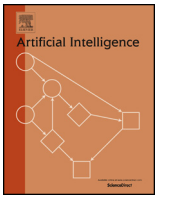
Artificial Intelligence 321 (2023) 103936

Contents lists available at ScienceDirect

Artificial Intelligence

journal homepage: www.elsevier.com/locate/artint



Polynomial combined first-order rewritings for linear and guarded existential rules ☆

Georg Gottlob [a], Marco Manna [b], Andreas Pieris [c,d,*]

[a] Department of Computer Science, University of Oxford, UK
[b] Department of Mathematics and Computer Science, University of Calabria, Italy
[c] School of Informatics, University of Edinburgh, UK
[d] Department of Computer Science, University of Cyprus, Cyprus

ARTICLE INFO

Article history:
Received 4 May 2021
Received in revised form 19 April 2023
Accepted 25 April 2023
Available online 2 May 2023

Keywords:
Ontologies
Existential rules
Tuple-generating dependencies
Guardedness
Conjunctive queries
Query answering
Query rewriting
Combined approach

ABSTRACT

We consider the problem of ontological query answering, that is, the problem of answering a database query (typically a conjunctive query) in the presence of an ontology. This means that during the query answering process we also need to take into account the knowledge that can be inferred from the given database and ontology. Building, however, ontology-aware database systems from scratch, with sophisticated optimization techniques, is a highly non-trivial task that requires a great engineering effort. Therefore, exploiting conventional database systems is an important route towards efficient ontological query answering. Nevertheless, standard database systems are unaware of ontologies. An approach to ontological query answering that enables the use of standard database systems is the so-called polynomial combined query rewriting, originally introduced in the context of description logics: the conjunctive query q and the ontology Σ are rewritten in polynomial time into a first-order query q_Σ (in a database-independent way), while the database D and the ontology Σ are rewritten in polynomial time into a new database D_Σ (in a query-independent way), such that the answer to q in the presence of Σ over D coincides with the answer to q_Σ over D_Σ. The latter can then be computed by exploiting a conventional database system.

In this work, we focus on linear and guarded existential rules, which form robust rule-based languages for modeling ontologies, and investigate the limits of polynomial combined query rewriting. In particular, we show that this type of rewriting can be successfully applied to (i) linear existential rules when the rewritten query can use the full power of first-order queries, (ii) linear existential rules when the arity of the underlying schema is fixed and the rewritten query is positive existential, namely it uses only existential quantification, conjunction, and disjunction, and (iii) guarded existential rules when the underlying schema is fixed and the rewritten query is positive existential. We can show that the above results reach the limits (under standard complexity-theoretic assumptions such as PSPACE $\neq$ EXPTIME) of polynomial combined query rewriting in the case of linear and guarded existential rules.

☆ This paper is an extended and revised version of the papers [1], [2] and [3].

* Corresponding author.
E-mail addresses: georg.gottlob@cs.ox.ac.uk (G. Gottlob), manna@mat.unical.it (M. Manna), apieris@inf.ed.ac.uk (A. Pieris).

https://doi.org/10.1016/j.artint.2023.103936

1. Introduction

Over the past two decades we have seen a shift from a world where most data used by public and private organizations was stored in well-structured relational databases of modest size and treated as complete to a world where data is very large, heterogeneous, distributed in different sources, and incomplete. This makes the task of extracting useful information from such data by means of queries extremely tedious and complex. At the same time, not only do we have massive amounts of data, but we also have very large amounts of knowledge about the application domain of the data in the form of taxonomies, or even full-fledged ontologies. This gave rise to a new research field, recently dubbed *knowledge-enriched data management* [4], that lies at the intersection of data management and knowledge representation and reasoning. A major challenge for knowledge-enriched data management is to provide end users with flexible and integrated access to data by exploiting the available knowledge about the underlying application domain. This builds on the hypothesis that end users may have a deep understanding of a specific domain of interest, but are not able to formulate complex queries and understand performance implications.

Ontology-based data access (OBDA) [5], also known as *ontology-based data integration*, has been proposed as a general paradigm for addressing the above central challenge. It facilitates access to data by separating the end user from the raw data sources. This is done by using an ontology, which models the underlying application domain and is semantically linked with the data via declarative mappings, as a mediator between the data sources and the end user. The purpose of the ontology is two-fold:

1. It provides an integrated global view of the data that is very close to the conceptual model of the underlying application domain of which the end user has a good understanding. This makes the raw data accessible via database queries formulated solely in the vocabulary of the ontology, without requiring any knowledge of the actual structure of the data sources.
2. It enriches the possibly incomplete data sources with domain knowledge. This allows us to infer new knowledge, not explicit in the data, enabling more complete answers to queries.

The main algorithmic task underlying the OBDA paradigm is querying knowledge-enriched data, or, in other words, querying data in the presence of an ontology. This means that during the query answering process we also need to take into account the inferred knowledge. This problem is known as *ontological query answering*.

1.1. Query rewriting

Building ontology-aware database systems from scratch, with sophisticated optimization techniques, is a highly non-trivial task that requires a great engineering effort. An alternative route towards efficient ontological query answering is to use conventional database management systems (DBMSs). The fact that DBMSs are unaware of ontologies can be addressed by *query rewriting*: the database query q (typically a conjunctive query) and the ontology Σ are rewritten into a new query q_Σ, the so-called *rewriting*, which computes the answer to q in the presence of Σ over *all* input databases. It is, of course, essential that q_Σ is expressed in a language that can be handled by standard DBMSs. The typical language is that of first-order (FO) queries.

The Pure Approach. What has been described above is the so-called *pure approach* to FO rewritability in the sense that the FO rewriting q_Σ should be powerful enough to compute the correct answer to the given query q under the given ontology Σ over all input databases. This essentially means that the construction of q_Σ should be independent of any database. The advantage of such a pure approach to FO rewritability should be clear: we can pre-compute q_Σ offline, and whenever the database D changes, we simply need to re-evaluate q_Σ over D, without having to re-compute it. This approach has been successfully applied to a range of lightweight description logics, mainly the members of the DL-Lite family [6], as well as classes of existential rules such as linear existential rules [7,8]; details on existential rules are given below. On the other hand, such a pure approach to FO rewritability comes with two inevitable shortcomings:

1. Query rewriting algorithms generate from a reasonably sized conjunctive query a very large FO query, which can be prohibitive for efficient execution by a standard database system. We actually know that even for lightweight ontology languages such as DL-Lite$_\mathcal{R}$ [6], the logical underpinning of the OWL 2 QL profile of OWL 2,[1] there is no FO rewriting of polynomial size, unless the polynomial hierarchy collapses [9]. Further strong evidence for the non-existence of an FO rewriting of polynomial size in the case of DL-Lite$_\mathcal{R}$ was given in [10]. In particular, it was shown that the existence of such an FO rewriting is equivalent to a major open problem in computational complexity such as $\mathrm{NC}_1 = \mathrm{NP/poly}$.[2] We also know that an exponential blow-up is provably unavoidable when the rewriting should be an existential positive FO query (i.e., an FO query that uses only existential quantification, conjunction, and disjunction) [9].

[1] https://www.w3.org/TR/owl2-profiles/#OWL_2_QL.

[2] NC_1 is the class of decision problems decidable by uniform Boolean circuits with a polynomial number of gates of at most two inputs and depth $O(\log n)$, whereas NP/poly is the non-uniform analogue of NP.

2. FO rewritability applies only to lightweight ontology languages for which the data complexity of ontological query answering (i.e., when the query and the ontology are considered fixed) is very low. More precisely, the problem of evaluating a fixed FO query over an input database is known to be in AC_0,[3] a class that is properly contained in DLogSpace. Therefore, useful formalisms with PTime-hard (or even DLogSpace-hard) data complexity, such as the description logic $\mathcal{EL}$ [11], are immediately excluded.

The Combined Approach. To overcome the above shortcomings, a finer approach to FO rewritability has been proposed in the context of description logics, known as the *combined approach* [12]. The crucial difference compared to the pure approach is that rewriting the database in a query-independent way is also possible. More precisely, the conjunctive query q and the ontology Σ are rewritten into a new query q_Σ (in a database-independent way), while the database D and the ontology Σ are rewritten into a new database D_Σ (in a query-independent way), such that the answer to q in the presence of Σ over D coincides with the answer to q_Σ over D_Σ.

It has been shown that, indeed, the two shortcomings of the pure approach to FO rewritability discussed above can be overcome by adopting the combined approach. It has been successfully applied to a range of lightweight description logics, mainly members of the DL-Lite and $\mathcal{EL}$ families, with the guarantee that both the database and the query rewriting are feasible in polynomial time [12–14]. It has been also applied to description logics that go beyond the members of the DL-Lite and $\mathcal{EL}$ families [15].

