorithm that computes the perfect rewriting of a conjunctive query w.r.t. a linear Datalog$^{\pm}$ ontology. The key ideas underlying our algorithm are as follows.

Differently from the existing techniques, our algorithm computes a *bounded* Datalog query. In particular, we exploit the notion of *predicate boundedness* in Datalog programs [10]. Intuitively speaking, given a (possibly recursive) Datalog program Π, if a predicate q is bounded in Π, then it is possible to construct a non-recursive Datalog program Π_q such that, for each atom $\underline{a}$ with predicate q, $D \cup \Pi \models \underline{a}$ iff $D \cup \Pi_q \models \underline{a}$. So, given a CQ $Q : q(\mathbf{X}) \leftarrow \phi(\mathbf{X}, \mathbf{Y})$ and a linear Datalog$^{\pm}$ ontology Σ, the algorithm constructs a Datalog query Q_Σ in which the predicate q, i.e., the head-predicate of the given query, is bounded.

Our algorithm uses the fact that linear Datalog$^{\pm}$ enjoys the *bounded-derivation depth property (BDDP)* [5]. This implies that, for every database D, instead of evaluating Q over $\mathit{chase}(D, \Sigma)$, it suffices to evaluate it over a finite part of the chase whose size does not depend on D.

We also show that there exists a strong connection between predicate boundedness and BDDP exemplified by a semantic class of Datalog programs, called *output predicate bounded*, where all the predicates that do not occur in the body of a rule are bounded. Roughly, all the tuples of these predicates are constructed after a finite number of steps.

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

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

The expansion of the query (based on resolution) is profoundly optimized by eliminating body-atoms that are logically implied (w.r.t. the given ontology) by other atoms in the same rule. This elimination implies the avoidance of the construction of redundant rules. The elimination technique that we apply here extends the one proposed in [15].

Due to the above advancements, the rewriting produced by our algorithm is not exponential in the number of atoms of the given query, but is only exponential in the number of *non-eliminable* atoms, i.e., the atoms that cannot be eliminated by our query elimination technique. Our experiments show that this number is, in general, less than the number of non-eliminable existential join variables defined in [21].

Roadmap. After some technical definitions in Section 2, our rewriting algorithm (without query elimination) is presented in Section 3. In Section 4 the query elimination technique is defined. An implementation and experimental evaluation of the proposed technique is discussed in Section 5. We conclude with a brief outlook on further research.

2. PRELIMINARIES

In this section we recall some basics on databases, tuple-generating dependencies, queries, and the chase procedure.

General. We define the following pairwise disjoint (possibly infinite) sets of symbols: *(i)* a set Γ of *constants* (constitute the "normal" domain of a database), *(ii)* a set Γ_N of *labelled nulls* (used as place-holders for unknown values, and thus can be also seen as variables), and *(iii)* a set Γ_V of *variables* (used in queries and constraints). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$, where $k \geqslant 0$. Let $[n]$ be the set $\{1, \ldots, n\}$, for any integer $n \geqslant 1$.

A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* (or *predicates*), each with its associated arity. 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 is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. Conjunctions of atoms are often identified with the sets of their atoms. A *relational instance* (or simply *instance*) I for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(\mathbf{t})$, where r is an n-ary predicate of $\mathcal{R}$, and $\mathbf{t} \in (\Gamma \cup \Gamma_N)^n$. We denote as $r(I)$ the set $\{\mathbf{t} \mid r(\mathbf{t}) \in I\}$. A *database* is a finite relational instance.

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 $h : \Gamma \cup \Gamma_N \cup \Gamma_V \to \Gamma \cup \Gamma_N \cup \Gamma_V$ such that: *(i)* if $t \in \Gamma$, then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n)$ is in A_1, then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2. The notion of homomorphism naturally extends to conjunctions of atoms.

Tuple-Generating Dependencies. Given a schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ over $\mathcal{R}$ is a first-order formula $\forall\mathbf{X}\forall\mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \to \exists\mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called 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 an instance I for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq I$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq I$.

Datalog. A *Datalog rule* ρ is an expression of the form $\underline{a}_0 \leftarrow \underline{a}_1, \ldots, \underline{a}_n$, for $n \geqslant 0$, where $\underline{a}_i$ is an atom, and every variable occurring in $\underline{a}_0$ must appear in at least one of the atoms $\underline{a}_1, \ldots, \underline{a}_n$. The atom $\underline{a}_0$ is called the *head* of ρ, denoted as $head(\rho)$, while the set of atoms $\{\underline{a}_1, \ldots, \underline{a}_n\}$ is called the *body* of ρ, denoted as $body(\rho)$. A *Datalog program* Π over a schema $\mathcal{R}$ is a set of Datalog rules such that, for each $\rho \in \Pi$, the predicate of $head(\rho)$ does not occur in $\mathcal{R}$. The *extensional database (EDB)* predicates are those that do not occur in the head of any rule of Π; all the other predicates are called *intensional database (IDB)* predicates.

A *model* of Π is an instance over $\mathcal{R}$ that satisfies the set of TGDs obtained by considering Π as a set of universally quantified implications. The semantics of Π w.r.t. a database D for $\mathcal{R}$, denoted as $\Pi(D)$, is the minimum model of Π containing D (which is unique and always exists). $\Pi(D)$ can be computed by a least fixpoint iteration. Let T_Π be the union of D with the set of all atoms $\underline{a}$ such that there exists a rule $\rho \in \Pi$ and a homomorphism h such that $h(body(\rho)) \subseteq D$ and $h(head(\rho)) = \underline{a}$. We write $T^i_\Pi(D)$ for the result of iterating this operation i times. Formally, $T^0_\Pi(D) = D$ and $T^{i+1}_\Pi(D) = T_\Pi(T^i_\Pi(D))$; let $T^\omega_\Pi(D) = \bigcup_{i \geqslant 0} T^i_\Pi(D)$. For every database D for $\mathcal{R}$, there exists an integer k such that $T^k_\Pi(D) = T^\omega_\Pi(D)$, i.e., $T^k_\Pi(D)$ is a fixpoint of T_Π; in general, k depends on D. It is well-known that $\Pi(D) = T^\omega_\Pi(D)$.

A Datalog program Π over a schema $\mathcal{R}$ is *bounded* if there exists a constant k such that, for every database D for $\mathcal{R}$, $T^k_\Pi(D) = T^\omega_\Pi(D)$. A refined notion of boundedness, which is more appropriate for our work, is *predicate boundedness* [10]. If q is an IDB predicate of Π, then $T^i_{\Pi,q}(D)$ (resp., $T^\omega_{\Pi,q}(D)$) is the set of atoms in $T^i_\Pi(D)$ (resp., $T^\omega_\Pi(D)$) with predicate q. The predicate q *is bounded in* Π if there exists a constant k such that, for every database D for $\mathcal{R}$, $T^k_{\Pi,q}(D) = T^\omega_{\Pi,q}(D)$. Intuitively, if an IDB predicate q is bounded in a (recursive) Datalog program Π, then it is possible to obtain all the atoms of $\Pi(D)$ with predicate q, for every database D, using a non-recursive program Π_q (see Appendix B). Note that the problem of predicate boundedness is undecidable [14].

Queries. An n-ary *Datalog query* Q over a schema $\mathcal{R}$ is a pair $\langle q, \Pi \rangle$, where Π is a Datalog program over $\mathcal{R}$, and q is an n-ary predicate which occurs in the head of at least one rule of Π. Q is a *bounded Datalog query* if the predicate q is bounded in Π. Q is a *union of conjunctive queries (UCQs)* if q is the only IDB predicate in Π, and for each rule $\rho \in \Pi$, q does not occur in $body(\rho)$. Finally, Q is a *conjunctive query (CQ)* if it is a UCQs, and Π contains exactly one rule. A *Boolean Datalog query* is a Datalog query of arity zero. The *answer* to an n-ary Datalog query $Q = \langle q, \Pi \rangle$ over a database D is the set $\{\mathbf{t} \mid q(\mathbf{t}) \in \Pi(D)\}$, denoted as $Q(D)$. A Boolean Datalog query Q has only the empty tuple as possible answer; Q has *positive* answer over D, denoted as $D \models Q$, iff $\langle\rangle \in Q(D)$, or, equivalently, $Q(D) \neq \varnothing$.

CQ 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 instances I such that $I \models D \cup \Sigma$, which means that $I \supseteq D$ and I satisfies Σ. The *answer* to a CQ Q w.r.t. D and Σ, denoted as $ans(Q, D, \Sigma)$, is the set $\{\mathbf{t} \mid \mathbf{t} \in Q(I), \text{ for each } I \in mods(D, \Sigma)\}$. The *answer* to a Boolean CQ (BCQ) Q w.r.t. D and Σ is *positive*,

written $D \cup \Sigma \models Q$, iff $ans(Q, D, \Sigma) \neq \varnothing$. Note that CQ answering under general TGDs is undecidable [2], even when the schema and the set of TGDs are fixed [4].

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

TGD Chase Procedure. The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [18], and later for checking query containment [16]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted instance 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*.

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 $z_i \in \Gamma_N$ is a "fresh" labelled 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 w.r.t. 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 w.r.t. Σ, denoted as $chase(D, \Sigma)$; see Appendix B for the formal definition. The (possibly infinite) chase of D w.r.t. Σ is a *universal model* of D w.r.t. Σ, i.e., for each instance $I \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$ to I [13, 12]. This implies that for a BCQ Q, $D \cup \Sigma \models Q$ iff $chase(D, \Sigma) \models Q$.

