Polynomial Rewritings for Linear Existential Rules

Georg Gottlob[1] **Marco Manna**[2] **Andreas Pieris**[3]

[1]Department of Computer Science, University of Oxford, UK
[2]Department of Mathematics and Computer Science, University of Calabria, Italy
[3]Institute of Information Systems, Vienna University of Technology, Austria

georg.gottlob@cs.ox.ac.uk, manna@mat.unical.it
pieris@dbai.tuwien.ac.at

Abstract

We consider the scenario of ontology-based query answering. It is generally accepted that true scalability in this setting can only be achieved via query rewriting, which in turn allows for the exploitation of standard RDBMSs. In this work, we close two open fundamental questions related to query rewriting. We establish that linear existential rules are polynomially combined rewritable, while full linear rules are polynomially (purely) rewritable; in both cases, the target query language consists of first-order or non-recursive Datalog queries. An immediate consequence of our results is that DLR-Lite$_\mathcal{R}$, the extension of DL-Lite$_\mathcal{R}$ with n-ary roles, is polynomially combined rewritable.

1 Introduction

This paper considers the well-known setting of ontology-based query answering (OBQA), where an ontology is used to enrich the extensional dataset with intensional knowledge. A database D is combined with an ontology Σ, while an input query is answered against the logical theory $(D \cup \Sigma)$. In this setting, Description Logics (DLs) and existential rules are popular ontology languages, while conjunctive queries (CQs) is the main querying tool. Thus, efficient procedures for CQ answering under such languages are of great importance.

Among KR researchers there is a clear consensus that the required level of scalability in OBQA can only be achieved via query rewriting, which attempts to reduce OBQA into the problem of evaluating a query over a relational database. Two query languages are usually considered: *first-order queries*, which can be translated into standard SQL, and *non-recursive Datalog queries*, which can be translated into SQL with view definitions (SQL DDL). Therefore, query rewriting allows for the exploitation of existing technology provided by standard RDBMSs. The main approaches to query rewriting are the *pure approach* [Calvanese *et al.*, 2007] and the *combined approach* [Lutz *et al.*, 2009]:

Pure Approach: An ontology Σ can be combined with a CQ q into a database query q_Σ, such that, for *every* database D, q_Σ over D yields exactly the same result as q evaluated against $(D \cup \Sigma)$. This approach applies only to languages for which the data complexity of CQ answering is below PTIME; thus, useful formalisms with PTIME-hard data complexity are immediately excluded.

Combined Approach: A refined approach to query answering, which overcomes the above shortcoming, is the combined approach that allows for the encoding of the consequences of the ontology in the given database. An ontology Σ can be incorporated together with a given CQ q into a database query q_Σ, but also with a given database D into a database D_Σ, such that q_Σ over D_Σ yields the same result as q evaluated over $(D \cup \Sigma)$.

Both approaches have been applied to DLs and classes of existential rules; see, e.g., [Calvanese *et al.*, 2007; Lutz *et al.*, 2009; Kontchakov *et al.*, 2010; 2011; Calì *et al.*, 2012a; 2012b; Gottlob *et al.*, 2014b]. However, they both suffer from a crucial weakness, which may revoke the key advantage of query rewriting, i.e., the use of RDBMSs: (pure or combined) rewriting algorithms may generate from a reasonably sized CQ an exponentially sized database query, which can be prohibitive for efficient execution by an RDBMS. This naturally leads to the *polynomial* version of the pure approach and the combined approach, where the rewriting algorithms are required to terminate after polynomially many steps.

The polynomial pure approach has been applied to DL-Lite$_\mathcal{F}$ [Kontchakov *et al.*, 2010], one of the core languages of the DL-Lite family of DLs, and to unary inclusion dependencies [Kikot *et al.*, 2011], that is, a small fragment of inclusion dependencies. The polynomial combined approach has been applied to $\mathcal{ELH}^{dr}_\bot$ [Lutz *et al.*, 2009], an extension of the well-known DL $\mathcal{EL}$, to DL-Lite$^\mathcal{N}_{horn}$ [Kontchakov *et al.*, 2010; 2011], one of the most expressive logic of the DL-Lite family, and only recently to the main guarded- and sticky-based classes of existential rules [Gottlob *et al.*, 2014b]. In this work, we are mainly concerned about existential rules.

Research Challenges. The problem of applying the polynomial pure and combined approach to existing classes of existential rules is relatively understood. Nevertheless, there are still basic open questions regarding the well-known formalism of *linear (existential) rules*, that is, assertions of the form $\forall\mathbf{X}\forall\mathbf{Y}(s(\mathbf{X},\mathbf{Y}) \rightarrow \exists\mathbf{Z}\, p(\mathbf{X},\mathbf{Z}))$, where $s(\mathbf{X},\mathbf{Y})$ and $p(\mathbf{X},\mathbf{Z})$ are atomic formulas [Calì *et al.*, 2012a]. Despite their simplicity, linear rules are powerful enough for capturing prominent database dependencies, and in particular inclusion dependencies, as well as key DLs such as DL-Lite$_\mathcal{R}$ and DLR-Lite$_\mathcal{R}$, the extension of DL-Lite$_\mathcal{R}$ with n-ary roles. Ev-

idently, linear rules form a central language that deserves our attention. As stated in [Gottlob *et al.*, 2014b], a major open question is the following:

Can we apply the polynomial combined approach to the class of linear rules?

The answer to the above question is affirmative under the assumption that the arity of the underlying schema is bounded; implicit in [Gottlob *et al.*, 2014a]. However, little is known for arbitrary linear rules, without any assumption on the underlying schema. Regarding the analogous question for the polynomial pure approach, it is well-known that the answer is negative; implicit in [Gottlob *et al.*, 2014a]. Noticeably, this negative result cannot be transferred to full linear rules, i.e., linear rules without existential quantification. This gives rise to the following fundamental question:

Can we apply the polynomial pure approach to the class of full linear rules?

Our Results. It is the precise aim of the current work to give answers to the above open questions. In fact, we show that the answer to both questions is affirmative:

The class of linear existential rules is polynomially combined $\mathcal{Q}$-rewritable, while full linear existential rules are polynomially (purely) $\mathcal{Q}$-rewritable, where $\mathcal{Q}$ consists of first-order or non-recursive Datalog queries.

Since linear rules generalize DLR-Lite$_\mathcal{R}$, we immediately conclude that DLR-Lite$_\mathcal{R}$ is also polynomially combined $\mathcal{Q}$-rewritable. These are non-trivial results that required novel techniques beyond the state of the art. In fact, all the formalisms for which the polynomial pure and combined approach have been applied, share a property that is crucial for the existing rewriting techniques: given a (Boolean) CQ q, a database D, and an ontology Σ, if q is entailed by $(D \cup \Sigma)$, then q admits a proof of polynomial size, i.e., q is already entailed by a polynomially sized subinstance of the underlying canonical model of D and Σ. However, the above property does not hold for (full) linear rules [Gottlob *et al.*, 2014b]. Hence, by following the same principle as in existing techniques, i.e., to simulate the polynomially sized proof of the given query via a first-order or a non-recursive Datalog query, the obtained rewritings will unavoidably be of exponential size. Thus, our challenge is to simulate query proofs not necessarily of polynomial size via polynomially sized queries.