1.2. Research challenges

All the results discussed above concerning the combined approach to FO rewritability are about description logics. But what about rule-based ontology languages? It is generally agreed that rule-based ontologies are well-suited for data-intensive applications such as OBDA, since they allow us to conveniently deal with higher-arity relations, which naturally appear in standard relational databases. Therefore, studying whether the combined approach to FO rewritability can be applied to rule-based ontology languages is a highly relevant task that deserves our attention.

Towards this direction, we focus on ontologies modeled using *existential rules*, also called *tuple-generating dependencies*, i.e., first-order sentences of the form

$$\forall \bar{x} \forall \bar{y} \, (\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \, \psi(\bar{x}, \bar{z}))$$

with ϕ and ψ being conjunctions of atoms. Sentences of the above form are also known in the literature as *Datalog$^\pm$ rules* [16]. However, ontological query answering under arbitrary existential rules is undecidable (see, e.g., [17]). This has led to an intensive research activity during the last decade for identifying restrictions on existential rules that lead to decidability. The basic decidable paradigms that emerged from this effort are guardedness [17,18], (weak-)acyclicity [19,20], stickiness [21], and shyness [22].

In this work, we concentrate on guardedness and investigate the limits of the polynomial combined approach to FO rewritability – the term polynomial refers to the fact that both the database and the query rewriting should be feasible in polynomial time. An existential rule as the one above is *guarded* if ϕ has an atom that contains (or guards) all the universally quantified variables [17]. A central subclass of guarded existential rules is that of *linear* existential rules, i.e., existential rules where ϕ consists of a single atom [7]. Interestingly, the main members of the $\mathcal{EL}$ family of description logics are (up to a certain normal form) special cases of guarded existential rules, while the main members of the DL-Lite family (modulo some easily handled features) are special cases of linear existential rules. By employing simple complexity-theoretic arguments, we can delineate the limits of the polynomial combined approach to FO rewritability in the case of guarded and linear existential rules:

Targeting Arbitrary FO Queries. Evaluation of FO queries is known to be PSpace-complete. Thus, when the rewritten query has the freedom to use the full power of FO queries, it is unlikely that the polynomial combined approach can be applied to ontology languages for which the complexity of ontological query answering is beyond PSpace. This immediately excludes guarded existential rules, for which ontological query answering is 2ExpTime-complete [17]. It actually excludes the class of guarded existential rules even when the arity of underlying schema is fixed (unless PSpace = ExpTime) as in this case ontological query answering is known to be ExpTime-complete [17].

On the other hand, ontological query answering for linear existential rules is PSpace-complete. Therefore, there is no complexity-theoretic argument against the polynomial combined approach being applied to linear existential rules.

Targeting Existential Positive FO Queries. The problem of evaluating positive existential FO queries is known to be NP-complete. Hence, when the rewritten query should be a positive existential FO query, it is unlikely that the polynomial combined approach can be applied to linear existential rules (unless NP = PSpace).

On the other hand, ontological query answering for guarded existential rules when the underlying schema is fixed, as well as for linear existential rules when the arity of the underlying schema is fixed, is NP-complete (this is actually

[3] The class AC_0 consists of those languages that are accepted by polynomial-size circuits of constant depth and unbounded fan-in (the number of inputs to their gates).

a result of the present work – further details are given below). Hence, there is no complexity-theoretic argument against the polynomial combined approach being applied to those cases when the rewritten FO query should be positive existential.

The above discussion leads to the following fundamental research questions concerning the limits of the polynomial combined approach to FO rewritability:

1. Is it applicable to linear existential rules when the rewritten query can use the full power of FO queries?
2. Is it applicable to linear existential rules when the arity of the underlying schema is fixed, and the rewritten FO query should be positive existential?
3. Is it applicable to guarded existential rules when the underlying schema is fixed, and the rewritten FO query should be positive existential?

1.3. Summary of contributions

The goal of the present work is to provide answers to the above challenging questions. Our main results can be summarized as follows:

- In Section 3, we show that linear existential rules enjoy the so-called bounded witness property (Theorem 3.1). This result essentially tells us that for ontological query answering under linear existential rules it suffices to apply a bounded number of inference steps, while the bound depends only on the set of rules and the CQ, but not on the input database. This is a result of independent interest and it allows us to provide answers to our main research questions. Note that such a result can be obtained from [23] for single-head linear existential rules, that is, linear existential rules with only one atom in the right-hand side of the implication. However, this result cannot be straightforwardly transferred from single-head to multi-head linear existential rules. We now provide a proof that deals with linear existential rules in their full generality.
- In Section 4, we show that (i) the polynomial combined approach is applicable to linear existential rules, and (ii) it is also applicable when the rewriting should be a positive existential FO query providing that the arity of the underlying schema is fixed (Theorem 4.1). This result heavily relies on the fact that linear existential rules enjoy the bounded witness property. As a corollary, ontological query answering in the case of linear existential rules over schemas of fixed arity is in NP since evaluating positive existential FO queries is in NP.
  Note that the results for linear existential rules over schemas of fixed arity, i.e., item (ii) of Theorem 4.1, as well as the NP upper bound for ontological query answering, were known only for single-head linear existential rules [9,23]. We now provide proofs that deal with linear existential rules in their full generality.
- Finally, in Section 5, we show that the polynomial combined approach is applicable to guarded existential rules when the rewriting should be a positive existential FO query providing that the underlying schema is fixed (Theorem 5.1). This result exploits item (ii) of Theorem 4.1, i.e., the fact that the polynomial combined approach is applicable to linear existential rules when the rewriting should be a positive existential FO query providing that the arity of the underlying schema is fixed. In fact, we provide a polynomial-time combined reduction from ontological query answering under guarded existential rules over fixed schemas, to ontological query answering under linear existential rules over schemas of fixed arity, and then apply item (ii) of Theorem 4.1. As a corollary, ontological query answering in the case of guarded existential rules over fixed schemas is in NP.
  The latter complexity result was only known for single-head guarded existential rules [17]. This was recently brought to our attention by a colleague of ours [24], and it is also explicitly discussed in [25]. We now have a proof that deals with guarded existential rules in their full generality.

Single-head vs. Multi-head Existential Rules. We conclude this introductory section by discussing further the subtle issue regarding single-head and multi-head existential rules. As mentioned above, some of our results were only known for single-head existential rules. This, however, may sound contradictory to the general assumption that, for ontological query answering purposes, allowing for a conjunction of atoms in the right-hand side of an existential rule is actually syntactic sugar. This is because we can always convert a set Σ of existential rules into a set Σ_1 of existential rules with only one atom in the right-hand side such that Σ and Σ_1, although not logically equivalent, are equivalent for ontological query answering. This, in fact, relies on a very simple transformation that replaces each existential rule $\sigma \in \Sigma$ of the form

$$\forall \bar{x} \forall \bar{y} \big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, R_1(\bar{x}_1, \bar{z}_1) \wedge \cdots \wedge R_n(\bar{x}_n, \bar{z}_n) \big),$$

where $\bar{x}_i \subseteq \bar{x}$ and $\bar{z}_i \subseteq \bar{z}$, with the set of existential rules

$$\forall \bar{x} \forall \bar{y} \big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \mathrm{Aux}_\sigma(\bar{x}, \bar{z}) \big)$$
$$\forall \bar{x} \forall \bar{z} (\mathrm{Aux}_\sigma(\bar{x}, \bar{z}) \rightarrow R_i(\bar{x}_i, \bar{z}_i)), \text{ for } i \in \{1, \ldots, n\},$$

with Aux_σ being a fresh predicate not occurring in Σ. Notice further that if Σ is linear (resp., guarded), then also Σ_1 is linear (resp., guarded).

Due to the above transformation, it is usually the case that ontological query answering for linear and guarded existential rules is studied under the assumption that in the right-hand side of an existential rule we have only one atom. The reason is purely technical, i.e., to simplify the technical definitions and proofs. Although this simplifying assumption can be made, in general, without affecting the generality of the results, this is *not* true for schemas of fixed arity, as well as for fixed schemas. This is because, even if we start from a set of existential rules over a fixed schema, the obtained set of single-head existential rules after the transformation mentions an unbounded number of new predicates each of unbounded arity (see the auxiliary predicates).

2. Preliminaries

We consider the disjoint countably infinite sets **C**, **N** and **V** of *constants*, *nulls* and *variables*, respectively. We refer to constants, nulls and variables as *terms*. For an integer $n \geq 1$, we may write $[n]$ for the set $\{1, \ldots, n\}$.