Linear TGDs. A TGD is *linear* iff its body contains a single atom [5]; linear TGDs generalizes the well-known *inclusion dependencies*. The key property of this class, that we are going to exploit in our rewriting algorithm, is the so-called *bounded-derivation depth property (BDDP)* [5]. Roughly speaking, $chase(D, \Sigma)$ can be decomposed in levels, where D has level 0, and an atom has level $k + 1$ if it is obtained, during the chase, due to atoms with maximum level k. We refer to the part of the chase up to level k as $chase^k(D, \Sigma)$; for the formal definitions see Appendix B. A class $\mathcal{C}$ of TGDs enjoys the BDDP iff for every BCQ Q over a schema $\mathcal{R}$, for every database D for $\mathcal{R}$, and for every set $\Sigma \in \mathcal{C}$ over $\mathcal{R}$, if $D \cup \Sigma \models Q$, then $chase^k(D, \Sigma) \models Q$, where k depends only on Q and $\mathcal{R}$, but not on the database D.

As established in [5], the class of linear TGDs enjoys the BDDP. In particular, given a BCQ $Q = \langle q, \rho \rangle$ over a schema $\mathcal{R}$, a database D for $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$, if $D \cup \Sigma \models Q$, then $chase^k(D, \Sigma) \models Q$, for $k = n \cdot (2w)^w \cdot 2^{|\mathcal{R}| \cdot (2w)^w}$, where n is the number of atoms in the body of ρ, and w is the maximum arity over all predicates of $\mathcal{R}$; clearly, k is constant w.r.t. D. In the rest of the paper, we will refer to the value k by $\mathsf{b}(Q, \mathcal{R})$.

3. QUERY REWRITING

In this section we present the algorithm that computes the perfect rewriting of a query w.r.t. a set of linear TGDs. More precisely, given a BCQ Q over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$, our goal is to compute a bounded Datalog query Q_Σ over $\mathcal{R}$ such that, for *every* database D for $\mathcal{R}$, $D \cup \Sigma \models Q$ iff $D \models Q_\Sigma$. Before delving into the details, let us give an informal description of the algorithm.

3.1 Informal Description

Our algorithm accepts a BCQ Q over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$, and proceeds in four steps.

Skolemization. First, the existential quantifiers in the TGDs of Σ are eliminated by reducing each TGD into *Skolem normal form*. Then, each rule $\underline{b} \rightarrow \underline{h}_1, \ldots, \underline{h}_n$ can be replaced with the set of rules $\{\underline{b} \rightarrow \underline{h}_i\}_{i \in [n]}$, since the joins among existentially quantified variables are preserved by the introduced Skolem terms. Finally, by exploiting the notion of predicate graph, we identify rules that are not needed for answering Q, and thus are eliminated. During this step, we reduce Σ into an equisatisfiable set Σ_f of rules with singleton heads. This step is necessary since the subsequent steps assume singleton rule-heads. As already noticed in [19], by transforming Σ into Σ_f via Skolemization, we avoid the introduction of an auxiliary predicate for each TGD of Σ (as in [9] and [15]), and thus we avoid the generation of additional rules due to these auxiliary predicates.

Rule Saturation. This step computes the so-called *saturated set* of Σ_f, written $[\Sigma_f]$, by applying the well-known *resolution* inference rule. A rule of $[\Sigma_f]$, obtained by applying k times the resolution rule, "mimics" a derivation of the chase under Σ_f (rules with functional terms in their head can be seen as TGDs), which involves $k + 1$ applications of the TGD chase rule. Notice that $[\Sigma_f]$ is, in general, infinite. However, since linear TGDs enjoy the BDDP (see Section 2), it suffices to "mimic" the chase of level up to $\mathsf{b}(Q, \mathcal{R})$. Thus, we just need to compute only a finite part of $[\Sigma_f]$.

Query Saturation. During this step the so-called *saturated query* of Q, denoted as $[Q, \Sigma_f]$, is computed. This is done by applying again the resolution inference rule by considering only the non-function-free rules of $[\Sigma_f]$. Roughly, given a database D, if one of the atoms due to which $chase(D, \Sigma_f)$ entails Q was obtained by a chase derivation that involves TGDs with functional terms, then we "bypass" this derivation. In order to guarantee the entailment of Q by $chase(D, \Sigma_f)$, we exploit the atom of D due to which the first TGD in the "bypassed" derivation was triggered. Notice again that $[Q, \Sigma_f]$ is, in general, infinite. Nevertheless, since linear TGDs have the BDDP, as in the case of $[\Sigma_f]$, we just need to compute only a finite part of $[Q, \Sigma_f]$.

Finalization. In this step the rewritten query Q_Σ is obtained by discarding the non-function-free rules of $[\Sigma_f] \cup [Q, \Sigma_f]$. In addition, some auxiliary predicates and rules are added in order to ensure that Q_Σ is a Datalog query over $\mathcal{R}$, i.e., the predicates of $\mathcal{R}$ are EDB predicates.

3.2 The Rewriting Algorithm

Let us now formalize the above four steps. Recall that the algorithm accepts as input a BCQ $Q = \langle q, \rho \rangle$ over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$.

Skolemization. Let Σ_f be the set of rules obtained from Σ by replacing each TGD σ of the form

$$r_0(\mathbf{X}, \mathbf{Y}) \rightarrow \exists Z_1 \ldots \exists Z_m\, r_1(\mathbf{X}, Z_1, \ldots, Z_m), \ldots, r_n(\mathbf{X}, Z_1, \ldots, Z_m),$$

with the set of rules

$$\begin{aligned} r_0(\mathbf{X},\mathbf{Y}) &\rightarrow r_1(\mathbf{X}, f^\sigma_{Z_1}(\mathbf{X}),\ldots,f^\sigma_{Z_m}(\mathbf{X})),\\ &\vdots\\ r_0(\mathbf{X},\mathbf{Y}) &\rightarrow r_n(\mathbf{X}, f^\sigma_{Z_1}(\mathbf{X}),\ldots,f^\sigma_{Z_m}(\mathbf{X})),\end{aligned}$$

where each $f^\sigma_{Z_i}$ is a fresh Skolem function of arity $|\mathbf{X}|$.

Example 3. By Skolemizing the linear TGD $r(X,Y) \rightarrow \exists Z \exists W\, p(X,W), s(W,Z,X)$ we get the two rules $r(X,Y) \rightarrow p(X, f^\sigma_W(X))$ and $r(X,Y) \rightarrow s(f^\sigma_W(X), f^\sigma_Z(X), X)$. ■

The *Skolem terms* that can appear in Σ_f are defined inductively as follows: each variable and constant is a Skolem term, and if f is an n-ary Skolem function, and $t_1,\ldots,t_n$ are Skolem terms, then $f(t_1,\ldots,t_n)$ is a Skolem term.

Now, by exploiting the *predicate graph* G of $\Sigma_f \cup \{\rho\}$ we can identify rules of Σ_f that can be eliminated. G is the graph where the vertices are the predicates occurring in $\Sigma_f \cup \{\rho\}$, and there exists an edge from p_1 to p_2 iff there exists a rule $\tau \in \Sigma_f \cup \{\rho\}$ such that p_2 occurs in $head(\rho)$ and p_1 occurs in $body(\rho)$. All the rules of Σ_f containing at least one predicate p such that there is no path in G from p to q are discarded; recall that q is the head-predicate of the given query Q. Note that Σ_f is actually a set of linear TGDs since, for each $\sigma \in \Sigma_f$, $body(\sigma)$ is a single function-free atom.

Rule Saturation. The saturated set of Σ_f is computed by applying the resolution inference rule. Formally, by applying $\underline{b}_1 \rightarrow \underline{h}_1$ on $\underline{b}_2 \rightarrow \underline{h}_2$, the rule $\gamma(\underline{b}_1) \rightarrow \gamma(\underline{h}_2)$ is obtained, where $\gamma = \mathsf{MGU}(\underline{h}_1, \underline{b}_2)$ is the *most general unifier* for $\underline{h}_1$ and $\underline{b}_2$; we assume that the reader is familiar with the notion of unification (see, e.g., [17]). W.l.o.g. we assume that the above rules do not have variables in common.

In what follows, let Σ_f be the set $\{\sigma_1,\ldots,\sigma_n\}$. Consider a sequence $S = \sigma_{i_1}\ldots\sigma_{i_k}$, for $k \geqslant 1$, of TGDs of Σ_f; note that it is possible to have repeated TGDs. The rule obtained due to S, written $\sigma[i_1\ldots i_k]$, is defined as follows: $\sigma[i_1] = \sigma_{i_1}$, and $\sigma[i_1\ldots i_k]$ is obtained by applying $\sigma[i_1\ldots i_{k-1}]$ on σ_{i_k}.