2 Preliminaries

Technical Definitions. Consider the following pairwise disjoint (infinite countable) sets: a set $\mathbf{C}$ of *constants*, a set $\mathbf{N}$ of *labeled nulls*, and a set $\mathbf{V}$ of *variables*. A *term* t is a constant, null, or variable. An *atom* has the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate, and $t_1, \ldots, t_n$ are terms. For an atom $\underline{a}$, we denote $dom(\underline{a})$, $null(\underline{a})$ and $var(\underline{a})$ the set of its terms, nulls, and variables, respectively; these extend to sets of atoms. Conjunctions of atoms are often identified with the sets of their atoms. An *instance* I is a (possibly infinite) set of atoms $p(\mathbf{t})$, where $\mathbf{t}$ is a tuple of constants and nulls. A *database* D is a finite instance such that $dom(D) \subset \mathbf{C}$. A *homomorphism* is a substitution $h : \mathbf{C} \cup \mathbf{N} \cup \mathbf{V} \to \mathbf{C} \cup \mathbf{N} \cup \mathbf{V}$ that is the identity on $\mathbf{C}$. We assume familiarity with *conjunctive queries (CQs)*. We write $q(I)$ for the answer to a CQ q over I. A Boolean CQ (BCQ) q has a positive answer over I, denoted $I \models q$, if $() \in q(I)$.

An *existential rule* (or simply *rule*) σ is a constant-free first-order formula $\forall \mathbf{X} \forall \mathbf{Y} (\varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, p(\mathbf{X}, \mathbf{Z}))$, where $(\mathbf{X} \cup \mathbf{Y} \cup \mathbf{Z}) \subset \mathbf{V}$, φ is a conjunction of atoms; $\varphi(\mathbf{X}, \mathbf{Y})$ is the *body* of σ, while $p(\mathbf{X}, \mathbf{Z})$ is the *head* of σ. A class of existential rules that is of special interest for the current work consists of *linear existential rules* (or simply *linear rules*), that is, rules with only one body-atom [Calì *et al.*, 2012a]. More precisely, the class of linear rules, denoted LIN, is defined as the family of all possible sets of linear rules. Given a set Σ of rules, $sch(\Sigma)$ is the set of predicates occurring in Σ. For brevity, we will omit the universal quantifiers. An instance I satisfies σ, written $I \models \sigma$, if the following holds: whenever there exists a homomorphism h such that $h(p(\mathbf{X}, \mathbf{Y})) \in I$, then there exists $h' \supseteq h_{|\mathbf{X}}$, where $h_{|\mathbf{X}}$ is the restriction of h on $\mathbf{X}$, such that $h'(p(\mathbf{X}, \mathbf{Z})) \in I$; I satisfies a set Σ of rules, denoted $I \models \Sigma$, if I satisfies each σ of Σ.

The *models* of a database D and a set Σ of rules, denoted $mods(D, \Sigma)$, is the set $\{I \mid I \supseteq D \text{ and } I \models \Sigma\}$. The *answer* to a CQ q w.r.t. D and Σ is defined as the set of tuples $ans(q, D, \Sigma) = \bigcap_{I \in mods(D,\Sigma)} \{\mathbf{t} \mid \mathbf{t} \in q(I)\}$. The answer to a BCQ q is *positive*, denoted $(D \cup \Sigma) \models q$, if $ans(q, D, \Sigma) \neq \varnothing$. The problem of ***CQ answering*** is defined as follows: given a CQ q, a database D, a set Σ of rules, and a tuple of constants $\mathbf{t}$, decide whether $\mathbf{t} \in ans(q, D, \Sigma)$. In case q is a BCQ, the above problem is called ***BCQ answering***. These decision problems are LOGSPACE-equivalent, and we thus focus only on BCQ answering. For query answering purposes, we can assume, w.l.o.g., that both the query and the database contain only predicates of $sch(\Sigma)$.

We assume familiarity with the *chase procedure*. Recall that the chase for a database D and a set Σ of rules, denoted $chase(D, \Sigma)$, is a *universal model* of D and Σ, and thus $(D \cup \Sigma) \models q$ iff $chase(D, \Sigma) \models q$, for each BCQ q. The *chase graph* for D and Σ, denoted $CG(D, \Sigma)$, is a directed labeled graph (N, E, λ), where N is the set of nodes, λ is the node labeling function $N \to chase(D, \Sigma)$, and an edge $(v, u) \in E$ iff $\lambda(u)$ is obtained from $\lambda(v)$ via a single chase step. It is easy to verify that, whenever $\Sigma \in \mathsf{LIN}$, $CG(D, \Sigma)$ is a directed forest.

Query Rewriting. For our purposes, we are going to consider two central query languages, namely first-order queries (FO), and non-recursive Datalog queries (NDL); see, e.g., [Abiteboul *et al.*, 1995]). The formal definition of polynomial combined $\mathcal{Q}$-rewritability, where $\mathcal{Q} \in \{\mathsf{FO}, \mathsf{NDL}\}$, which is at the basis of the polynomial combined approach, is as follows; in the sequel, fix a class $\mathbb{C}$ of existential rules:

Definition 1 The class $\mathbb{C}$ is *polynomially combined $\mathcal{Q}$-rewritable* if, for every BCQ q, database D, and $\Sigma \in \mathbb{C}$, we can rewrite in polynomial time: (i) q and Σ, independently of D, into $q_\Sigma \in \mathcal{Q}$, and (ii) D and Σ, independently of q, into a database D_Σ, such that, $(D \cup \Sigma) \models q$ iff $D_\Sigma \models q_\Sigma$. ∎

The rewriting of q and Σ is the query compilation phase, while the rewriting of D and Σ is the database compilation

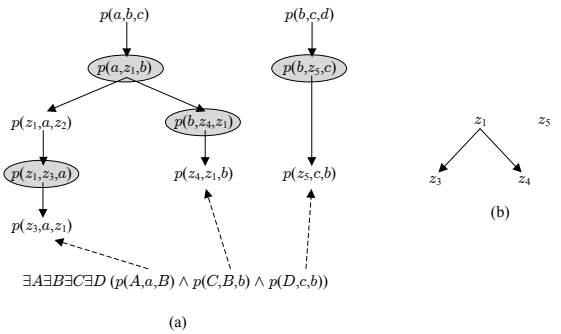

Figure 1: Illustration of a proof generator.

phase. If the database compilation phase is dropped, then we get the notion of polynomial $\mathcal{Q}$-rewritability.

Definition 2 The class $\mathbb{C}$ is *polynomially $\mathcal{Q}$-rewritable* if, for every BCQ q, and $\Sigma \in \mathbb{C}$, it is possible to rewrite in polynomial time: q and Σ into $q_\Sigma \in \mathcal{Q}$, such that, for every database D, $(D \cup \Sigma) \models q$ iff $D \models q_\Sigma$. ∎

3 Proof Generator

In this section, we introduce our main technical tool, called proof generator, which formalizes the intuitive idea that BCQ answering under linear rules can be realized as a reachability problem on the chase graph. Let us first illustrate the key ideas underlying the proof generator via a simple example.