Relational Instances. Consider a countably infinite set **Rel** of *relation symbols* (also called *predicates*) with associated arity. We write $\text{ar}(R)$ for the arity of a predicate R. A *(relational) schema* **S** is a finite subset of **Rel**. We write $\text{ar}(\mathbf{S})$ for the arity of **S**, i.e., the number $\max_{R \in \mathbf{S}}\{\text{ar}(R)\}$. An *atom* over **S** is an expression of the form $R(\bar{t})$, where $R \in \mathbf{S}$ and $\bar{t}$ is a tuple of terms. For an atom α, $\text{dom}(\alpha)$, $\text{null}(\alpha)$ and $\text{var}(\alpha)$ is the set of its terms, nulls, and variables, respectively; these notations naturally extend to sets of atoms. Given a set of terms T, we define $\mathsf{B}(T, \mathbf{S}) = \{R(\bar{t}) \mid R \in \mathbf{S} \text{ and } \bar{t} \in T^{\text{ar}(R)}\}$, that is, the set of atoms that can be formed using terms of T and predicates of **S**. An *instance* over **S** is a (possibly infinite) set of atoms over **S** with constants and nulls, while a *database* over **S** is a finite instance over **S** with only constants. We write $\mathbb{D}$ for the family of all databases. We also write $\mathbb{D}[\mathbb{S}]$, where $\mathbb{S}$ is a (finite or infinite) family of schemas, for the family of all databases over a schema $\mathbf{S} \in \mathbb{S}$.

Homomorphisms. A *homomorphism* from a set of atoms A to a set of atoms B is a function $h : \text{dom}(A) \to \text{dom}(B)$ that is the identity on **C** with $R(h(\bar{t})) \in B$ for every $R(\bar{t}) \in A$. We write $A \to B$ for the fact that there is a homomorphism from A to B. For a set of terms S, we say that A and B are *S-isomorphic*, denoted $A \simeq_S B$, if there is a 1-1 homomorphism h from A to B that is the identity on S and h^{-1} maps B to A.

Queries. A *first-order query* (FO) over a schema **S** is an expression of the from

$$q(\bar{x}) \ := \ \{\bar{x} \mid \phi\},$$

where ϕ is a first-order formula that uses predicates from **S** and mentions only constants and variables, $\bar{x}$ is a tuple (possibly with repetitions) over the free variables of ϕ, and each free variable of ϕ occurs at least once in $\bar{x}$. We say that q is *existential positive* ($\exists\text{FO}^+$) if ϕ uses only existential quantification, conjunction, and disjunction. For a database D over **S**, the *evaluation of $q(\bar{x})$ over D*, denoted $q(D)$, is the set of tuples

$$\left\{\bar{c} \in \text{dom}(D)^{|\bar{x}|} \mid \bar{c} \sim \bar{x} \text{ and } D \models_{\mathsf{FO}} \phi(\bar{c})\right\},$$

where $\bar{c} \sim \bar{x}$ denotes the fact that $\bar{c} = (c_1, \ldots, c_n)$ is compatible with $\bar{x} = (x_1, \ldots, x_n)$, i.e., $x_i = x_j$ implies $c_i = c_j$, $\phi(\bar{c})$ is the sentence obtained after instantiating each free variable of ϕ with the corresponding constant in $\bar{c}$, and $\models_{\mathsf{FO}}$ denotes the standard active domain semantics of first-order logic. Recall that by active domain semantics we essentially mean that the quantified variables range over the terms occurring in the underlying database, i.e., its active domain; hence the name "active domain semantics". Let $\mathbb{FO}$ and $\exists\mathbb{FO}^+$ be the family of all FO and $\exists\text{FO}^+$ queries.

A subclass of first-order queries that is central for the present work is that of conjunctive queries, which actually corresponds to first-order queries that use only existential quantification and conjunction. In particular, a *conjunctive query* (CQ) over a schema **S** is a formula of the form

$$q(\bar{x}) \ := \ \underbrace{\exists \bar{y} \left(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m)\right)}_{\phi},$$

where each $R_i(\bar{v}_i)$ is an atom without nulls, each variable mentioned in the $\bar{v}_i$'s appears either in $\bar{x}$ or $\bar{y}$, $\bar{x}$ is tuple (possibly with repetitions) over the free variables of ϕ, and each free variable of ϕ occurs at least once in $\bar{x}$. If $\bar{x}$ is empty, then q is a *Boolean CQ*. For an instance I over **S**, the *evaluation of $q(\bar{x})$ over I*, denoted $q(I)$, is the set

$$\left\{\bar{c} \in (\text{dom}(I) \cap \mathbf{C})^{|\bar{x}|} \mid \bar{c} \sim \bar{x} \text{ and } q(\bar{c}) \to I\right\},$$

i.e., are the tuples of constants $\bar{c}$ such that $q(\bar{c})$ can be mapped via a homomorphism to I; notice that, by abuse of terminology, we may treat a conjunction of atoms as a set of atoms. Let $\mathbb{CQ}$ be the family of all CQs. For a family of schemas $\mathbb{S}$, let $\mathbb{CQ}[\mathbb{S}]$ be the family of all CQs over a schema of $\mathbb{S}$. Note that the evaluation of CQs is defined not only for databases but also for instances. This is not needed for arbitrary FO (or even $\exists\text{FO}^+$) queries since in the rest of the paper will always be evaluated over databases.

Tuple-generating Dependencies. A *tuple-generating dependency* (TGD) σ over a schema $\mathbf{S}$ is a first-order sentence of the form

$$\forall \bar{x} \forall \bar{y} \big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}) \big),$$

where ϕ and ψ are (non-empty) conjunctions of atoms over $\mathbf{S}$ that mention only variables. For brevity, we write σ as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ and use comma instead of $\wedge$ for joining atoms. We call ϕ and ψ the *body* and *head* of σ, denoted $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$, respectively. An instance I over $\mathbf{S}$ satisfies σ, written $I \models \sigma$, if, whenever $\phi(\bar{x}, \bar{y}) \rightarrow I$ via a homomorphism h, then $\psi(\bar{x}, \bar{z}) \rightarrow I$ via a homomorphism h' that agrees with h on $\bar{x}$. The instance I satisfies a set Σ of TGDs, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$. Let $\mathbb{TGD}$ be the family of all finite sets of TGDs.[4] A class $\mathbb{C}$ of TGDs is a subset of $\mathbb{TGD}$. We also write $\mathbb{C}[\mathbb{S}]$, where $\mathbb{S}$ is a family of schemas, for the class of TGDs $\{\Sigma \in \mathbb{C} \mid \Sigma \text{ is over a schema of } \mathbb{S}\}$.

Guardedness and Linearity. A TGD σ is *guarded* if there exists an atom in $\mathsf{body}(\sigma)$, called *guard*, that contains all the body variables. By convention, the leftmost body atom of a guarded TGD σ is the guard, denoted $\mathsf{guard}(\sigma)$, and all the other atoms are the *side* atoms of σ. We write $\mathbb{G}$ for the class of sets of guarded TGDs. A subclass of $\mathbb{G}$, which is crucial for our work, is the class of *linear* TGDs, denoted $\mathbb{L}$, which collects all the sets of TGDs with only one body-atom.

Ontological Query Answering. Given a database D and a set Σ of TGDs, both over a schema $\mathbf{S}$, a *model* of D and Σ is a (possibly infinite) instance $I \supseteq D$ such that $I \models \Sigma$. We write $\mathsf{mods}(D, \Sigma)$ for the set of models of D and Σ. The *certain answers* to a CQ q over $\mathbf{S}$ w.r.t. D and Σ is defined as the set of tuples

$$\mathsf{cert}(q, D, \Sigma) = \bigcap_{I \in \mathsf{mods}(D, \Sigma)} q(I).$$

Ontological query answering is the problem of computing the set $\mathsf{cert}(q, D, \Sigma)$. The associated decision problem is defined as follows. Let $\mathbb{C}$ be a class of TGDs:

PROBLEM :	$\mathsf{OQA}(\mathbb{C})$
INPUT :	A database D, a set $\Sigma \in \mathbb{C}$ of TGDs, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{\lvert\bar{x}\rvert}$.
QUESTION :	Does $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$?

Polynomial Combined Rewritability. We now introduce the central notion of polynomial combined first-order rewritability. As discussed in the Introduction, this has been proposed as an alternative to query rewriting with the aim of overcoming the limitations of the standard first-order rewritability approach for ontological query answering.