Example 4. Consider the linear TGDs

$$\begin{aligned} \sigma_1 &: r(X_1,Y_1) \rightarrow s(Y_1, f_1(Y_1)),\\ \sigma_2 &: s(X_2,Y_2) \rightarrow p(Y_2,Y_2,f_2(Y_2)),\\ \sigma_3 &: p(X_3,Y_3,Z_3) \rightarrow t(Z_3).\end{aligned}$$

The rule obtained due to $\sigma_1\sigma_2$, i.e., by applying σ_1 on σ_2, is $\sigma[12] : r(X_1,Y_1) \rightarrow p(f_1(Y_1), f_1(Y_1), f_2(f_1(Y_1)))$, while the rule obtained due to $\sigma_1\sigma_2\sigma_3$, i.e., by applying $\sigma[12]$ on σ_3, is $\sigma[123] : r(X_1,Y_1) \rightarrow t(f_2(f_1(Y_1)))$. As shown in Figure 1(a), $\sigma[12]$ mimics the derivation of the chase of $\{r(a,b)\}$ that involves the TGDs σ_1 and σ_2, while $\sigma[123]$ mimics the derivation that involves the TGDs σ_1, σ_2 and σ_3. ■

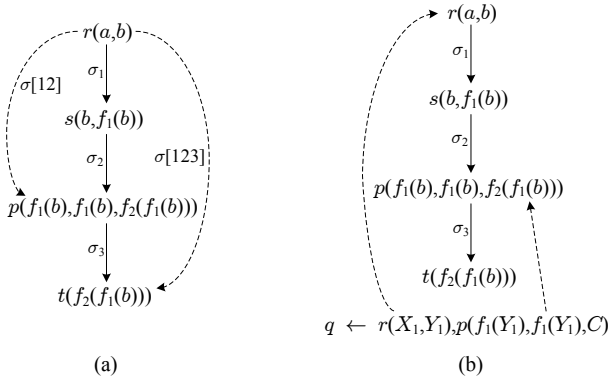

Figure 1: Rule and Query Saturation.

The set $[\Sigma_f]_k$, for $k \geqslant 1$, is the set of all the rules (modulo bijective variable renaming) obtained due to a sequence $\sigma_{i_1}\ldots\sigma_{i_k}$ of TGDs of Σ_f, where $\langle i_1,\ldots,i_k\rangle \in [n]^k$. It is not difficult to see that $[\Sigma_f]_k$ contains all the rules obtained starting from some TGD of Σ_f, and applying $k-1$ times the resolution rule. The *saturated set* of Σ_f, denoted as $[\Sigma_f]$, is the set $[\Sigma_f]_1 \cup [\Sigma_f]_2 \cup \ldots \cup [\Sigma_f]_k$, where $k = \mathsf{b}(Q,\mathcal{R})$; recall that $\mathsf{b}(Q,\mathcal{R})$ is the depth up to which it suffices to construct the chase, under linear TGDs, for CQ answering purposes.

Query Saturation. The saturated query of Q is constructed by applying the resolution inference rule. A useful notion is the *depth* of rules defined as follows. Let t be a Skolem term. If t is a variable or constant, then the depth of t is zero. If $t = f(t_1,\ldots,t_n)$, where f is a Skolem function, then the depth of t is one plus the depth of t_i, where $i \in [n]$, with maximal depth. The *depth* of a rule τ is the depth of the Skolem term with maximal depth that occurs in τ.

In the sequel, let $\mathsf{f}([\Sigma_f])$ be the set $\{\sigma_1,\ldots,\sigma_n\}$ obtained from $[\Sigma_f]$ be keeping only the rules containing Skolem terms. Consider a sequence $S = \sigma_{i_1}\ldots\sigma_{i_k}$, for $k \geqslant 1$, of rules of $\mathsf{f}([\Sigma_f])$; note that repetition of rules in S is allowed. The rule obtained due to S starting from ρ, denoted as $\rho[i_1\ldots i_k]$, is defined as follows: $\rho[i_1]$ is obtained by applying σ_{i_1} on ρ, and $\rho[i_1\ldots i_k]$ by applying σ_{i_k} on $\rho[i_1\ldots i_{k-1}]$.

Example 5. Consider the TGDs σ_1 and σ_2 given in the Example 4, and also the BCQ $Q = \langle q,\tau\rangle$, where τ is the rule $q \leftarrow s(A,B), p(B,B,C)$. By applying σ_1 on τ we get the rule $\tau[1] : q \leftarrow r(X_1,Y_1), p(f_1(Y_1), f_1(Y_1), C)$. The rule obtained due to $\sigma_1\sigma_2$, i.e., by applying σ_2 in $\tau[1]$, is $\tau[12] : q \leftarrow r(X_1,Y_1), s(X_2, f_1(Y_2))$. Finally, the rule obtained by applying σ_1 on $\tau[12]$, is $\tau[121] : q \leftarrow r(X_1,Y_1)$. As depicted in Figure 1(b), with $\tau[1]$ we bypass the derivation of the chase of $D = \{r(a,b)\}$ that involves only σ_1. More precisely, $chase(D,\Sigma)$, where $\Sigma = \{\sigma_1,\sigma_2\}$, entails Q since there exists homomorphism h such that $h(body(\tau)) \subseteq chase(D,\Sigma)$. However, the atom $h(s(A,B))$ is mapped to an atom obtained due to a chase derivation that involves TGDs (in this case σ_1) with Skolem functions. By constructing $\tau[1]$ we bypass this derivation, but the query is still entailed since the new atom $r(X_1,Y_1)$ is mapped to D. ■

The *saturated query* of Q (w.r.t. $\mathsf{f}([\Sigma_f])$), written $[Q,\Sigma_f]$, is the set of all the rules (modulo bijective variable renaming), with depth *at most* $\mathsf{b}(Q,\mathcal{R})$, obtained due to a sequence $\sigma_{i_1}\ldots\sigma_{i_k}$ of TGDs of $\mathsf{f}([\Sigma_f])$, where $k \geqslant 1$ and $\langle i_1,\ldots,i_k\rangle \in [n]^k$, starting from ρ; note that $\rho \in [Q,\Sigma_f]$.

Finalization. We are now ready, by exploiting the set of rules $[\Sigma_f] \cup [Q,\Sigma_f]$, to construct the rewritten query Q_Σ. Let $\mathsf{ff}([\Sigma_f] \cup [Q,\Sigma_f])$ be the set obtained from $[\Sigma_f] \cup [Q,\Sigma_f]$ by keeping only the function-free rules, i.e., the rules without Skolem terms neither in their body nor in their head. Π_{Q_Σ} is obtained by adding to $\mathsf{ff}([\Sigma_f] \cup [Q,\Sigma_f])$ a rule of the form $r(X_1,\ldots,X_n) \rightarrow \hat{r}(X_1,\ldots,X_n)$, for each n-ary predicate $r \in \mathcal{R}$, where $\hat{r}$ is an auxiliary predicate, and by replacing each occurrence of a predicate r in $\mathsf{ff}([\Sigma_f] \cup [Q,\Sigma_f])$ with $\hat{r}$. Q_Σ is the pair $\langle q, \Pi_{Q_\Sigma}\rangle$.

3.3 Termination and Correctness

It is immediate to see that the Skolemization step always terminates (since the given set of TGDs is finite). Obviously, the finalization step terminates, as long as the set $[\Sigma_f] \cup [Q, \Sigma_f]$ is finite. Hence, in order to establish the termination of our algorithm, it remains to show that the rule saturation and the query saturation steps terminate. In the rest of this section, let $Q = \langle q, \rho \rangle$ be a BCQ over a schema $\mathcal{R}$, and Σ be a set of linear TGDs over $\mathcal{R}$.

PROPOSITION 1. *The sets $[\Sigma_f]$ and $[Q, \Sigma_f]$ are finite.*

The rest of this subsection is devoted to establish soundness and completeness. Notice that, due to the *occurs check* in unification, for each $\tau \in [\Sigma_f]$, $body(\tau)$ is function-free. Therefore, $[\Sigma_f]$ is actually a set of linear TGDs. Let us now give some technical results. The first such result implies that the chase under Σ_f can be used for CQ answering purposes.

LEMMA 2. *For every database D for $\mathcal{R}$, $D \cup \Sigma \models Q$ iff $chase^k(D, \Sigma_f) \models Q$, where $k = \mathsf{b}(Q, \mathcal{R})$.*

The next result formalizes the intuitive explanation given for the query saturation step, namely, each rule of $[\Sigma_f]$, obtained by applying k times the resolution rule, "mimics" a derivation of the chase under Σ_f, which involves $k + 1$ applications of the TGD chase rule.

LEMMA 3. *For every database D for $\mathcal{R}$, $chase^k(D, \Sigma_f) = chase(D, [\Sigma_f])$, where $k = \mathsf{b}(Q, \mathcal{R})$.*

The following lemma formalizes the idea of "bypassing" chase derivations which involve TGDs with Skolem terms. In fact, justifies the elimination of the non-function-free rules of $[\Sigma_f] \cup [Q, \Sigma_f]$ during the finalization step.

LEMMA 4. *For every database D for $\mathcal{R}$, $chase(D, [\Sigma_f]) \models Q$ iff there exists a BCQ $Q' = \langle q, \tau \rangle$, where $\tau \in \Pi_{Q_\Sigma}$, such that $chase(D, \Pi_{Q_\Sigma}) \models Q'$.*

By exploiting the above three lemmas, and the fact that $\Pi_{Q_\Sigma}(D)$ and $chase(D, \Pi_{Q_\Sigma})$ coincide, it is easy to establish soundness and completeness of our rewriting algorithm, i.e., for every database D for $\mathcal{R}$, $D \cup \Sigma \models Q$ iff $D \models Q_\Sigma$.

3.4 Structure of the Rewriting

We now consider the structure of the rewriting. By exploiting the notion of BDDP, we identify a semantic class of Datalog programs in which all the *output* IDB predicates, i.e., the predicates that cannot occur in the body of a rule, are bounded. Then, we show that Π_{Q_Σ} falls in this class, and since q is an output IDB predicate, we get that Q_Σ is a bounded Datalog query. In the sequel, if the body of a rule τ contains only EDB predicates, then is called *input* rule, and if the predicate of $head(\tau)$ is an output IDB predicate, then is called *output* rule.