Example 1 Let $D = \{p(a,b,c), p(b,c,d)\}$, and let Σ be the set of linear rules:

$$\underline{b} \rightarrow \exists W\, p(X,W,Y) \qquad \underline{b} \rightarrow \exists W\, p(Z,W,Y)$$
$$\underline{b} \rightarrow \exists W\, p(Y,X,W) \qquad \underline{b} \rightarrow p(Y,Z,X),$$

where $\underline{b} = p(X,Y,Z)$. Given the BCQ

$$q = \exists A \exists B \exists C \exists D\, (p(A,a,B) \wedge p(C,B,b) \wedge p(D,c,b)),$$

as illustrated in Figure 1(a), there exists a homomorphism h (dashed arrows in the figure) that maps q to an initial segment of the chase graph of D and Σ, and thus $(D \cup \Sigma) \models q$. It is interesting to observe that the nulls occurring in $h(q)$, i.e., z_1, z_3, z_4 and z_5, form a rooted forest F, depicted in Figure 1(b), with the following properties; for brevity, let $\nu(z)$ be the atom in $CG(D,\Sigma)$ where the null z is invented (see shaded atoms in Figure 1(a) for $\nu(z_1)$, $\nu(z_3)$, $\nu(z_4)$ and $\nu(z_5)$): (i) for every root node z, $\nu(z)$ is reachable from D; (ii) for every edge (z,w), $\nu(w)$ is reachable from $\nu(z)$; and (iii) for every atom $\underline{a} \in h(q)$, there exists a unique path π in F that contains all the nulls in $\underline{a}$, and, assuming that the leaf node of π is z, $\underline{a}$ is reachable from $\nu(z)$. Indeed, it is easy to verify that $\nu(z_1)$ and $\nu(z_5)$ are reachable from D, $\nu(z_3)$ and $\nu(z_4)$ are reachable from $\nu(z_1)$, and finally, $h(p(A,a,B)) = p(z_3,a,z_1)$ is reachable from $\nu(z_3)$, $h(p(C,B,b)) = p(z_4,z_1,b)$ from $\nu(z_4)$, and $h(p(D,c,b)) = p(z_5,c,b)$ from $\nu(z_5)$. ∎

Roughly speaking, the triple consisting of: (1) the homomorphism h, that maps q to the chase; (2) the function ν, that gives the atoms in the chase where the nulls occurring in $h(q)$ were invented; and (3) the forest F, that encodes how the nulls of $h(q)$ depend on each other, as well as the order of their generation, is what we call a proof generator for q w.r.t. D and Σ. The existence of such a triple, allows us to generate the part of the chase due to which the query is entailed, i.e., the proof of the query (hence the name "proof generator"). Therefore, for query answering purposes under linear rules, we simply need to check if such a proof generator exists. Let us now formalize the above intuitive ideas.

Rooted Forests. A *rooted tree* T on a set $\mathbf{S}$ is a tree with $\mathbf{S}$ be the node set of T, where one node, denoted $root(T)$, has been designated the root, and the edges have an orientation away from the root. The *tree-order* of T is the (non-strict) partial order $\preceq_T$ over $\mathbf{S}$ such that $v \preceq_T u$ iff the unique path from the root to u passes through v. The corresponding strict partial order is given by: $v \prec_T u$ iff $v \preceq_T u$ and $v \neq u$. A *rooted forest* F on a set $\mathbf{S}$ is a collection of rooted trees $T_1, \ldots, T_n$ on $\mathbf{S}_1, \ldots, \mathbf{S}_n$, resp., where $\mathbf{S}_i \cap \mathbf{S}_j = \varnothing$, for $1 \leqslant i < j \leqslant n$, and $\mathbf{S} = \bigcup_{i \in [n]} \mathbf{S}_i$. We define $root(F) = \bigcup_{i \in [n]} \{root(T_i)\}$. The *forest-order* of F is the partial order $\preceq_F = \bigcup_{i \in [n]} \preceq_{T_i}$, while $v \prec_F u$ iff $v \preceq_F u$ and $v \neq u$. Consider a set $\mathbf{T} \subseteq \mathbf{S}$ such that, for each $v, u \in \mathbf{T}$, $v \preceq_F u$ or $u \preceq_F v$. It can be verified that there exists a unique $v \in \mathbf{T}$, denoted $greatest_F(\mathbf{T})$, such that $u \prec_F v$, for each $u \in \mathbf{T}$.

Query Answering and Proof Generators. For technical clarity, we focus, w.l.o.g., on linear rules in normal form, i.e., linear rules with at most one occurrence of an existentially quantified variable; see, e.g., [Gottlob *et al.*, 2014b]. Fix a BCQ q, a database D, and a set $\Sigma \in \mathsf{LIN}$. As said above, a proof generator for q w.r.t. D and Σ is a triple consisting of a homomorphism $h : var(q) \rightarrow (dom(D) \cup \mathbf{N})$, a function $\nu : null(h(q)) \rightarrow chase(D,\Sigma)$, and a rooted forest F on $null(h(q))$. Such a triple, in order to be a valid candidate for a proof generator, must satisfy certain properties. In particular, (1) the atoms of $h(q)$, as well as the atoms where the nulls of $h(q)$ were invented, do not contain incomparable (w.r.t. $\preceq_F$) nulls of $h(q)$, and (2) for each $z \in null(h(q))$, z is the most recently generated null among the nulls in $\nu(z)$. The above properties are formalized via the proof generator scheme. In the sequel, for brevity, let $\mathbf{U} = null(h(q))$.

Definition 3 A *proof generator scheme* for q w.r.t.D and Σ is a triple (h, ν, F), with $h : var(q) \rightarrow (dom(D) \cup \mathbf{N})$ be a mapping, $\nu : null(h(q)) \rightarrow chase(D,\Sigma)$ be a (total) function, and F be a rooted forest on $null(h(q))$, such that:

1. For each $\underline{a} \in (h(q) \cup \{\nu(z) \mid z \in \mathbf{U}\})$, and for each $z, w \in (null(\underline{a}) \cap \mathbf{U})$, $z \preceq_F w$ or $w \preceq_F z$; and
2. For each $z \in \mathbf{U}$, $z = greatest_F(null(\nu(z)) \cap \mathbf{U})$. ∎

We are now ready to define the notion of the proof generator. Roughly, a proof generator is a proof generator scheme that gives rise to a proof of the given query, that is, a (finite) part of the chase due to which the query is entailed. The latter can be easily verified by applying some reachability checks on the chase graph. A path $\pi = v_1 \ldots v_n$, where $n > 1$, in $CG(D,\Sigma) = (N,E,\lambda)$ is called *generating* for a null z if, $z \notin null(\lambda(v_i))$, for each $i \in [n-1]$, and $z \in null(\lambda(v_n))$. Given two (distinct) atoms $\underline{a}, \underline{b} \in chase(D,\Sigma)$, we write $\underline{a} \rightsquigarrow \underline{b}$ for the fact that $\underline{b}$ is reachable from $\underline{a}$ in $CG(D,\Sigma)$. Furthermore, given a null z occurring in $chase(D,\Sigma)$, we write $\underline{a} \rightsquigarrow_z \underline{b}$ for the fact that $\underline{b}$ is reachable from $\underline{a}$ in $CG(D,\Sigma)$ via a path that is generating for z.