A *database rewriter* is a function that takes as input a database and a set of TGDs, and outputs a database, while a *query rewriter* is a function that takes as input a CQ and a set of TGDs, and outputs an FO query. We will always make clear what is the domain and codomain of database and query rewriters.

A class $\mathbb{C} \subseteq \mathbb{TGD}$ of TGDs is *polynomially combined* FO-*rewritable* (resp., $\exists$FO$^+$-*rewritable*) if there are polynomial-time computable database and query rewriters

$$f_{\mathsf{DB}} : \mathbb{D} \times \mathbb{C} \rightarrow \mathbb{D} \quad \text{and} \quad f_{\mathsf{Q}} : \mathbb{CQ} \times \mathbb{C} \rightarrow \mathbb{FO} \quad (\text{resp., } f_{\mathsf{Q}} : \mathbb{CQ} \times \mathbb{C} \rightarrow \exists\mathbb{FO}^+),$$

respectively, such that, for every database D, set $\Sigma \in \mathbb{C}$, and CQ q, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_\Sigma)$ with $D_\Sigma = f_{\mathsf{DB}}(D, \Sigma)$ and $q_\Sigma = f_{\mathsf{Q}}(q, \Sigma)$.

The above are defined analogously in case we focus on a family of schemas $\mathbb{S}$. More precisely, $\mathbb{C}[\mathbb{S}]$ is *polynomially combined* FO-*rewritable* (resp., $\exists$FO$^+$-*rewritable*) if there are polynomial-time computable database and query rewriters

$$f^{\mathbb{S}}_{\mathsf{DB}} : \mathbb{D}[\mathbb{S}] \times \mathbb{C}[\mathbb{S}] \rightarrow \mathbb{D} \quad \text{and} \quad f^{\mathbb{S}}_{\mathsf{Q}} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{C}[\mathbb{S}] \rightarrow \mathbb{FO} \quad (\text{resp., } f^{\mathbb{S}}_{\mathsf{Q}} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{C}[\mathbb{S}] \rightarrow \exists\mathbb{FO}^+),$$

respectively, such that, for every database $D \in \mathbb{D}[\mathbb{S}]$, set $\Sigma \in \mathbb{C}[\mathbb{S}]$, and CQ $q \in \mathbb{CQ}[\mathbb{S}]$, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_\Sigma)$ with $D_\Sigma = f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma)$ and $q_\Sigma = f^{\mathbb{S}}_{\mathsf{Q}}(q, \Sigma)$.

The Chase Procedure. The chase procedure is a useful algorithmic tool when reasoning with TGDs. Let us first define a single chase application. A *trigger* for a set Σ of TGDs on an instance I is a pair (σ, h), where $\sigma \in \Sigma$ and h is a homomorphism from $\mathsf{body}(\sigma)$ to I. An *application* of (σ, h) to I returns an instance $J = I \cup h'(\mathsf{head}(\sigma))$, where h' extends h in such a way that (i) for each existentially quantified variable z of σ, $h'(z) \in \mathbf{N} \setminus \mathsf{dom}(I)$, and (ii) for distinct existentially quantified variables z and w of σ, $h'(z) \neq h'(w)$. Such a trigger application is denoted by $I\langle \sigma, h \rangle J$.

The main idea of the chase is, starting from a database D, to exhaustively apply distinct triggers for the given set Σ of TGDs on the instance constructed so far, and keep doing this until a fixpoint is reached. This is formalized as follows:

[4] We work with finite sets of TGDs. Thus, in the rest of the paper, a set of TGDs is always finite.

- A finite sequence of instances $(I_i)_{0\leq i\leq n}$, with $D = I_0$ and $n \geq 0$, is a *chase derivation* of D w.r.t. Σ if: (i) for each $0 \leq i < n$, there is a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$, (ii) for each $0 \leq i < j < n$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j$ implies $h_i \neq h_j$, and (iii) there is no trigger (σ, h) for Σ on I_n such that $(\sigma, h) \notin \{(\sigma_i, h_i)\}_{0\leq i<n}$. In this case, the result of the chase is the (finite) instance I_n.
- An infinite sequence of instances $(I_i)_{i\geq 0}$ is a *chase derivation* of D w.r.t. Σ if: (i) for each $i \geq 0$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$, (ii) for each $i, j > 0$ such that $i \neq j$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j$ implies $h_i \neq h_j$, and (iii) for each $i \geq 0$ and for every trigger (σ, h) for Σ on I_i, there exists $j \geq i$ such that $I_j\langle\sigma, h\rangle I_{j+1}$. The latter is called the *fairness condition*, and it ensures that all the triggers will be applied. The result of the chase is the infinite instance is $\bigcup_{i\geq 0} I_i$.

We write $\mathsf{chase}_\delta(D, \Sigma)$ for the result of a (finite or infinite) chase derivation δ of D w.r.t. Σ. The key property of the chase follows:

Proposition 2.1. *Consider a database D, a set Σ of TGDs, and a chase derivation δ of D w.r.t. Σ. For every instance $I \in \mathsf{mods}(D, \Sigma)$, it holds that $\mathsf{chase}_\delta(D, \Sigma) \rightarrow I$.*

In our later technical proofs, we will silently use a crucial consequence of the above result, namely a tuple $\bar{c}$ belongs to $\mathsf{cert}(q, D, \Sigma)$ iff there exists a prefix $(I_i)_{0\leq i\leq n}$, for $n \geq 0$, of a chase derivation of D w.r.t. Σ such that $\bar{c} \in q(I_n)$.

We conclude our discussion on the chase procedure by defining a useful relation on the result of chase derivations. Consider a chase derivation $\delta = (I_i)_{i\geq 0}$ of a database D w.r.t. a set Σ of TGDs, and assume that for each $i \geq 0$, $I_i\langle\sigma_i, h_i\rangle I_{i+1}$. The *parent relation* of δ, denoted $\prec^p_\delta$, is a binary relation over $\mathsf{chase}_\delta(D, \Sigma)$ such that $\alpha \prec^p_\delta \beta$ iff there is $i \geq 0$ such that $\alpha \in h_i(\mathsf{body}(\sigma_i))$ and $\beta \in I_{i+1} \setminus I_i$. Let $\prec^{p,+}_\delta$ be the transitive closure of $\prec^p_\delta$. Note that $\prec^p_\delta$ forms a (possibly infinite) directed acyclic graph.

3. Bounded witness property and linearity

We start our analysis by introducing the so-called bounded witness property for a class of TGDs and then show that the class of linear TGDs enjoys this property. This essentially tells us that for ontological query answering purposes it suffices to apply a bounded number of chase steps, while the bound depends only on the set of TGDs and the CQ, but not on the input database. As we shall see, the fact that linear TGDs enjoy the bounded witness property is crucial for obtaining the desired polynomial combined first-order rewritings, which is the subject of Sections 4 and 5.

Definition 3.1 *(Bounded Witness Property).* A class $\mathbb{C}$ of TGDs enjoys the *bounded witness property due to f*, where f is a computable function from $\mathbb{C} \times \mathbb{CQ}$ to the natural numbers, if, for every database D, set $\Sigma \in \mathbb{C}$ of TGDs, CQ $q(\bar{x})$, and tuple $\bar{c} \in \mathbf{C}^{|\bar{x}|}$, $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$ implies the existence of a sequence $(I_i)_{0\leq i\leq n}$, with $n \leq f(\Sigma, q)$, that is a prefix of a chase derivation of D w.r.t. Σ, and $\bar{c} \in q(I_n)$. We simply say that $\mathbb{C}$ enjoys the *bounded witness property* if it enjoys the bounded witness property due to some computable function f from $\mathbb{C} \times \mathbb{CQ}$ to the natural numbers.[5] □

We now proceed with main result of this section, that is, the class of linear TGDs enjoys the bounded witness property. We first define the function $f_{\mathbb{L}}$ that maps $\mathbb{L} \times \mathbb{CQ}$ to the natural numbers. Given a set Σ of TGDs and a CQ q, both over a schema $\mathbf{S}$:

$$f_{\mathbb{L}}(\Sigma, q) \;=\; |\Sigma| \cdot |\mathbf{S}| \cdot (2 \cdot |q| \cdot \mathsf{ar}(\mathbf{S}) + \mathsf{ar}(\mathbf{S}))^{\mathsf{ar}(\mathbf{S})}.$$

It is clear that $f_{\mathbb{L}}$ is a computable function. We proceed to show that:

Theorem 3.1. *$\mathbb{L}$ enjoys the bounded witness property due to $f_{\mathbb{L}}$.*

The proof of Theorem 3.1 proceeds in two main steps:

1. We characterize when $\mathbb{L}$ enjoys the bounded witness property due to $f_{\mathbb{L}}$ via parsimonious quasi chase derivations, i.e., sequences that are "almost" a prefix of a chase derivation, and the number of distinct triggers involved is bounded by $f_{\mathbb{L}}$.
2. We then show that, when we focus on $\mathbb{L}$, the fact that a tuple is a certain answer can be witnessed via a parsimonious quasi chase derivation.