Definition 1. A Datalog program Π is called *output predicate bounded*, abbreviated as *opb-Datalog*, if it can be partitioned into three sets Π_I, Π_B and Π_O such that: *(i)* Π_I is a set of input rules, *(ii)* Π_B falls in a class of TGDs that enjoys the BDDP, and *(iii)* Π_O is a set of output rules.

The following result establishes the boundedness of output IDB predicates in opb-Datalog programs (hence the name "output predicate bounded"). In fact, the strong connection among predicate boundedness and BDDP is shown.

PROPOSITION 5. *Consider an opb-Datalog program Π. Each output IDB predicate is bounded in Π.*

It is not difficult to verify that Π_{Q_Σ} can be partitioned into the sets Π_I, Π_B and Π_O, as in the Definition 1, where Π_I are the auxiliary rules introduced during the finalization step, Π_B are the (function-free) rules generated during the rule saturation step, and Π_O are the (function-free) rules generated during the query saturation step. Since Π_{Q_Σ} is an opb-Datalog program, and q is an output IDB predicate, by Proposition 5 we get that q is bounded in Π_{Q_Σ}. Therefore, Q_Σ is a bounded Datalog query. Note that if the given set of TGDs is acyclic, then the obtained query is a non-recursive Datalog query.

4. OPTIMIZING THE REWRITING

In this section we define the so-called *query elimination* technique, aiming at optimizing the rewritten query. In particular, we can exploit the saturated set (constructed during the rule saturation step) in order to identify body-atoms in a certain rule (during the query saturation step) that are logically implied, w.r.t. the given set of TGDs, by other atoms in the same rule. These implied atoms can be safely eliminated, and therefore we avoid the construction of superfluous rules during the saturation of the query.

4.1 Atom Coverage

The key idea underlying query elimination is formalized by the notion of *atom coverage*. Consider a rule ρ. The *partial freezing* of an atom $\underline{a}$ w.r.t. ρ, denoted as $\mathsf{fr}_\rho(\underline{a})$, is the atom obtained from $\underline{a}$ as follows: replace each variable V that occurs in $body(\rho) \setminus \{\rho\}$ with the constant $c_V \in \Gamma$. Let us now give the formal definition of atom coverage.

Definition 2. Consider a rule ρ, a set Σ of linear TGDs, and let $\underline{a}$ and $\underline{b}$ be atoms of $body(\rho)$. $\underline{a}$ *covers* $\underline{b}$ w.r.t. Σ, written $\underline{a} \prec^\rho_\Sigma \underline{b}$, iff the following condition holds: there exists a TGD $\sigma \in [\Sigma_f]$ and homomorphisms h and μ such that, $h(body(\sigma)) = \mathsf{fr}_\rho(\underline{a})$ and $h(head(\sigma)) = \mu(\mathsf{fr}_\rho(\underline{b}))$.

Roughly speaking, the existence of h implies that an atom with the same equality type as $\underline{a}$ triggers some TGD during the chase under Σ, while the existence of μ ensures that an atom with the same equality type as $\underline{b}$ is mapped to the generated atom. Thus, if $\underline{a}$ is entailed by the chase, then also $\underline{b}$ is entailed, and we can safely eliminate it.

LEMMA 6. *Consider a rule ρ, and a set Σ of linear TGDs. Suppose that $\underline{a} \prec^\rho_\Sigma \underline{b}$, where $\underline{a}, \underline{b} \in body(\rho)$, and let ρ' be such that $body(\rho') = body(\rho) \setminus \{\underline{b}\}$. Then, for each instance I that satisfies $[\Sigma_f]$, there exists h_ρ such that $h_\rho(body(\rho)) \subseteq I$ iff there exists $h_{\rho'}$ such that $h_{\rho'}(body(\rho')) \subseteq I$.*

4.2 Unique Elimination Strategy

An *atom elimination strategy* for a rule ρ is a permutation of its body-atoms. For each $\underline{a} \in body(\rho)$, the *cover set* of $\underline{a}$, written $cover(\underline{a}, \rho, \Sigma)$, is the set $\{\underline{b} \mid \underline{b} \in body(\rho) \wedge \underline{b} \prec^\rho_\Sigma \underline{a}\}$; when ρ and Σ are obvious from the context, we shall denote the above set as $cover(\underline{a})$. By exploiting the cover set of the body-atoms, we associate to each atom elimination strategy S for ρ a subset of $body(\rho)$, denoted as $Eliminate(\rho, \Sigma, S)$, which is actually the set of body-atoms of ρ that can be eliminated (according to S). Formally, the set $Eliminate(\rho, \Sigma, S)$ is computed by applying the following procedure; in the sequel, let $S = [\underline{a}_1, \ldots, \underline{a}_n]$, where $n = |body(\rho)|$:

$A := \varnothing$
for each $i := 1$ to n do
  $\underline{a} := S[i]$
  if $cover(\underline{a}) \neq \varnothing$ then
    $A := A \cup \{\underline{a}\}$
    for each $\underline{b} \in body(\rho) \setminus A$
      $cover(\underline{b}) := cover(\underline{b}) \setminus \{\underline{a}\}$
return A.

We are now ready to establish the uniqueness (w.r.t. the number of eliminated atoms) of the atom elimination strategy for a rule. The proof of this result uses heavily the fact that the relation $\prec^{\rho}_{\Sigma}$ is transitive.

LEMMA 7. *Consider a rule ρ, and a set Σ of linear TGDs. Let S_1 and S_2 be arbitrary atom elimination strategies for ρ. It holds that, $|Eliminate(\rho, \Sigma, S_1)| = |Eliminate(\rho, \Sigma, S_2)|$.*

Henceforth, we refer to *the* atom elimination strategy for a rule ρ by S_ρ.

4.3 Query Elimination

Let us now describe how query elimination can be incorporated into our rewriting algorithm. Recall that the algorithm accepts as input a BCQ $Q = \langle q, \rho \rangle$ over a schema $\mathcal{R}$, and a set Σ of linear TGDs over $\mathcal{R}$.

First, note that the Skolemization step is not affected. Suppose now that $\rho^\star$ is obtained by eliminating from $body(\rho)$ the atoms $Eliminate(\rho, \Sigma, S_\rho)$, and let $Q^\star = \langle q, \rho^\star \rangle$. The *optimized saturated set* of Σ_f, denoted as $[\Sigma_f]^\star$, is the set $[\Sigma_f]_1 \cup [\Sigma_f]_2 \cup \ldots \cup [\Sigma_f]_k$, where $k = \mathsf{b}(Q^\star, \mathcal{R})$, i.e., the only difference during the rule saturation step is the value of k.

Now, let $\mathsf{f}([\Sigma_f]) = \{\sigma_1, \ldots, \sigma_n\}$. Consider a sequence $S = \sigma_{i_1} \ldots \sigma_{i_k}$, for $k \geqslant 1$, of TGDs of $\mathsf{f}([\Sigma_f])$. The *optimized rule* obtained due to S starting from $\rho^\star$, denoted as $\rho^\star[i_1 \ldots i_k]$, is defined as follows: $\rho^\star[i_1]$ is the rule obtained by eliminating from $body(\rho')$ the atoms $Eliminate(\rho', \Sigma, S_{\rho'})$, where ρ' is obtained by applying σ_{i_1} on $\rho^\star$, and $\rho^\star[i_1 \ldots i_k]$ is obtained by eliminating from $body(\rho'')$ the set of atoms $Eliminate(\rho'', \Sigma, S_{\rho''})$, where ρ'' is obtained by applying σ_{i_k} on $\rho^\star[i_1 \ldots i_{k-1}]$. The *optimized saturated query* of Q (w.r.t. $\mathsf{f}([\Sigma_f])$), written $[Q, \Sigma_f]^\star$, is the set of the optimized rules (modulo variable renaming), with depth at most $\mathsf{b}(Q^\star, \mathcal{R})$, obtained due to a sequence $\sigma_{i_1} \ldots \sigma_{i_k}$ of TGDs of $\mathsf{f}([\Sigma_f])$, where $k \geqslant 1$ and $\langle i_1, \ldots, i_k \rangle \in [n]^k$, starting from $\rho^\star$; note that $\rho^\star \in [Q, \Sigma_f]^\star$. The finalization step is as defined in Subsection 3.2, except that the optimized saturated set $[Q, \Sigma_f]^\star$ is considered. The final *optimized rewriting* $Q^\star_\Sigma$ is the pair $\langle q, \Pi_{Q^\star_\Sigma} \rangle$.

Notice that the finiteness of the saturated query and the structure of the rewriting are not affected. By exploiting Lemma 6, and by adapting analogously Lemma 4, it can be shown that $Q^\star_\Sigma$ is indeed a perfect rewriting. For an example of the execution of the algorithm see Appendix D.

Let us now consider the size of the rewriting. Note that the schema $\mathcal{R}$ is considered fixed.

LEMMA 8. *The maximal size of the set $[\Sigma_f]^\star \cup [Q, \Sigma_f]^\star$ is $\mathcal{O}((nm)^m)$, and the maximal size of $\Pi_{Q^\star_\Sigma}$ is $\mathcal{O}((n+m)^m)$, where $n = |\Sigma|$ and $m = |body(\rho^\star)|$.*

Therefore, our algorithm runs in polynomial time in the size of the set of TGDs, and in exponential time in the number of non-eliminable atoms in the body of the query. The same upper bounds hold for the size of the rewritten query.