Definition 4 A *proof generator* for q w.r.t. D and Σ is a proof generator scheme (h, ν, F) such that:

1. For each $z \in root(F)$, there exists $\underline{a} \in D$ such that $\underline{a} \rightsquigarrow_z \nu(z)$;
2. For each edge (z, w) of F, $\nu(z) \rightsquigarrow_w \nu(w)$;
3. For each $\underline{a} \in h(q)$ with $z = greatest_F(null(\underline{a}))$, either $\underline{a} = \nu(z)$ or $\nu(z) \rightsquigarrow \underline{a}$; and
4. For each $\underline{a} \in h(q)$ with $null(\underline{a}) = \varnothing$, there exists $\underline{b} \in D$ such that $\underline{b} \rightsquigarrow \underline{a}$. ∎

Our main technical lemma follows:

Lemma 5 *$(D \cup \Sigma) \models q$ iff there exists a proof generator for q w.r.t. D and Σ.*

4 Linear Existential Rules

In this section, we give a positive answer to our first research question regarding linear rules and the polynomial combined approach. More precisely, we show that:

Theorem 6 LIN *is polynomially combined $\mathcal{Q}$-rewritable, where $\mathcal{Q} \in \{\mathsf{FO}, \mathsf{NDL}\}$.*

To establish the above theorem, we heavily rely on the notion of the proof generator. For the rest of this section, fix a BCQ q, a database D, and a set $\Sigma \in \mathsf{LIN}$. By Lemma 5, it suffices to construct in polynomial time a query $q_\Sigma \in \mathcal{Q}$ (independently of D), and a database D_Σ (independently of q), such that $D_\Sigma \models q_\Sigma$ iff a proof generator for q w.r.t. D and Σ exists. Roughly, the query q_Σ consists of three components: the first one guesses a triple (h, ν, F) as in Definition 3, the second component verifies that the guessed triple is a proof generator scheme, and finally, the third component verifies that the guessed triple is a proof generator for q w.r.t. D and Σ. The interesting part of q_Σ is its third component, which applies the crucial reachability checks required by Definition 4. Although the path among two atoms in the chase graph may be of exponential size, its existence can be checked via $\mathcal{Q}$-queries of polynomial size.

Let us make some assumptions that will simplify our task. Recall that Σ is in normal form, i.e., each rule has at most one occurrence of an existentially quantified variable. We further assume that the existentially quantified variable appears in the last position of the head-atom. Moreover, we assume that Σ contains only one predicate p of arity ω. As shown in [Gottlob *et al.*, 2014b], the above assumptions can be made without affecting the generality of our proof. We finally assume that q is of the form $\exists \mathbf{Z}(p(\mathbf{t}_1) \wedge \ldots \wedge p(\mathbf{t}_n))$, where $\mathbf{Z} = Z_1, \ldots, Z_\ell$, and $\mathbf{t}_i \in (dom(D) \cup \mathbf{Z})^\omega$, for each $i \in [n]$.

4.1 First-Order Rewriting

We first focus on the case where $\mathcal{Q} = \mathsf{FO}$.

Database Compilation Phase. We define

$$D_\Sigma \;=\; D \cup \{bit(0), bit(1), star(\star)\},$$

where bit and $star$ are auxiliary predicates not occurring in $sch(\Sigma)$, while 0, 1 and $\star$ are fresh constants not in D. The purpose of 0 and 1 is, intuitively speaking, to encode in binary form the nulls that appear in the proof of the given query q. Note that those nulls cannot be explicitly encoded in D_Σ, since in this case D_Σ will depend on q, and such a rewriting will not be consistent with Definition 1. The purpose of $\star$ is to help us to indicate that a path in the chase graph does not need to be generating for some null. Clearly, D_Σ does not depend on q, and it can be constructed on $\mathcal{O}(1)$ time.

Query Compilation Phase. Let us now proceed with the definition of q_Σ. As said above, q_Σ consists of three components: guess a triple (h, ν, F), verify that is a proof generator scheme, and finally, verify that is a proof generator. Before giving the general shape of q_Σ, let us introduce the variables, and their underlying meaning, that we will use in q_Σ — assume, for the moment, that α is a sufficiently large integer such that all the nulls that may occur in the proof of q can be represented via tuples of the form $\{0, 1\}^\alpha$:

- For each $i \in [\ell]$, $\mathbf{Z}_i = Z_i^1, ..., Z_i^\alpha$ represents the term (constant or null) to which the query variable Z_i is mapped to via the homomorphism h. Notice that such a term is encoded as a tuple of length α;
- For each $i \in [\ell]$, $\mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega$, where each $\mathbf{T}_i^j$ is an α-tuple, represents an atom in the chase where a null occurring in $h(q)$ was invented. In other words, assuming that $\mathbf{T}_i^\omega$ encodes the null z (the set Σ is in normal form, and thus an invented null appears at the last position), then $\mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega$ represents the atom $\nu(z)$; and
- For each $i \in [\ell - 1]$, the pair $(\mathbf{A}_i, \mathbf{B}_i)$ represents an edge in the rooted forest F on $null(h(q))$; for brevity, let $\mathcal{A} = \{\mathbf{A}_1, \ldots, \mathbf{A}_{\ell-1}\}$ and $\mathcal{B} = \{\mathbf{B}_1, \ldots, \mathbf{B}_{\ell-1}\}$

We define $\mathbf{Q}$ as the set that collects all the variables introduced above. Let us now comment on the key value α. It is well-known that, for query answering purposes under linear rules, it suffices to focus on an initial segment of the chase graph of depth $\delta = (n + 1) \cdot (2\omega)^\omega$ [Calì *et al.*, 2012a]. This implies that the maximum number of atoms that can appear in the proof of q is $(n \cdot \delta)$, which in turn implies that the number of nulls in the proof of q is bounded by $(n \cdot \delta \cdot \omega)$. Thus, it suffices to let $\alpha = \lceil \log(n \cdot \delta \cdot \omega) \rceil$. It is important to say that α is polynomial w.r.t. q and Σ, and independent of D.

The general shape of q_Σ is as follows:

$$\exists \mathbf{Q} \;(GuessTriple(\mathbf{Q}) \wedge ProofGeneratorScheme(\mathbf{Q}) \wedge ProofGenerator(\mathbf{Q}))\,.$$

With the aim of simplifying the definition of q_Σ, we make use of some useful shortcuts, given in Table 1. The employed shortcuts are self-explanatory, and thus we proceed with q_Σ, without discussing their semantic meaning.

The real purpose of the subquery $GuessTriple(\mathbf{Q})$ is to ensure that the guessed triple (h, ν, F) is valid, which means that $\nu : null(h(q)) \to chase(D, \Sigma)$ is indeed a (total) function, and F is a rooted forest on $null(h(q))$. This can be easily done via a simple first-order query. In what follows, we present the formal definition of $ProofGeneratorScheme(\mathbf{Q})$ and $ProofGenerator(\mathbf{Q})$.