[5] When f is a polynomial function, this is known in the literature as the *polynomial witness property* [9].

3.1. Some auxiliary notions

Before we give the technical details for the above two steps, we first need to introduce some auxiliary technical notions that are needed for the proof.

Quasi Chase Derivations. Let (σ, h) be a trigger for a set Σ of TGDs on an instance I. A *partial application* of (σ, h) to I returns an instance $J = I \cup K$, where $K \subseteq h'(\mathsf{head}(\sigma))$, while h' extends h in such a way that for each existentially quantified variable z of σ, $h'(z) \in \mathbf{N} \setminus \mathsf{dom}(I)$, and for distinct existentially quantified variables z and w of σ, $h'(z) \neq h'(w)$. Such a partial application is denoted $I\langle\sigma, h\rangle^p J$. A sequence of instances $(I_i)_{0\leq i\leq n}$, for $n \geq 0$, is a *quasi chase derivation* (resp., *quasi chase derivation with total applications*) of D w.r.t. Σ if, for each $0 \leq i < n$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle^p I_{i+1}$ (resp., $I_i\langle\sigma, h\rangle I_{i+1}$). Observe that a quasi chase derivation with total applications is trivially a quasi chase derivation, but the opposite is not necessarily true. A quasi chase derivation $\delta = (I_i)_{0\leq i\leq n}$, with $I_i\langle\sigma_i, h_i\rangle^p I_{i+1}$, is *k-parsimonious*, for $k \geq 0$, if $|\bigcup_{0\leq i<n}\{(\sigma_i, h_i)\}| \leq k$, i.e., at most k distinct triggers are involved in δ. The parent relation of δ, denoted $\prec_\delta^p$, and its transitive closure $\prec_\delta^{p,+}$, are defined as usual.

In a nutshell, a quasi chase derivation is "almost" a chase derivation, but it is not exactly a chase derivation for the following three reasons:

1. applications may be partial, i.e., during a trigger application some atoms are not generated,
2. triggers may repeat, i.e., the same trigger is used in different applications, and
3. some triggers have not been applied.

Of course, in the case of quasi chase derivations with total applications only the last two reasons apply. Moreover, although triggers may repeat in a k-parsimonious quasi chase derivation, at most k distinct triggers are used for its generation.

Contractions. Consider a database D, a set $\Sigma \in \mathbb{L}$ of TGDs, and a quasi chase derivation $\delta = (I_i)_{0\leq i\leq n}$, for $n \geq 0$, of D w.r.t. Σ. Let $A \subseteq I_n$ and h a function from $\mathsf{dom}(A)$ to $\mathsf{dom}(I_n)$. The (A, h)*-contraction of* δ is the sequence of instances $\delta' = (J_i)_{0\leq i\leq n}$ such that, for each $0 \leq i \leq n$,[6]

$$J_i \;=\; I_i \setminus \left\{\alpha \mid \alpha \in I_i \text{ and } A \prec_\delta^{p,+} \alpha\right\} \;\cup\; \left\{h(\alpha) \mid \alpha \in I_i \text{ and } A \prec_\delta^{p,+} \alpha\right\}.$$

For notational convenience, we write $A \prec_\delta^{p,+} \alpha$ for the fact that $\beta \prec_\delta^{p,+} \alpha$ for some atom $\beta \in A$. To understand the essence of the notion of contraction, observe first that, since we consider linear TGDs, $\prec_\delta^p$ actually forms a forest with the atoms of D being the roots, and thus, we can talk about the subtree of an atom. Now, in simple words, the (A, h)-contraction of δ is essentially what we obtain after updating, according to the function h, the atoms in the subtrees of $\prec_\delta^p$ rooted at A. A simple example that illustrates the notion of contraction follows.

Example 3.1. Consider the database $D = \{R(a, b)\}$ and the set $\Sigma \in \mathbb{L}$ consisting of

$$\sigma_1 = R(x, y) \;\rightarrow\; \exists z \exists w\, P(x, z), P(y, w),$$
$$\sigma_2 = P(x, y) \;\rightarrow\; S(y, x),$$
$$\sigma_3 = S(x, y) \;\rightarrow\; \exists z\, T(x, z), T(y, z).$$

Let $\delta = I_0, I_1, \ldots, I_6$ be the quasi chase derivation of D w.r.t. Σ with

$$I_0 = D$$
$$I_1 = I_0 \cup \{P(a, \bot_1), P(b, \bot_2)\}$$
$$I_2 = I_1 \cup \{S(\bot_1, a)\}$$
$$I_3 = I_2 \cup \{P(a, \bot_3), P(b, \bot_4)\}$$
$$I_4 = I_3 \cup \{S(\bot_3, a)\}$$
$$I_5 = I_4 \cup \{T(\bot_3, \bot_5), T(a, \bot_6)\}$$
$$I_6 = I_5 \cup \{S(\bot_4, b)\}.$$

It is easy to verify that δ is indeed a quasi chase derivation of D w.r.t. Σ. Let $A = \{P(a, \bot_3), P(b, \bot_4)\} \subseteq I_6$, and h be the function from $\mathsf{dom}(A)$ to $\mathsf{dom}(I_6)$ such that $h(a) = a$, $h(b) = b$, $h(\bot_3) = \bot_1$ and $h(\bot_4) = \bot_2$. The (A, h)-contraction of δ is the sequence of instances $\delta' = J_0, J_1, \ldots, J_6$ with

[6] Note that this notion is different than the well-known notion of homomorphism contraction.

$$J_0 = D$$
$$J_1 = J_0 \cup \{P(a, \bot_1), P(b, \bot_2)\}$$
$$J_2 = J_1 \cup \{S(\bot_1, a)\}$$
$$J_3 = J_2 \cup \{P(a, \bot_3), P(b, \bot_4)\}$$
$$J_4 = J_3$$
$$J_5 = J_4 \cup \{T(\bot_1, \bot_5), T(a, \bot_6)\}$$
$$J_6 = J_5 \cup \{S(\bot_2, b)\}.$$

In other words, the sequence δ' is obtained from δ as follows: for each $i \in \{0, 1 \ldots, 6\}$, J_i is obtained by replacing each $\alpha \in I_i$ such that $A \prec_\delta^{p,+} \alpha$ with $h(\alpha)$. □

We can now discuss the details of the proof for Theorem 3.1.

3.2. Bounded witness property and parsimonious quasi chase derivations

The characterization of the bounded witness property due to $f_{\mathbb{L}}$ via parsimonious quasi chase derivations (step 1 of the proof of Theorem 3.1) is as follows:

Proposition 3.1. *The following are equivalent:*

1. $\mathbb{L}$ *enjoys the bounded witness property due to* $f_{\mathbb{L}}$.
2. *For every database* D, *set* $\Sigma \in \mathbb{L}$, *CQ* $q(\bar{x})$, *and tuple* $\bar{c} \in \mathbf{C}^{|\bar{x}|}$, $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$ *implies there exists an* $f_{\mathbb{L}}(\Sigma, q)$*-parsimonious quasi chase derivation with total applications* $(I_i)_{0 \le i \le n}$ *of* D *w.r.t.* Σ *such that* $\bar{c} \in q(I_n)$.

Let $k = f_{\mathbb{L}}(\Sigma, q)$. It is clear that (1) implies (2) since a prefix of a chase derivation of length k of D w.r.t. Σ is by definition a k-parsimonious quasi chase derivation of D w.r.t. Σ. The other direction, however, is not immediate since a k-parsimonious quasi chase derivation δ of D w.r.t. Σ with total applications is not necessarily a prefix of length at most k of a chase derivation of D w.r.t. Σ since triggers may repeat. Thus, if we could eliminate from δ the repeated triggers, without introducing more repetitions of triggers, then we will end up with a finite prefix of a chase derivation of D w.r.t. Σ of length at most k, which in turn implies the bounded witness property due to $f_{\mathbb{L}}$. This can be achieved via iterative contractions as we explain next.