5. EXPERIMENTAL RESULTS

The rewriting algorithm presented in this paper is implemented in the Nyaya semantic data management system[4]. The rewriting engine is based on the IRIS Datalog Engine[5] extended to support the FO-rewritable Datalog$^\pm$ languages. Both IRIS and our extensions are implemented in Java.

The ideal experimental setting for our rewriting algorithm is an ODBA scenario where the ontological constraints are defined using linear Datalog$^\pm$. Since accepted benchmark for Datalog$^\pm$ ontologies are not yet available, we stick to the same experimental setting of [15], [19] and [21], where the testbed consists of DL-Lite ontologies whose expressive power falls into that of linear Datalog$^\pm$.

STOCKEXCHANGE (S) is an ontology representing the domain of financial institutions. UNIVERSITY (U) is a DL-Lite$_\mathcal{R}$ version of the LUBM Benchmark[6]. ADOLENA (A) describes disabilities and accessibility devices. The last ontology (P5X) is a synthetic ontology representing graph structures and paths of variable length on the graph. The queries used in the experiments and the ontologies can also be found at Nyaya's website.

We named our implementation Nyaya$^{\text{DTG}}$, and we compared it with four other rewriting-based ODBA systems for DL-Lite ontologies. QuOnto and Presto (based on [9] and [21], respectively) developed by the University of Rome La Sapienza, Requiem (based on [19]) developed by the KR group of the University of Oxford and TGD-REWRITE (based on [15]) also implemented in the Nyaya System.

We compared the various approaches in terms of the *size* of the rewriting, i.e., the number of rules, represented in standard Datalog notation, that occur in the rewritten query. Each rule corresponds to a query to be executed by the underlying DBMS, and thus is a key measure of the complexity of the rewriting. Regarding the size of the rewriting generated by Nyaya$^{\text{DTG}}$, we do not consider the auxiliary rules introduced during the finalization step. These rules are introduced for technical reasons, namely, to ensure that the rewriting is syntactically consistent with the definition of Datalog queries. They just copy the database in temporary tables, and are not considered during the query evaluation.

Table 1 shows the outcome of the experimental campaign. By QO, RQ, NY and PR we refer to QuOnto, Requiem, Nyaya and Presto, respectively, while NY$^{\text{DTG}}$ refers to Nyaya$^{\text{DTG}}$. QO, RQ and NY produce a rewriting in the form of a UCQs, PR produces a non-recursive Datalog query, while NY$^{\text{DTG}}$ produces a bounded Datalog query. Note that all these representations can be translated into equivalent first-order queries, and therefore into SQL queries.

From the analysis of the results it is possible to draw some interesting conclusions. In the first place, *(i)* a Datalog representation sensibly compresses the size of the rewriting. This is evident in the results of Presto and Nyaya$^{\text{DTG}}$ for the ontology P5X, whose constraints have been voluntarily engineered to generate an exponential blowup during the computation of a UCQ-rewriting. As expected, *(ii)* by computing the rewriting as a bounded Datalog query, instead of a non-recursive Datalog query, it is possible to further compress the rewriting whenever the predicate graph of the set of constraints is cyclic. More precisely, the auxiliary IDB

[4] http://mais.dia.uniroma3.it/Nyaya/Home.html
[5] http://www.iris-reasoner.org/
[6] http://swat.cse.lehigh.edu/projects/lubm/

Table 1: Comparison evaluation

Ontology	Query	Size				
		QO	RQ	NY	PR	NY^{DTG}
A	Q_1	783	402	249	69	58
	Q_2	1812	103	94	52	41
	Q_3	4763	104	104	55	43
	Q_4	7251	492	456	93	81
	Q_5	66068	624	624	71	65
U	Q_1	5	2	2	6	5
	Q_2	287	148	1	1	1
	Q_3	1260	224	4	8	7
	Q_4	5364	1628	2	6	5
	Q_5	9245	2960	2	11	10
S	Q_1	6	6	6	7	7
	Q_2	204	160	1	3	3
	Q_3	1194	480	2	5	4
	Q_4	1632	960	2	5	4
	Q_5	11487	2880	4	7	6
P5X	Q_1	14	14	10	11	11
	Q_2	86	77	57	16	16
	Q_3	530	390	324	16	16
	Q_4	3,476	1,953	1,642	16	16
	Q_5	23,744	9,766	8,219	16	16

predicates introduced by Presto to "collect" the database tuples needed to entail the input query, have also the effect to unfold the cyclic structures of the rewriting. $Nyaya^{DTG}$, by targeting a bounded Datalog program, preserves these cyclic structures, resulting in a more compact representation. Note that Presto and $Nyaya^{DTG}$ obtain exactly the same result on the set P5X of constraints because this set is acyclic. Finally, *(iii)* a Datalog query is not always more compact than a UCQs; see the ontologies S and U. For simple queries and ontologies, the approaches constructing a Datalog query inevitably generate rules that encode certain structural relationships of the set of constraints, e.g., the most general subsumee in Presto, and the chase derivations in $Nyaya^{DTG}$. This leads to corner cases where a UCQs is better than a Datalog query; for an example see Appendix E.

6. CONCLUSIONS

This paper presented a new query rewriting technique for linear $Datalog^{\pm}$ and DL-Lite. The main idea underlying our algorithm is the construction of a bounded Datalog query. The obtained rewriting, even if is syntactically recursive, can be always translated into an SQL query.

We plan to investigate rewriting techniques for the other first-order rewritable $Datalog^{\pm}$ languages, i.e., sticky and sticky-join $Datalog^{\pm}$. We are convinced that the technique proposed in this paper can be extended in order to treat also these more general languages.

Acknowledgements. This research has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement DIADEM no. 246858 and from the Oxford Martin School's grant no. LC0910-019.

7. REFERENCES

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

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

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

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

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

[6] A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.

[7] A. Calì, G. Gottlob, and A. Pieris. Query answering under non-guarded rules in datalog+/-. In *Proc. of RR*, pages 175–190, 2010.

[8] A. Calì, G. Gottlob, and A. Pieris. Query rewriting under non-guarded rules. In *Proc. AMW*, 2010.

[9] 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):385–429, 2007.

[10] S. S. Cosmadakis, H. Gaifman, P. C. Kanellakis, and M. Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *Proc. of STOC*, pages 477–490, 1988.

[11] R. de Virgilio, F. Giunchiglia, and L. Tanca, editors. *Semantic Web Information Management: A Model-Based Perspective*. Springer Verlag, 2010.

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

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

[14] H. Gaifman, H. G. Mairson, Y. Sagiv, and M. Y. Vardi. Undecidable optimization problems for database logic programs. In *Proc. of LICS*, pages 106–115, 1987.

[15] G. Gottlob, G. Orsi, and A. Pieris. Ontological queries: Rewriting and optimization. In *Proc. of ICDE*, pages 2–13, 2011.

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

[17] J. W. Lloyd. *Foundations of Logic Programming*. Springer, 1984.

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

[19] H. Pérez-Urbina, B. Motik, and I. Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):151–232, 2009.

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

[21] R. Rosati and A. Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. KR*, pages 290–300, 2010.

APPENDIX

A. TRANSLATION INTO SQL

Recall that the perfect rewriting Q_Σ of Example 2 is the logical *union* of the CQs

$$Q : q(B) \leftarrow \mathit{father}(A, B) \text{ and } Q' : q(B) \leftarrow \mathit{person}(B).$$

Assuming that Q and Q' are queries over the relational schema $\{\mathit{father}(f, p), \mathit{person}(p)\}$, Q_Σ can be written in SQL as follows:

```
SELECT father.p FROM father
UNION
SELECT person.p FROM person.
```

B. PRELIMINARIES

B.1 Predicate Boundedness

Consider the (recursive) Datalog program Π consisting of the rules

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

Observe that, for every database D, $T^2_{\Pi,q}(D) = T^\omega_{\Pi,q}(D)$. For example, if $D = \{r(a,b), r(c,d), t(e)\}$, then

$$T^1_{\Pi,q}(D) \;=\; \{q(a), q(c)\}$$

while

$$T^2_{\Pi,q}(D) \;=\; T^1_{\Pi,q}(D) \cup \{q(b), q(d)\} \;=\; T^\omega_{\Pi,q}(D).$$

Consequently, the predicate q is bounded in Π.

Let us now show that the predicate t is unbounded in Π. Towards a contradiction, suppose that t is bounded, or, equivalently, there exists $k \geqslant 0$ such that, for every database D, $T^k_{\Pi,q}(D) = T^\omega_{\Pi,q}(D)$. Consider now the database

$$D \;=\; \{r(c_1, c_2), \ldots, r(c_k, c_{k+1}), r(c_{k+1}, c_{k+2}), t(c_{k+2})\}.$$

It is not difficult to verify that

$$T^{k+1}_{\Pi,q}(D) \;=\; T^k_{\Pi,q}(D) \cup \{t(c_1)\}$$

which is a contradiction. Thus, t is unbounded in Π.

Since q is bounded in Π it follows that, for every database D, we can obtain all the atoms of $\Pi(D)$ with predicate q using a non-recursive Datalog program. In particular, the non-recursive Datalog program Π_q comprised by the rules

$$\begin{array}{rcl} r(X,Y) & \rightarrow & \mathit{aux}(X,Y) \\ r(X,Y) & \rightarrow & \mathit{aux}(Y,X) \\ \mathit{aux}(X,Y), t(Z) & \rightarrow & q(X), \end{array}$$

is such that $q(\Pi(D)) = q(\Pi_q(D))$, for every database D; let us recall that given an instance I, by $r(I)$ we denote the set of tuples $\{\mathbf{t} \mid r(\mathbf{t}) \in I\}$.