Proof Generator Scheme. We check whether the guessed triple is indeed a proof generator scheme. In fact, we simply need to check if the two conditions in Definition 3 hold. But

Shortcut	Definition
$\mathbf{X} = \mathbf{Y}$	$\bigwedge_{i \in [\lvert \mathbf{X} \rvert]} (\mathbf{X}[i] = \mathbf{Y}[i])$
$null(\mathbf{X})$	$\bigwedge_{X \in \mathbf{X}} bit(X)$
$qNull(\mathbf{X})$	$null(\mathbf{X}) \wedge \bigvee_{i \in [\ell]} (\mathbf{X} = \mathbf{Z}_i)$
$dom(X)$	$\exists \mathbf{Y} (p(\mathbf{Y}) \wedge \bigvee_{Y \in \mathbf{Y}} (X = Y))$
$const(\mathbf{X})$	$\exists Y \left(dom(Y) \wedge \bigwedge_{X \in \mathbf{X}} (X = Y) \right)$
$sstar(\mathbf{X})$	$\bigwedge_{X \in \mathbf{X}} star(X)$
$\mathbf{X} \in \mathcal{T}$	$\bigvee_{\mathbf{Y} \in \mathcal{T}} (\mathbf{X} = \mathbf{Y})$
$\mathbf{X} \prec_1 \mathbf{Y}$	$\neg(\mathbf{X} = \mathbf{Y}) \wedge \bigvee_{i \in [\ell - 1]} ((\mathbf{X} = \mathbf{A}_i) \wedge (\mathbf{Y} = \mathbf{B}_i))$
$\mathbf{X} \prec_{i+1} \mathbf{Y}$	$\exists \mathbf{W} ((\mathbf{X} \prec_i \mathbf{W}) \wedge (\mathbf{W} \prec_1 \mathbf{Y}))$
$\mathbf{X} \prec \mathbf{Y}$	$\bigvee_{i \in [\ell - 1]} (\mathbf{X} \prec_i \mathbf{Y})$
$min(\mathbf{X})$	$qNull(\mathbf{X}) \wedge (\mathbf{X} \in \mathcal{A}) \wedge \neg(\mathbf{X} \in \mathcal{B})$
$sup(\mathbf{X}, \mathcal{T})$	$qNull(\mathbf{X}) \wedge (\mathbf{X} \in \mathcal{T}) \wedge$ $\bigwedge_{\mathbf{Y} \in \mathcal{T}} (qNull(\mathbf{Y}) \rightarrow ((\mathbf{X} = \mathbf{Y}) \vee (\mathbf{Y} \prec \mathbf{X})))$

Table 1: $\mathbf{X}, \mathbf{Y}$ are tuples of variables, while $\mathcal{T}$ is a sequence of tuples of variables. $\mathbf{Z}_i$, $\mathbf{A}_i$ and $\mathbf{B}_i$ are variables of $\mathbf{Q}$. Recall that p is the unique predicate that occurs in Σ.

let us first introduce some auxiliary notation. For each $i \in [n]$, $\mathbf{tt}_i$ is defined as the tuple $f(\mathbf{t}_i[1]), \ldots, f(\mathbf{t}_i[w])$, where $f(t) = (t)^\alpha$, if $t \in \mathbf{C}$, and $f(t) = t^1, \ldots, t^\alpha$, if $t \in \mathbf{Z}$; recall that $p(\mathbf{t}_i)$ is an atom of q. Moreover, for each $j \in [\omega]$, $\mathbf{tt}_i^j = f(\mathbf{t}_i[j])$. For example, if $\omega = 3$, $\mathbf{t}_i = c, Z_3, Z_1$, and $\alpha = 2$, then $\mathbf{tt}_i = c, c, Z_3^1, Z_3^2, Z_1^1, Z_1^2$, while $\mathbf{tt}_i^1 = c, c$, $\mathbf{tt}_i^2 = Z_3^1, Z_3^2$ and $\mathbf{tt}_i^3 = Z_1^1, Z_1^2$.

The first condition in Definition 3, namely *"for each $\underline{a} \in (h(q) \cup \{\nu(z) \mid z \in \mathbf{U}\})$, and for each $z, w \in (null(\underline{a}) \cap \mathbf{U})$, $z \preceq_F w$ or $w \preceq_F z$"*, can be verified via $PGS_1(\mathbf{Q})$:

$$\bigwedge_{i \in [n]} \bigwedge_{j,k \in [\omega]} \Big(\Big(qNull(\mathbf{tt}_i^j) \wedge qNull(\mathbf{tt}_i^k) \Big) \rightarrow \Big((\mathbf{tt}_i^j = \mathbf{tt}_i^k) \vee (\mathbf{tt}_i^j \prec \mathbf{tt}_i^k) \vee (\mathbf{tt}_i^k \prec \mathbf{tt}_i^j) \Big) \Big) \wedge$$
$$\bigwedge_{i \in [\ell]} \bigwedge_{j,k \in [\omega]} \Big(\Big(qNull(\mathbf{T}_i^j) \wedge qNull(\mathbf{T}_i^k) \Big) \rightarrow \Big((\mathbf{T}_i^j = \mathbf{T}_i^k) \vee (\mathbf{T}_i^j \prec \mathbf{T}_i^k) \vee (\mathbf{T}_i^k \prec \mathbf{T}_i^j) \Big) \Big) .$$

The second condition, which asks that *"for each $z \in \mathbf{U}$, $z = greatest_F(null(\nu(z)) \cap \mathbf{U})$"*, can be checked via $PGS_2(\mathbf{Q})$:

$$\bigwedge_{i \in [\ell]} \left(qNull(\mathbf{T}_i^\omega) \rightarrow sup(\mathbf{T}_i^\omega, \mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega) \right) .$$

Notice that $PGS_2(\mathbf{Q})$ assumes the following: for each tuple $\mathbf{b}$ of bits in the image of $\mathbf{Z}_1, \ldots, \mathbf{Z}_\ell$, there exists $i \in [\ell]$ such that $\mathbf{T}_i^\omega$ is mapped to $\mathbf{b}$. This is enforced by the subquery of $GuessTriple(\mathbf{Q})$, which checks whether $\nu : null(h(q)) \rightarrow chase(D, \Sigma)$ is a total function. Consequently, the query $ProofGeneratorScheme(\mathbf{Q})$ is $(PGS_1(\mathbf{Q}) \wedge PGS_2(\mathbf{Q}))$.

Proof Generator. We proceed to check if the guessed triple is a proof generator for q w.r.t. D and Σ, or whether the four conditions in Definition 4 are fulfilled. Let us assume, for the moment, that we have access to the subqueries π and π_G. Intuitively, $\pi(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X}$ and $\mathcal{Y}$ are collections of tuples of variables representing some tuples in the chase, let say $\mathbf{t}_\mathcal{X}$ and $\mathbf{t}_\mathcal{Y}$, respectively, means that $p(\mathbf{t}_\mathcal{X}) \rightsquigarrow p(\mathbf{t}_\mathcal{Y})$. Analogously, $\pi_G(\mathcal{X}, \mathcal{Y})$, assuming that $\mathcal{Y}[\omega]$ represents the null z, is equivalent to $p(\mathbf{t}_\mathcal{X}) \rightsquigarrow_z p(\mathbf{t}_\mathcal{Y})$. Both π and π_G are formally defined below.