Assume that a trigger (σ, h) has been applied at steps i, j, for $i < j$, with H_i and H_j being the set of atoms generated at the i-th and j-th step, respectively. The key observation is that $H_i \simeq_S H_j$, where $S = \mathsf{dom}(H_i) \cap \mathsf{dom}(H_j)$. Due to linearity, we can safely move the subtrees rooted at H_j under the corresponding atoms of H_i, and consistently update the atoms without introducing more repetitions of triggers. This is essentially what the (H_j, μ)-contraction of δ does with μ being the S-isomorphism from H_j to H_i. This can be seen in Example 3.1. Observe that the instances I_1 and I_3 of the quasi chase derivation δ are obtained by applying the same trigger (σ_1, h) with $h(x) = a$ and $h(y) = b$, and δ' is obtained by moving the subtrees rooted at $\{P(a, \bot_3), P(b, \bot_4)\}$ under the corresponding atoms of $\{P(a, \bot_1), P(b, \bot_2)\}$ and consistently updating the atoms. The above discussion is formalized by the following technical lemma. For a quasi chase derivation $\delta = (I_i)_{0 \le i \le n}$ with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$, we write H_δ^i for the set of atoms generated at the i-th step, i.e., the set of atoms $h_i'(\mathsf{head}(\sigma_i))$, where h_i' is the extension of h_i employed during the trigger application $I_i \langle \sigma_i, h_i \rangle I_{i+1}$.

Lemma 3.1. *Consider a database* D, *a set* $\Sigma \in \mathbb{L}$, *a CQ* $q(\bar{x})$, *and a tuple* $\bar{c} \in \mathbf{C}^{|\bar{x}|}$. *Let* $\delta = (I_i)_{0 \le i \le n}$ *be a quasi chase derivation with total applications of* D *w.r.t.* Σ *with* $I_i \langle \sigma_i, h_i \rangle I_{i+1}$, *and assume* $\bar{c} \in q(I_n)$. *For a pair of indices* $0 \le k < \ell < n$ *with* $(\sigma_k, h_k) = (\sigma_\ell, h_\ell)$, *let* $\delta_{k,\ell} = (J_i)_{0 \le i \le n}$ *be the* (H_δ^ℓ, μ)*-contraction of* δ *with* μ *being the* $(\mathsf{dom}(H_\delta^k) \cap \mathsf{dom}(H_\delta^\ell))$*-isomorphism from* H_δ^ℓ *to* H_δ^k. *The following hold:*

1. $\delta_{k,\ell}$ *is a quasi chase derivation with total applications of* D *w.r.t.* Σ *such that, for each* $i \in \{0, 1, \ldots, n-1\}$, *there is a homomorphism* μ_i *from* $\mathsf{body}(\sigma_i)$ *to* J_i *such that* $J_i \langle \sigma_i, \mu_i \rangle J_{i+1}$.
2. *There is no* $\alpha \in J_n$ *such that* $H_\delta^\ell \prec_{\delta_{k,\ell}}^{p,+} \alpha$.
3. *For each* $i, j \in \{0, 1, \ldots, n-1\}$, $(\sigma_i, h_i) = (\sigma_j, h_j)$ *iff* $(\sigma_i, \mu_i) = (\sigma_j, \mu_j)$.
4. $\bar{c} \in q(J_n \setminus (H_\delta^\ell \setminus H_\delta^k))$.

Proof. For each $i \in \{0, 1, \ldots, n\}$, we define the sets of atoms

$$I_i^{in} = \left\{\alpha \in I_i \mid H_\delta^\ell \prec_\delta^{p,+} \alpha\right\} \quad \text{and} \quad I_i^{out} = I_i \setminus I_i^{in}.$$

In simple words, I_i^{in} are the atoms of I_i reachable by an atom from H_δ^ℓ, and thus, those atoms are updated by μ during the contraction. By construction,

$$\begin{aligned} J_i &= \mu(I_i^{in}) \cup I_i^{out} \\ &= \mu(I_i^{in}) \cup \mu(I_i^{out}) \cup (H_\delta^\ell \cap I_i^{out}) \\ &= \mu(I_i) \cup (H_\delta^\ell \cap I_i^{out}), \end{aligned}$$

which implies that $\mu(I_i) \subseteq J_i$, for each $i \in \{0, 1, \ldots, n\}$. We also define, for each $i \in \{0, 1, \ldots, n\}$, the function

$$\mu_i \ = \ \mu \circ h_i.$$

We proceed to prove the four statements claimed in Lemma 3.1:

1. We need to show that, for each $i \in \{0, \ldots, n-1\}$, $J_i \langle \sigma_i, \mu_i \rangle J_{i+1}$, i.e., $J_{i+1} = J_i \cup \mu_i'(\mathsf{head}(\sigma_i))$ for some $\mu_i' \supseteq \mu_i$. We first show that (σ_i, μ_i) is indeed a trigger for Σ on J_i. We know that $h_i(\mathsf{body}(\sigma_i)) \in I_i$. Since $\mu(I_i) \subseteq J_i$, we get that $\mu(h_i(\mathsf{body}(\sigma_i))) \in J_i$. Since, by definition, $\mu_i = \mu \circ h_i$, we get that (σ_i, μ_i) is a trigger for Σ on J_i. It remains to show that there exists $\mu_i' \supseteq \mu_i$ such that $J_{i+1} = J_i \cup \mu_i'(\mathsf{head}(\sigma_i))$. By hypothesis, we know that there exists $h_i' \supseteq h_i$ such that $I_{i+1} = I_i \cup h_i'(\mathsf{head}(\sigma_i))$. Since $\mu(I_{i+1}) \subseteq J_{i+1}$, we conclude that $\mu(h_i'(\mathsf{head}(\sigma_i))) \subseteq J_{i+1}$. Since μ is an isomorphism, it is clear that $\mu_i' = \mu \circ h_i'$ is an extension of μ_i, and the claim follows.
2. This is a consequence of the contraction operator and the triggers defined above. In fact, via an easy induction, it is possible to show, by following the new triggers, that all the atoms in $\mu(I_n^{in})$ are now descendants of H_δ^k. (Note that each atom in $\mu(I_n^{in})$ contains no term of $\mathsf{dom}(H_\delta^\ell) \setminus (\mathsf{dom}(H_\delta^k) \cap \mathsf{dom}(H_\delta^\ell))$ since μ replaces those terms with terms of $\mathsf{dom}(H_\delta^k) \setminus (\mathsf{dom}(H_\delta^k) \cap \mathsf{dom}(H_\delta^\ell))$.)
3. This statement is a direct consequence of the fact that $\mu_i = \mu \circ h_i$, $\mu_j = \mu \circ h_j$, and that μ is an isomorphism.
4. By hypothesis, $q(\bar{c}) \to I_n$ due to a homomorphism h. This implies that $\mu \circ h$ maps $q(\bar{c})$ to J_n. Observe that $\mu(I_n) \cap (H_\delta^\ell \setminus H_\delta^k) = \varnothing$, which immediately implies that $\mu \circ h$ maps $q(\bar{c})$ to $J_n \setminus (H_\delta^\ell \setminus H_\delta^k)$, as needed.

This completes the proof of Lemma 3.1. □

We can now explain how the direction (2) implies (1) of Proposition 3.1 is shown. Assume that $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$. By hypothesis, there exists an $f_{\mathbb{L}}(\Sigma, q)$-parsimonious quasi chase derivation with total applications $\delta = (I_i)_{0 \le i \le n}$ of D w.r.t. Σ such that $\bar{c} \in q(I_n)$. By iteratively applying Lemma 3.1, we can eventually construct an $f_{\mathbb{L}}(\Sigma, q)$-parsimonious quasi chase derivation with total applications $\delta' = (J_i)_{0 \le i \le n}$ of D w.r.t. Σ with $J_i \langle \sigma_i, h_i \rangle J_{i+1}$ such that, for each $0 < j < n$ for which there exists $0 \le i < j$ with $(\sigma_i, h_i) = (\sigma_j, h_j)$:

- there is no $\alpha \in J_n$ such that $H_\delta^j \prec_{\delta'}^{p,+} \alpha$, and
- $\bar{c} \in q(J_n \setminus (H_\delta^j \setminus H_\delta^i))$.

Therefore, we can drop the applications in δ' that use a trigger that has been applied before in order to obtain $\delta'' = (K_i)_{0 \le i \le m}$, where $m \le f_{\mathbb{L}}(\Sigma, q)$, that is a prefix of a chase derivation of D w.r.t. Σ and $\bar{c} \in q(K_m)$, as needed.

3.3. Ontological query answering via parsimonious quasi chase derivations

We now proceed with the second step of the proof of Theorem 3.1 and show that, for the class of linear TGDs, the fact that a tuple is a certain answer can be witnessed via a k-parsimonious quasi chase derivation, where k is given by the function $f_{\mathbb{L}}$.