B.2 TGD Chase Procedure

In what follows we assume a lexicographic order on $\Gamma \cup \Gamma_N$, such that every value in Γ_N follows all those in Γ. The *chase* of a database D w.r.t. a set Σ of TGDs, denoted by $\mathit{chase}(D, \Sigma)$, is the (possibly infinite) instance constructed by an iterative application of the TGD chase rule in a breadth-first (level-saturating) fashion —the lower the level of an atom, the earlier the atom has been obtained during the construction of the chase— as follows:

1. Let $\mathit{chase}(D, \Sigma) = D$, and for each $\underline{a} \in D$ let the *(derivation) level* of $\underline{a}$, denoted as $\mathit{level}(\underline{a})$, to be zero.
2. Let $\sigma_1, \ldots, \sigma_n$, for $n \geqslant 0$, be the TGDs which are applicable to $\mathit{chase}(D, \Sigma)$; assume that each $\sigma \in \{\sigma_1, \ldots, \sigma_n\}$ is applicable due to the homomorphisms $h^1_\sigma, \ldots, h^{m_\sigma}_\sigma$, for $m_\sigma \geqslant 1$, and let $I^i_\sigma = h^i_\sigma(\mathit{body}(\sigma))$, for each $i \in [m_\sigma]$.
3. Let $k = \min_{I \in \{I^1_\sigma, \ldots, I^{m_\sigma}_\sigma\}} \{\max_{\underline{a} \in I} \{\mathit{level}(\underline{a})\}\}$.
4. For each $\sigma \in \{\sigma_1, \ldots, \sigma_n\}$, let S_σ be the set of images $\{I \mid I \in \{I^1_\sigma, \ldots, I^{m_\sigma}_\sigma\}$ and $\max_{\underline{a} \in I} \{\mathit{level}(\underline{a})\} = k\}$, and if $S_\sigma \neq \varnothing$, then let $\widehat{I_\sigma}$ be the image of S_σ that precedes in lexicographic order.
5. Apply the TGD σ of $\{\sigma \mid S_\sigma \neq \varnothing\}$ that precedes in lexicographic order by utilizing the homomorphism h that maps $\mathit{body}(\sigma)$ to $\widehat{I_\sigma}$, and assign to every new atom the (derivation) level $k + 1$.
6. Goto step 2.

The *chase of level up to* k, for some integer $k \geqslant 0$, of D w.r.t. Σ, denoted as $\mathit{chase}^k(D, \Sigma)$, is the set of all the atoms of $\mathit{chase}(D, \Sigma)$ of level at most k.

Example B.1. Consider the set Σ consisting of the TGDs $\sigma_1 : r(X,Y), s(Y) \rightarrow \exists Z\, r(Z,X)$ and $\sigma_2 : r(X,Y) \rightarrow s(X)$, and let D be the database $\{r(a,b), s(b)\}$. During the construction of $\mathit{chase}(D, \Sigma)$ 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 $D \cup \{r(z_1, a), s(a), r(z_2, z_1), s(z_1), r(z_3, z_2), s(z_2), \ldots\}$. ■

C. QUERY REWRITING

In the sequel, let $Q = \langle q, \rho \rangle$ be a BCQ over a schema $\mathcal{R}$, and Σ be a set of linear TGDs over $\mathcal{R}$.

C.1 Termination and Correctness

PROPOSITION C.1. *The sets* $[\Sigma_f]$ *and* $[Q, \Sigma_f]$ *are finite.*

PROOF (SKETCH). Recall that $[\Sigma_f] = \Sigma_f \cup [\Sigma_f]_2 \cup \ldots \cup [\Sigma_f]_{\mathsf{b}(Q,\mathcal{R})}$. It is not difficult to see that each $[\Sigma_f]_i$ is finite. In particular, in a set $[\Sigma_f]_k$, for $k \geqslant 2$, we can have at most $|\Sigma_f|^k$ rules since the maximum number of TGD sequences of length k that we can build is $|\Sigma_f|^k$, and the rule obtained from a certain sequence is unique (modulo bijective variable renaming). The finiteness of the set $[Q, \Sigma_f]$ follows from the following two observations: *(i)* since the TGDs of Σ_f are linear, during the construction of $[Q, \Sigma_f]$ it is not possible to a get rule τ such that $|\mathit{body}(\tau)| > |\mathit{body}(\rho)|$, and *(ii)* by construction, it is not possible to have in $[Q, \Sigma_f]$ a Skolem term with depth more than $\mathsf{b}(Q, \mathcal{R})$. These two observations imply that we can have only finitely many rules in $[Q, \Sigma_f]$ (modulo bijective variable renaming). □

LEMMA C.2. *For every database* D *for* $\mathcal{R}$, $D \cup \Sigma \models Q$ *iff* $\mathit{chase}^k(D, \Sigma_f) \models Q$, *where* $k = \mathsf{b}(Q, \mathcal{R})$.

PROOF (SKETCH). It is possible to establish that there exists an isomorphism h from $\Gamma \cup \Gamma_N$ to the set of Skolem terms that can appear in $\mathit{chase}(D, \Sigma_f)$ with the following

properties: *(i)* $h(chase(D,\Sigma)) = chase(D,\Sigma_f)$, and *(ii)* if $z \in \Gamma_N$, then $h(z)$ is a Skolem term with depth greater than zero. More precisely, a null $z \in \Gamma_N$ that occurs in an atom of $chase(D,\Sigma)$ at level $\ell > 0$, is mapped by h to a Skolem term with dept ℓ. Therefore, $chase^k(D,\Sigma) \models Q$ iff $chase^k(D,\Sigma_f) \models Q$. The claim follows since, due to the BDDP of linear TGDs, $D \cup \Sigma \models Q$ iff $chase^k(D,\Sigma)$. □

LEMMA C.3. *For every D for $\mathcal{R}$, $chase^k(D,\Sigma_f) = chase(D,[\Sigma_f])$, where $k = \mathsf{b}(Q,\mathcal{R})$.*

PROOF (SKETCH). By construction $\Sigma_f \subseteq [\Sigma_f]$. This implies that $chase(D,\Sigma_f) \subseteq chase(D,[\Sigma_f])$, and therefore $chase^k(D,\Sigma_f) \subseteq chase(D,[\Sigma_f])$. It remains to establish the other direction. It is not difficult to show that all the atoms of $chase(D,[\Sigma_f])$ are obtained during the construction of its first level, or, equivalently, $chase(D,[\Sigma_f])$ coincides with $chase^1(D,[\Sigma_f])$. Therefore, it remains to show that $chase^1(D,[\Sigma_f]) \subseteq chase(D,\Sigma_f)$. Suppose that $\Sigma_f = \{\sigma_1,\ldots,\sigma_n\}$. It can be shown that, for each atom $\underline{a} \in chase^1(D,[\Sigma_f])$ obtained by applying a TGD $\sigma \in [\Sigma_f]$, which in turn was obtained due to a sequence $\sigma_{i_1}\ldots\sigma_{i_\ell}$, for $\ell \geqslant 1$, of TGDs of Σ_f, $\underline{a} \in chase^\ell(D,\Sigma_f)$; the proof is by induction on ℓ. The claim follows since $\ell \leqslant k$. □

LEMMA C.4. *For every D for $\mathcal{R}$, $chase(D,[\Sigma_f]) \models Q$ iff there exists a BCQ $Q' = \langle q,\tau\rangle$, where $\tau \in \Pi_{Q_\Sigma}$, such that $chase(D,\Pi_{Q_\Sigma}) \models Q'$.*

PROOF (SKETCH). ($\Rightarrow$) It suffices to show the following: if there exists $\rho' \in [Q,\Sigma_f]$ such that there exists a homomorphism h that maps $body(\rho')$ to $chase(D,[\Sigma_f])$, then the BCQ Q' exists. Since $chase(D,[\Sigma_f])$ and $chase^1(D,[\Sigma_f])$ coincide, $h(body(\rho')) \subseteq chase^1(D,[\Sigma_f])$. The proof is by induction on the number of atoms in $h_\tau(body(\rho'))$ obtained during the chase due to a TGD containing Skolem functions.