The first condition in Definition 4, that is, *"for each $z \in root(F)$, there exists $\underline{a} \in D$ such that $\underline{a} \rightsquigarrow_z \nu(z)$"*, can be verified via the query $PG_1(\mathbf{Q})$, which is defined as:

$$\bigwedge_{i \in [\ell]} \left(min(\mathbf{T}_i^\omega) \rightarrow \exists S_1 \ldots \exists S_\omega \left(p(S_1, \ldots, S_\omega) \wedge \pi_G((S_1)^\alpha, \ldots, (S_\omega)^\alpha, \mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega) \right) \right) .$$

The second condition, namely *"for each edge (z, w) of F, $\nu(z) \rightsquigarrow_w \nu(w)$"*, can be checked via the query $PG_2(\mathbf{Q})$:

$$\bigwedge_{i,j \in [\ell]} \left((qNull(\mathbf{T}_i^\omega) \wedge qNull(\mathbf{T}_j^\omega) \wedge (\mathbf{T}_i^\omega \prec_1 \mathbf{T}_j^\omega)) \rightarrow \pi_G(\mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega, \mathbf{T}_j^1, \ldots, \mathbf{T}_j^\omega) \right) .$$

The third condition, that is, *"for each $\underline{a} \in h(q)$ with $z = greatest_F(null(\underline{a}))$, either $\underline{a} = \nu(z)$ or $\nu(z) \rightsquigarrow \underline{a}$"*, can be verified by the query $PG_3(\mathbf{Q})$:

$$\bigwedge_{i \in [n]} \bigwedge_{j \in [\ell]} \left(sup(\mathbf{T}_j^\omega, \mathbf{tt}_i) \rightarrow \pi(\mathbf{T}_j^1, \ldots, \mathbf{T}_j^\omega, \mathbf{tt}_i) \right) .$$

Finally, the fourth condition, namely *"for each $\underline{a} \in h(q)$ with $null(\underline{a}) = \varnothing$, there exists $\underline{b} \in D$ such that $\underline{b} \rightsquigarrow \underline{a}$"*, can be checked via the query $PG_4(\mathbf{Q})$:

$$\bigwedge_{i \in [n]} \left(\left(\bigwedge_{j \in [\omega]} const(\mathbf{tt}_i^j) \right) \rightarrow \exists S_1 \ldots \exists S_\omega \left(p(S_1, \ldots, S_\omega) \wedge \pi((S_1)^\alpha, \ldots, (S_\omega)^\alpha, \mathbf{tt}_i) \right) \right) .$$

Consequently, $ProofGenerator(\mathbf{Q})$ is defined as $(PG_1(\mathbf{Q}) \wedge PG_2(\mathbf{Q}) \wedge PG_3(\mathbf{Q}) \wedge PG_4(\mathbf{Q}))$. To conclude the definition of q_Σ, it remains to define the crucial subqueries π and π_G.

The Subqueries π and π_G. As said above, $\pi(\mathcal{X}, \mathcal{Y})$ is equivalent to $p(\mathbf{t}_\mathcal{X}) \rightsquigarrow p(\mathbf{t}_\mathcal{Y})$, where $\mathbf{t}_\mathcal{X}$ and $\mathbf{t}_\mathcal{Y}$ are tuples in the chase represented by $\mathcal{X}$ and $\mathcal{Y}$, while $\pi_G(\mathcal{X}, \mathcal{Y})$ means that $p(\mathbf{t}_\mathcal{X}) \rightsquigarrow_z p(\mathbf{t}_\mathcal{Y})$, where $\mathcal{Y}[\omega]$ represents the null z. Assume that we have access to an auxiliary subquery $\pi_i(\mathcal{X}, \mathcal{Y}, \mathbf{T})$, which states the following: if $\mathbf{T} = (\star)^\alpha$, then $p(\mathbf{t}_\mathcal{Y})$ is reachable from $p(\mathbf{t}_\mathcal{X})$ in $CG(D, \Sigma)$ via some path of length at most 2^i; otherwise, if $\mathbf{T} = \mathcal{Y}[\omega]$, then $p(\mathbf{t}_\mathcal{Y})$ is reachable from $p(\mathbf{t}_\mathcal{X})$ via a path of length at most 2^i that is also generating for z. Then, the crucial subqueries can be defined as

$$\begin{aligned} \pi(\mathcal{X}, \mathcal{Y}) &\equiv \pi_\lambda(\mathcal{X}, \mathcal{Y}, \mathbf{T}) \wedge sstar(\mathbf{T}) \\ \pi_G(\mathcal{X}, \mathcal{Y}) &\equiv \pi_\lambda(\mathcal{X}, \mathcal{Y}, \mathbf{T}) \wedge (\mathbf{T} = \mathcal{Y}[\omega]), \end{aligned}$$

where 2^λ is the maximum size of a path among two atoms in the proof of q. Recall that, for query answering purposes under linear rules, it suffices to focus on an initial segment of the chase graph of depth $\delta = (n + 1) \cdot (2\omega)^\omega$ [Calì *et al.*, 2012a]. Therefore, $2^\lambda \leqslant \delta$; hence, for our purposes, it suffices to set $\lambda = \lceil \log \delta \rceil$. It is crucial for our construction that λ is polynomial w.r.t. q and Σ, and independent of D.

Let us now proceed with the formal definition of π_i. To this end, we need an effective way to check if a tuple is obtained from some other tuple during the chase by applying a certain linear rule. This is achieved via the subquery $\widehat{\sigma}(\mathcal{X}, \mathbf{T})$, where $\mathcal{X}$ is a sequence of tuples (of variables) of length 2ω, which has the following intuitive meaning: if $\mathbf{T} = (\star)^\alpha$, then the tuple in the chase graph represented by $\mathcal{X}[\omega+1], \ldots, \mathcal{X}[2\omega]$ can be obtained from $\mathcal{X}[1], \ldots, \mathcal{X}[\omega]$ by applying σ; otherwise, if $\mathbf{T} = \mathcal{X}[2\omega]$, then in addition we need that the null represented by $\mathbf{T}$ is invented in the generated atom — the formal definition of $\widehat{\sigma}$ follows.

Assume that σ is a linear rule of the form $p(t_1, \ldots, t_\omega) \rightarrow p(t_{\omega+1}, \ldots, t_{2\omega})$. We define the function $\xi : [2\omega] \rightarrow [2\omega]$ in such a way that, for each $i \in [2\omega]$: if $t_i \notin \{t_{i+1}, \ldots, t_{2\omega}\}$, then $\xi(i) = i$; otherwise, $\xi(i)$ is the smallest $j > i$ such that $t_i = t_j$ and $t_i \notin \{t_{i+1}, \ldots, t_{j-1}\}$. In words, given the position i of a variable V occurring in σ, $\xi(i)$ is the next position (from left-to-right) where the same variable V (if any) occurs in σ. For example, if σ is the rule $p(X_1, X_1, X_2) \rightarrow p(X_2, X_1, X_3)$, we have that $\xi(1) = 2$, $\xi(2) = 5$, $\xi(3) = 4$, $\xi(4) = 4$, $\xi(5) = 5$, $\xi(6) = 6$.