Proposition 3.2. *Consider a database D, a set $\Sigma \in \mathbb{L}$, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathbf{C}^{|\bar{x}|}$. If $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$, then there exists an $f_{\mathbb{L}}(\Sigma, q)$-parsimonious quasi chase derivation with total applications $(J_i)_{0 \le i \le n}$ of D w.r.t. Σ such that $\bar{c} \in q(J_n)$.*

To establish the above result it suffices to show that the prefix $(I_i)_{0 \le i \le n}$ of some chase derivation of D w.r.t. Σ that witnesses the fact that $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$, i.e., $\bar{c} \in q(I_n)$, can be converted into an $f_{\mathbb{L}}(\Sigma, q)$-parsimonious quasi chase derivation with total applications $(J_i)_{0 \le i \le n}$ of D w.r.t. Σ such that $\bar{c} \in q(J_n)$. To this end, we can rely again on iterative contractions as we explain next.

Consider two S-isomorphic atoms α and β of I_n, where α does not appear in the subtree rooted at β, and S is the set of terms occurring in $h(q(\bar{c}))$ with h being a homomorphism that maps $q(\bar{c})$ to I_n. Due to linearity, we can safely move the subtree rooted at β under α, and consistently update its atoms, without affecting the fact that $\bar{c}$ is a certain answer. We can achieve this via the (β, μ)-contraction of $(I_i)_{0 \le i \le n}$ with μ being the S-isomorphism from β to α.[7] Notice, however,

[7] For a singleton instance $\{\gamma\}$ we simply write γ.

that such a contraction may create partial applications and repeated triggers. Hence, the result is not necessarily a prefix of a chase derivation, but a quasi chase derivation. This is formalized by the following technical lemma.

Lemma 3.2. *Consider a database D, a set $\Sigma \in \mathbb{L}$, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathbf{C}^{|\bar{x}|}$. Let $\delta = (I_i)_{0 \le i \le n}$ be a quasi chase derivation of D w.r.t. Σ such that $q(\bar{c}) \to I_n$ via h. For atoms $\alpha, \beta \in I_n$ such that $\beta \not\prec_\delta^{p,+} \alpha$ and $\alpha \simeq_S \beta$, where $S = \mathsf{dom}(h(q(\bar{c})))$, let $\delta_{\alpha,\beta} = (J_i)_{0 \le i \le n}$ be the (β, μ)-contraction of δ with μ being the S-isomorphism from β to α. The following hold:*

1. *$\delta_{\alpha,\beta}$ is a quasi chase derivation of D w.r.t. Σ.*
2. *There is no $\gamma \in J_n$ such that $\beta \prec_{\delta_{\alpha,\beta}}^{p,+} \gamma$.*
3. *For each $\gamma, \gamma' \in J_n$, $\gamma \simeq_S \gamma'$ iff $\mu^{-1}(\gamma) \simeq_S \mu^{-1}(\gamma')$.*
4. *$q(\bar{c}) \to J_n$ via h.*

Proof. For each $i \in \{0, 1, \ldots, n\}$, we define the sets of atoms

$$I_i^{in} = \left\{ \gamma \in I_i \mid \beta \prec_\delta^{p,+} \gamma \right\} \quad \text{and} \quad I_i^{out} = I_i \setminus I_i^{in}.$$

In simple words, I_i^{in} are the atoms of I_i reachable from β, and thus, those atoms are updated by μ during the contraction. Moreover, we define the set of integers

$$X = \{i \mid h_i(\mathsf{body}(\sigma_i)) = \beta\} \cup \{i \mid \beta \prec_\delta^{p,+} h_i(\mathsf{body}(\sigma_i))\}.$$

We finally define, for each $i \in \{0, 1, \ldots, n\}$, the function

$$\mu_i = \begin{cases} \mu \circ h_i & \text{if } i \in X \\ h_i & \text{otherwise.} \end{cases}$$

We proceed to show the four statements of Lemma 3.2.

1. We show that, for each $i \in \{0, 1, \ldots, n\}$, $J_i \langle \sigma_i, \mu_i \rangle^p J_{i+1}$. By hypothesis, there exists $h_i' \supseteq h_i$ such that $I_{i+1} = I_i \cup h_i'(\mathsf{head}(\sigma_i))$. We proceed by case analysis:
   - $\dfrac{h_i(\mathsf{body}(\sigma_i)) \in I_i^{out} \text{ and } h_i(\mathsf{body}(\sigma_i)) \neq \beta}{}$

     Since $i \notin X$, $\mu_i = h_i$. Since $I_i^{out} \subseteq J_i$, we get that $\mu_i(\mathsf{body}(\sigma_i)) \in J_i$, and thus, (σ_i, μ_i) is a trigger for Σ on J_i. Since $J_{i+1} \setminus J_i = I_{i+1} \setminus I_i$ whenever $i \notin X$, we conclude that $J_i \langle \sigma_i, \mu_i \rangle^p J_{i+1}$ due to $h_i' \supseteq \mu_i$.
   - $\dfrac{h_i(\mathsf{body}(\sigma_i)) \in I_i^{out} \text{ and } h_i(\mathsf{body}(\sigma_i)) = \beta}{}$

     Since $i \in X$, $\mu_i = \mu \circ h_i$. Clearly, $\mu(\beta) = \mu(h_i(\mathsf{body}(\sigma_i))) = \alpha$. Since $\alpha \in I_i^{out}$ and $I_i^{out} \subseteq J_i$, we have that $\mu_i(\mathsf{body}(\sigma_i)) \in J_i$, and thus, (σ_i, μ_i) is a trigger for Σ on J_i. Since $J_{i+1} \setminus J_i = \mu^{-1}(I_{i+1} \setminus I_i)$ whenever $i \in X$, we conclude that $J_i \langle \sigma_i, \mu_i \rangle^p J_{i+1}$ due to $\mu \circ h_i' \supseteq \mu_i$.
   - $\dfrac{h_i(\mathsf{body}(\sigma_i)) \in I_i^{in}}{}$

     Since $\mu(I_i^{in}) \subseteq J_i$, $\mu(h_i(\mathsf{body}(\sigma_i))) = \mu_i(\mathsf{body}(\sigma_i)) \subseteq J_i$, and thus, (σ_i, μ_i) is a trigger for Σ on J_i. Since $J_{i+1} \setminus J_i = \mu(I_{i+1} \setminus I_i)$ whenever $i \in X$, we get that $J_i \langle \sigma_i, \mu_i \rangle^p J_{i+1}$ due to $\mu \circ h_i' \supseteq \mu_i$.
2. This is a consequence of the contraction operator and the triggers defined above. Actually, via an easy induction, it is possible to show, by following the new triggers, that all the atoms in $\mu(I_n^{in})$ are now descendants of α. (Note that each atom in $\mu(I_n^{in})$ contains no term of $\mathsf{dom}(\beta) \setminus S$ since μ replaces those terms with terms of $\mathsf{dom}(\alpha) \setminus S$.)
3. This statement follows from the fact that μ^{-1} is an S-isomorphism.
4. This holds since $\{\gamma \in J_n \mid \mathsf{dom}(\gamma) \subseteq S\} = \{\gamma \in I_n \mid \mathsf{dom}(\gamma) \subseteq S\}$.

This completes the proof of Lemma 3.2. □

We are now ready to explain how Proposition 3.2 is shown. By hypothesis, there exists a prefix $\delta = (I_i)_{0 \le i \le n}$, for $n \ge 0$, of a chase derivation of D w.r.t. Σ such that $q(\bar{c}) \to I_n$ via a homomorphism h. By definition, δ is a quasi chase derivation of D w.r.t. Σ. The binary relation $\simeq_S$ over I_n, where $S = \mathsf{dom}(h(q(\bar{c})))$, is an equivalence relation. Let $I_n/\simeq_S$ be the set of all equivalence classes in I_n w.r.t. $\simeq_S$. For each equivalence class C in $I_n/\simeq_S$, its canonical atom is arbitrarily chosen from the set of atoms $\{\alpha \in C \mid \text{ there is no } \beta \in C \text{ such that } \beta \prec_\delta^{p,+} \alpha\}$. For an atom $\alpha \in I_n$, we write $[\alpha]$ for the canonical atom of its equivalence class, and ι_α for the S-isomorphism that maps α to $[\alpha]$. We proceed to construct an $f_{\mathbb{L}}(\Sigma, q)$-parsimonious quasi chase derivation with total applications $\delta' = (J_i)_{0 \le i \le n}$ of D w.r.t. Σ such that $\bar{c} \in q(J_n)$.