($\Leftarrow$) By hypothesis, there exists a homomorphism h such that $h(body(\tau)) \subseteq chase(D,[\Sigma_f])$. Suppose that τ was obtained due to the sequence $\sigma_1\ldots\sigma_m$ of TGDs of $\mathsf{f}([\Sigma_f])$, where $m \geqslant 1$, starting from ρ. Let τ_i be the (intermediate) rule obtained due to the sequence $\sigma_1\ldots\sigma_i$, for $i \in [m]$, starting from ρ. It is possible to show the following auxiliary claim:

CLAIM C.5. *Consider an arbitrary rule τ_i, for $i \in [m]$. If there exists a homomorphism h_i such that $h_i(body(\tau_i)) \subseteq chase(D,[\Sigma_f])$, then there exists a homomorphism h_{i-1} such that $h_{i-1}(body(\tau_{i-1})) \subseteq chase(D,[\Sigma_f])$.*

Clearly, by starting from $\tau_m = \tau$, and applying m times the above claim, we immediately get that there exists a homomorphism that maps $body(\rho)$ to $chase(D,[\Sigma_f])$, and the claim follows. □

THEOREM C.6. *For every database D for $\mathcal{R}$, $D \cup \Sigma \models Q$ iff $D \models Q_\Sigma$.*

PROOF (SKETCH). By Lemmas C.2–C.4, $D \cup \Sigma \models Q$ iff there exists a BCQ $Q' = \langle q,\tau\rangle$, where $\tau \in \Pi_{Q_\Sigma}$, such that $chase(D,\Pi_{Q_\Sigma}) \models Q'$, for every database D for $\mathcal{R}$. Note that $T^\omega_{\Pi_{Q_\Sigma}}(D)$ and $chase(D,\Pi_{Q_\Sigma})$ coincide. Since $T^\omega_{\Pi_{Q_\Sigma}}(D)$ and $\Pi_{Q_\Sigma}(D)$ are equal, then $q \in \Pi_{Q_\Sigma}(D)$; thus, $D \models Q_\Sigma$. □

C.2 Structure of the Rewriting

PROPOSITION C.7. *Consider an opb-Datalog program Π. Each output IDB predicate is bounded in Π.*

PROOF (SKETCH). The claim is equivalent to the following statement: for every output IDB predicate p, there exists a constant $k_p \geqslant 0$ such that $T^\omega_{\Pi,p}(D) = T^{k_p}_{\Pi,p}(D)$, for every database D for $\mathcal{R}$. Fix an arbitrary output IDB predicate p. Since $T^\omega_\Pi(D) = chase(D,\Pi)$, we immediately get that $T^\omega_{\Pi,p}(D)$ is the set $\{p(\mathbf{t}) \mid p(\mathbf{t}) \in chase(D,\Pi)\}$. By hypothesis, Π can be partitioned into Π_I, Π_B and Π_O as in the Definition 1. It is not difficult to see that $chase(D,\Pi) = chase(chase(D_I,\Pi_B),\Pi_O)$, where D_I is the (finite) instance $chase(D,\Pi_I)$. Let Π^p_O be the set of rules of Π_O with head-predicate p, and for each $\tau \in \Pi^p_O$, let Q^p_τ be the CQ $\langle p,\tau\rangle$. Clearly, $T^\omega_{\Pi,p}(D)$ is the set of atoms

$$\left\{p(\mathbf{t}) \mid \mathbf{t} \in \bigcup_{\tau\in\Pi^p_O} Q^p_\tau\left(chase(D_I,\Pi_B)\right)\right\}.$$

Since Π_B falls in a class of TGDs which enjoys the BDDP, we get that $T^\omega_{\Pi,p}(D)$ is the set

$$\left\{p(\mathbf{t}) \mid \mathbf{t} \in \bigcup_{\tau\in\Pi^p_O} Q^p_\tau(chase^{k_p}(D_I,\Pi_B))\right\},$$

where $k_p = \max_{\tau\in\Pi^p_O}\{\mathsf{b}(Q^p_\tau,\mathcal{R})\}$, which is exactly the set $T^{k_p}_{\Pi,p}(D)$, and the claim follows. □

D. OPTIMIZING THE REWRITING

D.1 Atom Coverage

LEMMA D.1. *Consider a rule ρ, and a set Σ of linear TGDs. Suppose that $\underline{a} \prec^\rho_\Sigma \underline{b}$, where $\underline{a},\underline{b} \in body(\rho)$, and let ρ' be such that $body(\rho') = body(\rho) \setminus \{\underline{b}\}$. Then, for each I that satisfies $[\Sigma_f]$, there exists h_ρ such that $h_\rho(body(\rho)) \subseteq I$ iff there exists $h_{\rho'}$ such that $h_{\rho'}(body(\rho')) \subseteq I$.*

PROOF. ($\Rightarrow$) By hypothesis, there exists a homomorphism h that maps $body(\rho)$ to I. Since $body(\rho') \subset body(\rho)$, we immediately get that h maps $body(\rho')$ to I.

($\Leftarrow$) By hypothesis, there exists a homomorphism h such that $h(body(\rho')) \subseteq I$. Clearly, $h(body(\rho) \setminus \{\underline{b}\}) \subseteq I$. Since $\underline{a} \prec^\rho_\Sigma \underline{b}$ it follows that there exists $\sigma \in [\Sigma_f]$, and homomorphisms λ and μ such that $\lambda(body(\sigma)) = \mathsf{fr}_\rho(\underline{a})$ and $\mu(head(\sigma)) = \mu(\mathsf{fr}_\rho(\underline{b}))$. Let $\hat\lambda$ (resp., $\hat\mu$) be the substitutions obtained from λ (resp., μ) by replacing each occurrence of a constant c_X with the variable X. It is easy to verify that $\hat\lambda(body(\sigma)) = \underline{a}$ and $\hat\lambda(head(\sigma)) = \hat\mu(\underline{b})$. Since $h \circ \hat\lambda$ maps $body(\sigma)$ to I, and I satisfies $[\Sigma_f]$, we get that $h \circ \hat\lambda(head(\sigma)) \in I$. Thus, $h(\hat\lambda(body(\sigma))) = h(\hat\mu(\underline{b})) \in I$. The substitution $\hat\mu$ is the identity on the variables that occur in atoms of $body(\sigma) \setminus \{\underline{b}\}$, and thus h and $h \circ \hat\mu$ are compatible. Therefore, the desired homomorphism is $h \cup (h \circ \hat\mu)$. □

D.2 Unique Elimination Strategy

LEMMA D.2. *Consider a rule ρ, and a set Σ of linear TGDs. Let S_1 and S_2 be arbitrary atom elimination strategies for ρ. It holds that, $|Eliminate(\rho,\Sigma,S_1)| = |Eliminate(\rho,\Sigma,S_2)|$.*

PROOF (SKETCH). We assume that S_1 and S_2 are exactly the same except two consecutive elements. In other words, for each $i \in \{1, \ldots, k-1, k+2, \ldots, n\}$, $S_1[i] = S_2[i]$, $S_1[k] = S_2[k+1]$ and $S_1[k+1] = S_2[k]$. Note that the above assumption does not harm the generality of the proof since, any given two strategies S and S', S can be obtained from S' (and vice versa) by applying a sequence of transformations, where two consecutive (intermediate) strategies are exactly the same except two consecutive elements. For example, consider the strategies $S = [\underline{a}, \underline{b}, \underline{c}, \underline{d}]$ and $S' = [\underline{c}, \underline{a}, \underline{d}, \underline{b}]$. S' can be obtained from S as follows: $S = [\underline{a}, \underline{b}, \underline{c}, \underline{d}] \rightarrow [\underline{a}, \underline{c}, \underline{b}, \underline{d}] \rightarrow [\underline{a}, \underline{c}, \underline{d}, \underline{b}] \rightarrow [\underline{c}, \underline{a}, \underline{d}, \underline{b}] = S'$. Let us now establish the claim. For brevity, given a strategy S, let $\mathit{Eliminate}^{\ell}(\rho, \Sigma, S)$ be the subset of $\mathit{Eliminate}(\rho, \Sigma, S)$ computed after ℓ applications of the for-loop. Clearly, $\mathit{Eliminate}^{k-1}(\rho, \Sigma, S_1)$ and $\mathit{Eliminate}^{k-1}(\rho, \Sigma, S_2)$ are equal. In what follows, let $\underline{a}_k = S_1[k] = S_2[k+1]$ and $\underline{a}_{k+1} = S_1[k+1] = S_2[k]$. The proof proceeds by case analysis whether $\mathit{cover}(\underline{a}_k)$ and $\mathit{cover}(\underline{a}_{k+1})$ are empty or not after $k-1$ applications of the for-loop. By exploiting the fact that the binary relation $\prec^{\rho}_{\Sigma}$ is transitive, it is possible to show that the cases

- $\mathit{cover}(\underline{a}_k) \supset \{\underline{a}_{k+1}\}$ and $\mathit{cover}(\underline{a}_{k+1}) = \{\underline{a}_k\}$,
- $\mathit{cover}(\underline{a}_k) = \{\underline{a}_{k+1}\}$ and $\mathit{cover}(\underline{a}_{k+1}) \supset \{\underline{a}_k\}$,
- $\mathit{cover}(\underline{a}_k) = \{\underline{a}_{k+1}\}$ and $\underline{a}_k \notin \mathit{cover}(\underline{a}_{k+1})$,
- $\mathit{cover}(\underline{a}_{k+1}) = \{\underline{a}_k\}$ and $\underline{a}_{k+1} \notin \mathit{cover}(\underline{a}_k)$,

are excluded since always we get a contradiction. □

D.3 Query Elimination

Example D.1. Let $\mathcal{R} = \{r, s, p\}$ be a relational schema. Consider the set

$$\Sigma \;=\; \begin{cases} r(X,Y) \rightarrow \exists Z\, s(X,Z,Y), p(Z) \\ s(X,Z,Y) \rightarrow r(Y,X), \end{cases}$$

of linear TGDs over $\mathcal{R}$, and the BCQ $Q = \langle q, \tau \rangle$, where τ is the rule $q \leftarrow r(A,B), s(A,C,B), p(B)$. By Skolemizing Σ we get the set

$$\Sigma_f \;=\; \begin{cases} \sigma_1 \;:\; r(X,Y) \rightarrow s(X, f(X,Y), Y) \\ \sigma_2 \;:\; r(X,Y) \rightarrow p(f(X,Y)) \\ \sigma_3 \;:\; s(X,Z,Y) \rightarrow r(Y,X). \end{cases}$$

The saturated set $[\Sigma_f]$ of Σ_f, constructed during the rule saturation step, is the union $\mathsf{ff}([\Sigma_f]) \cup \mathsf{f}([\Sigma_f])$, where

$$\mathsf{ff}([\Sigma_f]) \;=\; \begin{cases} \sigma_3 \;:\; s(X,Z,Y) \rightarrow r(Y,X) \\ \sigma[13] \;:\; r(X,Y) \rightarrow r(Y,X) \\ \sigma[313] \;:\; s(X,Z,Y) \rightarrow r(X,Y), \end{cases}$$

is the set of function-free rules of $[\Sigma_f]$, while

$$\mathsf{f}([\Sigma_f]) \;=\; \begin{cases} \sigma_1 \;:\; r(X,Y) \rightarrow s(X, f(X,Y), Y) \\ \sigma_2 \;:\; r(X,Y) \rightarrow p(f(X,Y)) \\ \sigma[31] \;:\; s(X,Z,Y) \rightarrow s(Y, f(Y,X), X) \\ \sigma[32] \;:\; s(X,Z,Y) \rightarrow p(f(Y,X)) \\ \sigma[131] \;:\; r(X,Y) \rightarrow s(Y, f(Y,X), X) \\ \sigma[132] \;:\; r(X,Y) \rightarrow p(f(Y,X)) \\ \sigma[312] \;:\; s(X,Z,Y) \rightarrow p(f(X,Y)) \\ \sigma[3131] \;:\; s(X,Z,Y) \rightarrow s(X, f(X,Y), Y). \end{cases}$$

are the rules of $[\Sigma_f]$ containing Skolem functions.