We proceed by considering the following two cases where σ contains or not an existentially quantified variable:

- If σ contains an existentially quantified variable, which implies $\{t_{\omega+1}, \ldots, t_{2\omega-1}\} \subseteq \{t_1, \ldots, t_\omega\}$ and $t_{2\omega} \notin \{t_1, \ldots, t_\omega\}$, then $\widehat{\sigma}(\mathcal{X}, \mathbf{T})$ is defined as
$$null(\mathcal{X}[2\omega]) \wedge (sstar(\mathbf{T}) \vee (\mathbf{T} = \mathcal{X}[2\omega])) \wedge \bigwedge_{i \in [2\omega-1]} (\mathcal{X}[i] = \mathcal{X}[\xi(i)]) \wedge \bigwedge_{i \in [w]} (\mathcal{X}[i] \neq \mathcal{X}[2\omega]).$$
- If σ does not contain an existentially quantified variable, then $\widehat{\sigma}(\mathcal{X}, \mathbf{T})$ is defined as
$$sstar(\mathbf{T}) \wedge \bigwedge_{i \in [2\omega-1]} (\mathcal{X}[i] = \mathcal{X}[\xi(i)]).$$

We have now all the ingredients to define π_i. This is done inductively as follows: $\pi_0(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ is defined as

$$\bigwedge_{\mathcal{W} \in \{\mathcal{X}, \mathcal{Y}\}} \bigwedge_{\mathbf{W} \in \mathcal{W}} (const(\mathbf{W}) \vee null(\mathbf{W})) \wedge \left((\mathcal{X} = \mathcal{Y}) \vee \bigvee_{\sigma \in \Sigma} \widehat{\sigma}(\mathcal{X}\mathcal{Y}, \mathbf{T}) \right),$$

while $\pi_{i+1}(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ is defined as (calligraphic letters, as usual, denote sequences of α-tuples of variables)

$$\exists \mathcal{Z} \left(\bigwedge_{i \in [\omega]} (\mathcal{X}[i] \in \mathcal{Y} \rightarrow \mathcal{X}[i] \in \mathcal{Z}) \wedge \forall \mathcal{U} \forall \mathcal{V} \forall \mathbf{W} \left((((\mathcal{U} = \mathcal{X}) \wedge (\mathcal{V} = \mathcal{Z}) \wedge sstar(\mathbf{W})) \vee ((\mathcal{U} = \mathcal{Z}) \wedge (\mathcal{V} = \mathcal{Y}) \wedge (\mathbf{W} = \mathbf{T}))) \rightarrow \pi_i(\mathcal{U}, \mathcal{V}, \mathbf{W}) \right) \right).$$

One may claim that the definition of $\pi_{i+1}(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ can be simplified by exploiting the query $\widehat{\pi}_{i+1}(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ defined as

$$\exists \mathcal{Z} \exists \mathbf{W} \left(\bigwedge_{i \in [\omega]} (\mathcal{X}[i] \in \mathcal{Y} \rightarrow \mathcal{X}[i] \in \mathcal{Z}) \wedge \widehat{\pi}_i(\mathcal{X}, \mathcal{Z}, \mathbf{W}) \wedge sstar(\mathbf{W}) \wedge \widehat{\pi}_i(\mathcal{Z}, \mathcal{Y}, \mathbf{T}) \right),$$

with $\widehat{\pi}_0(\mathcal{X}, \mathcal{Y}, \mathbf{T}) = \pi_0(\mathcal{X}, \mathcal{Y}, \mathbf{T})$. However, in this case, the size of $\widehat{\pi}_\lambda$ will be exponential in λ, and thus exponential in ω. The construction of q_Σ is now complete, and the next key result can be established:

Proposition 7 *The following hold:*

1. *D_Σ is constructible in $\mathcal{O}(1)$ time, independently of q;*
2. *$q_\Sigma \in \mathsf{FO}$, and is constructible in polynomial time, independently of D; and*
3. *$D_\Sigma \models q_\Sigma$ iff there exists a proof generator for q w.r.t. D and Σ.*

Note that the actual size of q_Σ is $\mathcal{O}(|\Sigma|\alpha^2 n^5 \omega^5)$. Clearly, Proposition 7 and Lemma 5 imply Theorem 6 when $\mathcal{Q} = \mathsf{FO}$.

4.2 Non-Recursive Datalog Rewriting

Let us now focus on the case where $\mathcal{Q} = \mathsf{NDL}$. To avoid notational clutter, in what follows we will refer to the database D_Σ and the FO-query q_Σ defined above by D_Σ^{FO} and q_Σ^{FO}, respectively. Starting from q_Σ^{FO}, we can construct in polynomial time (independently of D) a query $q_\Sigma \in \mathsf{NDL}$ such that $D_\Sigma^{FO} \models q_\Sigma^{FO}$ iff $D_\Sigma \models q_\Sigma$, where D_Σ is defined as

$$D_\Sigma^{FO} \cup \{neq(c, d) \mid c, d \in dom(D_\Sigma^{FO}) \text{ and } c \neq d\},$$

which in turn implies Theorem 6 when $\mathcal{Q} = \mathsf{NDL}$. Let us briefly explain how q_Σ is constructed.

It is interesting to observe that q_Σ^{FO} can "almost" be transformed into a positive existential first-order query. To achieve this we need to overcome two difficulties: (1) $\pi(\mathcal{X}, \mathcal{Y})$ and $\pi_G(\mathcal{X}, \mathcal{Y})$ are defined via a formula that involves universal quantification; and (2) q_Σ^{FO} uses negation. To overcome difficulty (1), let us assume, for the moment, that we have access to the auxiliary predicates R and R_G (R stands for reachable) that store all the pairs $(\mathcal{X}, \mathcal{Y})$ that make the subqueries $\pi(\mathcal{X}, \mathcal{Y})$ and $\pi_G(\mathcal{X}, \mathcal{Y})$, respectively, true. Then, in q_Σ^{FO}, we replace $\pi(\mathcal{X}, \mathcal{Y})$ and $\pi_G(\mathcal{X}, \mathcal{Y})$ with the atomic formula $\mathsf{R}(\mathcal{X}, \mathcal{Y})$ and $\mathsf{R}_G(\mathcal{X}, \mathcal{Y})$, respectively; this gives us the query $[q_\Sigma^{FO}]_\mathsf{R}$. It is easy to verify that $[q_\Sigma^{FO}]_\mathsf{R}$ can be transformed in polynomial time into negation normal form, where each negated atom is of the form $\neg(X = Y)$ or $\neg bit(X)$. Since D_Σ gives us access to the relation neq, we can replace $\neg(X = Y)$ with $neq(X, Y)$. Moreover, we can replace $\neg bit(X)$ with the equivalent formula $(dom(X) \vee star(X))$. After applying the above replacements, we obtain the positive existential query $[q_\Sigma^{FO}]_\mathsf{R}^+$.

It is well-known that each positive existential query can be effectively rewritten as a non-recursive Datalog query. Thus, $[q_\Sigma^{FO}]_\mathsf{R}^+$ can be transformed in polynomial time into an NDL-query $[q_\Sigma]_\mathsf{R}$. In order to obtained the desired NDL-query q_Σ, it remains to show that the crucial predicates R and R_G can be defined via a non-recursive Datalog query of polynomial size. Assuming that we have access to the predicate R_λ that holds all the triples $(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ that make the subquery $\pi_\lambda(\mathcal{X}, \mathcal{Y}, \mathbf{T})$ true, the desired predicates are defined via the rules

$$\begin{array}{rcl} \mathsf{R}_\lambda(\mathcal{X}, \mathcal{Y}, \mathbf{W}), sstar(\mathbf{W}) & \rightarrow & \mathsf{R}(\mathcal{X}, \mathcal{Y}) \\ \mathsf{R}_\lambda(\mathcal{X}, \mathcal{Y}, \mathcal{Y}[\omega]) & \rightarrow & \mathsf{R}_G(\mathcal{X}, \mathcal{Y}). \end{array}$$

Finally, the predicate R_λ can be defined via the polynomially sized NDL-query obtained after transforming the "almost"

positive existential FO-query $\widehat{\pi}_\lambda$ into an exponentially more succinct non-recursive Datalog query — recall that $\widehat{\pi}_\lambda$ is the naive implementation of π_λ via a query of exponential size (given at the end of Section 4.1).

It is interesting to observe that, when the target query language is FO, the database compilation phase is feasible in constant time, while for NDL rewritings it takes polynomial time. This is because FO-queries are powerful enough to express inequalities, which is not the case for NDL-queries.

5 Dropping Existential Quantifiers

We proceed to give a positive answer to our second research question regarding the class of full linear rules and the polynomial pure approach. Recall that full linear rules are linear rules without existential quantification, and the corresponding class is denoted FLIN. We show that:

Theorem 8 FLIN *is polynomially $\mathcal{Q}$-rewritable, where $\mathcal{Q} \in \{\mathsf{FO}, \mathsf{NDL}\}$.*

The above result can be shown via simplified versions of the rewriting procedures presented above. Fix a BCQ q, and a set $\Sigma \in \mathsf{FLIN}$, under the same assumptions as in Section 4. We first focus on the case where $\mathcal{Q} = \mathsf{FO}$, and we are going to construct an FO-query q_Σ^{FO}. Since Σ is full, for every database D, $chase(D, \Sigma)$ does not contain null values. Therefore, q_Σ^{FO} can be obtained from the FO-query q_Σ defined in Section 4.1, by keeping only $PG_4(\mathbf{Q})$, which checks if *"for each $\underline{a} \in h(q)$ with $null(\underline{a}) = \varnothing$, there exists $\underline{b} \in D$ such that $\underline{b} \rightsquigarrow \underline{a}$"*. Since there are no nulls in the chase, we do not need anymore to represent the terms in the chase via α-tuples. Thus, $\mathbf{Q} = Z_1, \ldots, Z_\ell$, and q_Σ^{FO} is defined as

$$\exists \mathbf{Q} \left(\bigwedge_{i \in [n]} \left(\exists \mathbf{S} \left(p(\mathbf{S}) \wedge \pi_\lambda(\mathbf{S}, \mathbf{t}_i) \right) \right) \right),$$

where $\lambda = \lceil \log \omega^\omega \rceil$. The crucial subquery π_λ is defined inductively as follows: $\pi_0(\mathbf{X}, \mathbf{Y})$ is

$$\bigwedge_{\mathbf{W} \in \{\mathbf{X}, \mathbf{Y}\}, i \in [\omega]} dom(\mathbf{W}[i]) \wedge \left((\mathbf{X} = \mathbf{Y}) \vee \bigvee_{\sigma \in \Sigma} \widehat{\sigma}(\mathbf{XY}) \right),$$

while $\pi_{i+1}(\mathbf{X}, \mathbf{Y})$ is

$$\exists \mathbf{Z} \left(\forall \mathbf{U} \forall \mathbf{V} \left(((((\mathbf{U} = \mathbf{X}) \wedge (\mathbf{V} = \mathbf{Z})) \vee ((\mathbf{U} = \mathbf{Z}) \wedge (\mathbf{V} = \mathbf{Y}))) \rightarrow \pi_i(\mathbf{U}, \mathbf{V}) \right) \right).$$

The actual size of the obtained query is $\mathcal{O}(|\Sigma| n \omega^2 \log \omega)$. Having the first-order query q_Σ^{FO} in place, it is easy to construct an equivalent NDL-query q_Σ^{NDL}, in the same way as discussed in Section 4.2, and Theorem 8 follows.

6 Discussion and Future Work

The results of this work are, for the moment, of theoretical nature and we do not claim that they will directly lead to better practical algorithms. We believe that a smart implementation of the proposed techniques can lead to more efficient rewriting procedures; this will be the subject of future research. We are also planning to optimize the proposed rewriting algorithms, with the aim of constructing queries of optimal size. For example, after a more refined analysis, one can show that the bound α can be reduced from polynomial to logarithmic.

Acknowledgements. G. Gottlob was supported by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems – Principles and Architecture". M. Manna was supported by the MIUR project "SI-LAB BA2KNOW – Business Analitycs to Know", and by Regione Calabria, programme POR Calabria FESR 2007-2013, projects "ITravel PLUS" and "KnowRex: Un sistema per il riconoscimento e l'estrazione di conoscenza". A. Pieris was supported by the Austrian Science Fund (FWF): P25207-N23 and Y698.

References

[Abiteboul *et al.*, 1995] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[Calì *et al.*, 2012a] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012.

[Calì *et al.*, 2012b] Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.

[Calvanese *et al.*, 2007] 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(3):385–429, 2007.

[Gottlob *et al.*, 2014a] Georg Gottlob, Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, Thomas Schwentick, and Michael Zakharyaschev. The price of query rewriting in ontology-based data access. *Artif. Intell.*, 213:42–59, 2014.

[Gottlob *et al.*, 2014b] Georg Gottlob, Marco Manna, and Andreas Pieris. Polynomial combined rewritings for existential rules. In *KR*, 2014.

[Kikot *et al.*, 2011] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. Polynomial conjunctive query rewriting under unary inclusion dependencies. In *RR*, pages 124–138, 2011.

[Kontchakov *et al.*, 2010] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to query answering in DL-Lite. In *KR*, 2010.

[Kontchakov *et al.*, 2011] Roman Kontchakov, Carsten Lutz, David Toman, Frank Wolter, and Michael Zakharyaschev. The combined approach to ontology-based data access. In *IJCAI*, pages 2656–2661, 2011.

[Lutz *et al.*, 2009] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *IJCAI*, pages 2070–2075, 2009.