Now, consider the rule τ. It is easy to verify that the atom $r(A,B)$ covers $s(A,C,B)$ due to the existence σ_1. In particular, the homomorphism $h = \{X \rightarrow c_A, Y \rightarrow c_b\}$, where $c_A, c_B \in \Gamma$, maps $\mathit{body}(\sigma_1)$ to $\mathsf{fr}_{\tau}(r(A,B)) = r(c_A, c_B)$, and $\mu = \{C \rightarrow f(c_A, c_B)\}$ maps $\mathsf{fr}_{\tau}(s(A,C,B)) = s(c_A, C, c_B)$ to $h(\mathit{head}(\sigma_1))$; in fact, $\mathit{Eliminate}(\tau, \Sigma, S_{\tau}) = \{s(A,C,B)\}$. Thus, the rule $\tau^{\star}$ obtained by eliminating from $\mathit{body}(\tau^{\star})$ the set $\mathit{Eliminate}(\tau, \Sigma, S_{\tau})$ is $q \leftarrow r(A,B), p(B)$.

We now proceed by applying the query saturation step. As it can be verified, the atom $r(A,B)$ does not unify with any of the head-atoms of $\mathsf{f}([\Sigma_f])$, while the atom $p(B)$ unifies with the head-atoms of σ_2, $\sigma[32]$, $\sigma[132]$ and $\sigma[312]$. For instance, by applying σ_2 on $\tau^{\star}$ the rule $\tau^{\star}[2] : q \leftarrow r(A, f(X,Y)), r(X,Y)$ is obtained. It is not difficult to verify that all the rules that we obtain during the saturation of the query contain Skolem functions, and thus they do not contribute to the final rewriting.

During the finalization step, the Datalog program

$$\Pi_{Q_\Sigma} \;=\; \begin{cases} \hat{r}(X,Y) \leftarrow \hat{s}(X,Z,Y) \\ \hat{r}(Y,X) \leftarrow \hat{r}(X,Y) \\ \hat{r}(Y,X) \leftarrow \hat{s}(X,Z,Y) \\ q \leftarrow \hat{r}(A,B), \hat{p}(B) \\ \hat{r}(X,Y) \leftarrow r(X,Y) \\ \hat{s}(X,Y,Z) \leftarrow s(X,Y,Z) \\ \hat{p}(X) \leftarrow p(X) \end{cases}$$

is constructed, which is actually the rules of $\mathsf{ff}([\Sigma_f]) \cup \{\tau\}$ (after renaming the predicates) plus the auxiliary rules. The final rewritten query Q_Σ is the pair $\langle q, \Pi_{Q_\Sigma} \rangle$. Notice that the above query, although is syntactically recursive, is bounded, and therefore it is possible to obtain all the atoms of $\Pi_{Q_\Sigma}(D)$ with predicate q, for every database D, using a non-recursive Datalog query (and thus, a first-order query). ■

LEMMA D.3. *The maximal size of the set $[\Sigma_f]^{\star} \cup [Q, \Sigma_f]^{\star}$ is $\mathcal{O}((nm)^m)$, and the maximal size of $\Pi_{Q^{\star}_\Sigma}$ is $\mathcal{O}((n+m)^m)$, where $n = |\Sigma|$ and $m = |\mathit{body}(\rho^{\star})|$.*

PROOF (SKETCH). Let s be the size of the Skolemized set Σ_f. Recall that $[\Sigma_f]^{\star} = [\Sigma_f]_1 \cup [\Sigma_f]_2 \cup \ldots \cup [\Sigma_f]_k$, where $k = \mathsf{b}(Q^{\star}, \mathcal{R})$. It is clear that in a set $[\Sigma_f]_i$, for $i \geqslant 1$, we can have at most s^i rules since the maximum number of TGD sequences of length i that we can build is s^i, and the rule obtained due to a certain sequence is unique (modulo bijective variable renaming). Consequently, $|[\Sigma_f]^{\star}| \leqslant s + s^2 + \ldots + s^k = \mathcal{O}(s^m)$. Now, observe that, since the TGDs of Σ_f are linear, in $[Q, \Sigma_f]^{\star}$ it is not possible to have a rule τ such that $|\mathit{body}(\tau)| > |\mathit{body}(\rho^{\star})|$, and by construction, it is not possible to have in $[Q, \Sigma_f]^{\star}$ a Skolem term with depth more than $\mathsf{b}(Q^{\star}, \mathcal{R})$. Therefore, $|[Q, \Sigma_f]^{\star}| \leqslant (|\mathcal{R}| \cdot (f^k \cdot (w + c_{\rho^{\star}}) \cdot m)^w)^m = \mathcal{O}((sm)^m)$, where $k = \mathsf{b}(Q^{\star}, \mathcal{R})$, f is the number of Skolem functions occurring in Σ_f, w is the maximum arity over all predicates of $\mathcal{R}$, and $c_{\rho^{\star}}$ is the number of constants occurring in $\rho^{\star}$. Therefore, $|[\Sigma_f]^{\star} \cup [Q, \Sigma_f]^{\star}| = \mathcal{O}(s^m) + \mathcal{O}((sm)^m) = \mathcal{O}((sm)^m)$. Clearly, to establish an upper bound on the size of $\Pi_{Q^{\star}_\Sigma}$, it suffices to count the maximum number of function-free rules in $[\Sigma_f]^{\star} \cup [Q, \Sigma_f]^{\star}$. Due to the linearity of TGDs, in $[\Sigma_f]^{\star}$ we can have at most $(|\mathcal{R}| \cdot (w + c_\Sigma)^w)^2 = \mathcal{O}(s)$, where c_Σ is the number of constants occurring in Σ. In $[Q, \Sigma_f]^{\star}$ we can have $(|\mathcal{R}| \cdot (c_\Sigma + c_{\rho^{\star}} + w \cdot m)^w)^m = \mathcal{O}((s+m)^m)$. Consequently, the maximal size of $\Pi_{Q^{\star}_\Sigma}$ is $\mathcal{O}(s) + \mathcal{O}((s+m)^m) = \mathcal{O}((s+m)^m)$. Since s is polynomial w.r.t. n the claim follows. □

E. EXPERIMENTAL RESULTS

As already discussed in Section 5, a Datalog query is not always more compact than a UCQs. The following simple example illustrates this fact.

Example 2. Consider the set

$$\Sigma \;=\; \begin{cases} \mathit{brother}(X,Y) \;\rightarrow\; \mathit{brother}(Y,X) \\ \mathit{brother}(X,Y) \;\rightarrow\; \exists Z\, \mathit{relatives}(X,Z), \end{cases}$$

of linear TGDs, and the BCQ $Q = \langle q, \tau \rangle$, where τ is the rule $q \leftarrow \mathit{relatives}(X,Y)$. It is not difficult to verify that the rewritten query constructed by applying our algorithm is the pair $Q_\Sigma = \langle q, \Pi_{Q_\Sigma} \rangle$, where

$$\Pi_{Q_\Sigma} \;=\; \begin{cases} \mathit{brother}(X,Y) \;\rightarrow\; \mathit{brother}(Y,X) \\ q \;\leftarrow\; \mathit{brother}(X,Y) \\ q \;\leftarrow\; \mathit{relative}(X,Y). \end{cases}$$

On the other hand, the rewritten query constructed by applying, for example, the algorithm proposed in [15], is the UCQs $Q'_\Sigma = \langle q, \Pi_{Q'_\Sigma} \rangle$, where

$$\Pi_{Q'_\Sigma} \;=\; \begin{cases} q \;\leftarrow\; \mathit{brother}(X,Y) \\ q \;\leftarrow\; \mathit{relative}(X,Y). \end{cases}$$

Clearly, $|\Pi_{Q_\Sigma}| = 3 > |\Pi_{Q'_\Sigma}| = 2$. ■

By applying the rule saturation step of our algorithm, in fact we "mimic" all the possible chase derivations. However, there are simple cases, like the one exhibited in the above example, where this is not really necessary, and thus redundant rules are generated.

An interesting problem is to identify structural properties of the given set of constraints, that can be used to determine whether a UCQs is more suitable than a Datalog query for the representation of the rewriting.