Let $\alpha \in I_n$ with $\alpha \neq [\alpha]$. By Lemma 3.2, the (α, ι_α)-contraction $\delta_\alpha = (I_i^\alpha)_{0 \le i \le n}$ of δ enjoys the following properties:

- δ_α is a quasi chase derivation of D w.r.t. Σ,
- there is no $\beta \in J_n$ such that $\alpha \prec_{\delta_\alpha}^{p,+} \beta$,

- for each $\beta, \beta' \in J_n$, $\beta \simeq_S \beta'$ iff $\iota_\alpha^{-1}(\beta) \simeq_S \iota_\alpha^{-1}(\beta')$, and
- $q(\bar{c}) \to J_n$ via h.

We can therefore iteratively apply Lemma 3.2 as discussed above, until we get a quasi chase derivation $\delta^\diamond = (I_i^\diamond)_{0 \le i \le n}$ of D w.r.t. Σ such that the following hold:

1. For each $\alpha \in I_n^\diamond$, $\alpha \neq [\alpha]$ implies there is no $\beta \in I_n^\diamond$ such that $\alpha \prec_{\delta^\diamond} \beta$.
2. $q(\bar{c}) \to I_n^\diamond$ via h.

Note that $\delta^\diamond$ is not necessarily with total applications. However, it can be converted into one with total applications by simply adding the atoms that are needed in order to ensure that every application in $\delta^\diamond$ is total. Let $\delta' = (J_i)_{0 \le i \le n}$ be the resulted quasi chase derivation with total applications of D w.r.t. Σ. It is clear that $q(\bar{c}) \to J_n$ via h, and thus $\bar{c} \in q(I_n)$. It remains to show that δ' is $f_{\mathbb{L}}(\Sigma, q)$-parsimonious.

Since in δ' only canonical atoms trigger TGDs (the first property of $\delta^\diamond$ stated above, which is inherited by δ'), it suffices to count how many triggers for Σ on the instance

$$K \;=\; \{\alpha \mid \alpha \text{ is the canonical atom of a set } C \in J_n/\!\simeq_S\}$$

can be formed. Due to linearity, we can assume, without loss of generality, that in δ' at most $|q|$ atoms from D trigger a TGD of Σ, which implies that K contains at most $|q|$ atoms of D. Therefore, assuming that Σ and q are both over the schema $\mathbf{S}$,

$$|K| \;\le\; |\mathbf{S}| \cdot \big(2 \cdot |q| \cdot \mathsf{ar}(\mathbf{S}) + \mathsf{ar}(\mathbf{S})\big)^{\mathsf{ar}(\mathbf{S})},$$

which is the number of non-S^+-isomorphic atoms over $\mathbf{S}$ that can be formed, where S^+ consists of the set S and the set of constants T occurring in the atoms of D that trigger a TGD. Actually, the above upper bound is a consequence of the fact that $|S| \le |q| \cdot \mathsf{ar}(\mathbf{S})$ and $|T| \le |q| \cdot \mathsf{ar}(\mathbf{S})$. Since each atom of K can trigger several TGDs of Σ, the total number of triggers for Σ on K that can be formed is $f_{\mathbb{L}}(\Sigma, q)$, as needed.

Theorem 3.1 follows from Propositions 3.1 and 3.2.

3.4. Bounded witness property and linear TGDs in normal form

In the proofs of Section 4, where we show that the class of linear TGDs is polynomially combined first-order rewritable, it will be technically convenient to work with linear TGDs in normal form, i.e., TGDs with only one atom in the head that contains at most one occurrence of an existentially quantified variable that appears at the last position of the head-atoms. To this end, we establish a result similar to Theorem 3.1 for linear TGDs in normal form. Let us first recall the normalization procedure.

Consider a linear TGD σ over a schema $\mathbf{S}$ of the form

$$P(\bar{x}, \bar{y}) \;\to\; \exists z_1 \cdots \exists z_m \, R_1(\bar{x}_1, \bar{z}_1), \ldots, R_n(\bar{x}_n, \bar{z}_n),$$

where $n, m \ge 0$, $\bar{x}_i \subseteq \bar{x}$ and $\bar{z}_i \subseteq \{z_1, \ldots, z_m\}$ for each $i \in [m]$, $\bar{x} = \bigcup_{i \in [n]} \bar{x}_i$, and $\{z_1, \ldots, z_m\} = \bigcup_{i \in [n]} \bar{z}_i$. We define $\mathsf{N}(\sigma)$ as the set of linear TGDs consisting of

$$\begin{aligned}
P(\bar{x}, \bar{y}) &\to \exists z_1 \, \mathsf{Aux}_\sigma^1(\bar{x}, z_1) \\
\mathsf{Aux}_\sigma^1(\bar{x}, z_1) &\to \exists z_2 \, \mathsf{Aux}_\sigma^2(\bar{x}, z_1, z_2) \\
&\vdots \\
\mathsf{Aux}_\sigma^{m-1}(\bar{x}, z_1, \ldots, z_{m-1}) &\to \exists z_m \, \mathsf{Aux}_\sigma^m(\bar{x}, z_1, \ldots, z_m) \\
\mathsf{Aux}_\sigma^m(\bar{x}, z_1, \ldots, z_m) &\to R_1(\bar{x}_1, \bar{z}_1) \\
&\vdots \\
\mathsf{Aux}_\sigma^m(\bar{x}, z_1, \ldots, z_m) &\to R_n(\bar{x}_n, \bar{z}_n),
\end{aligned}$$

where the auxiliary predicates $\mathsf{Aux}_\sigma^1, \ldots, \mathsf{Aux}_\sigma^m$ do not belong to the schema $\mathbf{S}$. Given a set Σ of linear TGDs, we define the set $\mathsf{N}(\Sigma)$ as the set $\bigcup_{\sigma \in \Sigma} \mathsf{N}(\sigma)$. The next easy lemma collects some useful facts concerning the normalization procedure $\mathsf{N}(\cdot)$:

Lemma 3.3. *Consider a set $\Sigma \in \mathbb{L}$ over a schema $\mathbf{S}$. The following hold:*

1. $\mathsf{N}(\Sigma)$ *can be computed in polynomial time in the size of* Σ.

2. *Given a database D and a CQ $q(\bar{x})$ over* $\mathbf{S}$, $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(q, D, \mathsf{N}(\Sigma))$.

Note that item (2) of Lemma 3.3 holds due to the fact that the auxiliary predicates introduced during the normalization do not occur in D or q. Recall that our goal is to establish a result similar to Theorem 3.1 for the class of linear TGDs in normal form, more precisely, the class of TGDs $\mathbb{L}_\mathsf{N} = \{\mathsf{N}(\Sigma) \mid \Sigma \in \mathbb{L}\}$. Of course, it should not be surprising that $\mathbb{L}_\mathsf{N}$ enjoys the bounded witness property due to $f_\mathbb{L}$ since $\mathbb{L}_\mathsf{N} \subseteq \mathbb{L}$; this is an immediate consequence of Theorem 3.1. Thus, the goal of this new result is not to simply show that $\mathbb{L}_\mathsf{N}$ enjoys the bounded witness property, but to show that the property holds due to a function that provides a bound w.r.t. the original set of TGDs before the normalization. Since the normalization procedure does not preserve the arity of the predicates,[8] this result will be particularly important for showing in the next section that in the case of schemas of *fixed arity*, the class of linear TGDs is polynomially combined $\exists\mathrm{FO}^+$-rewritable.

It is not difficult to verify that, for every pair of sets of TGDs $\Sigma_1, \Sigma_2 \in \mathbb{L}$, $\mathsf{N}(\Sigma_1) = \mathsf{N}(\Sigma_2)$ implies $\Sigma_1 = \Sigma_2$; this is actually a consequence of the normalization procedure. Therefore, given a set $\Sigma \in \mathbb{L}_\mathsf{N}$ of TGDs, there exists a *unique* set of TGDs from $\mathbb{L}$, which we denote by Σ^-, such that $\Sigma = \mathsf{N}(\Sigma^-)$. In simple words, Σ^- is the (unique) set of linear TGDs that we obtain after "denormalizing" Σ. We also write $\mathsf{exvar}(\sigma)$ for the set of existentially quantified variables occurring in a TGD σ. We define the function $f_{\mathbb{L}_\mathsf{N}}$ that maps $\mathbb{L}_\mathsf{N} \times \mathbb{CQ}$ to the natural numbers as follows:

$$f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) = \left(\max_{\sigma \in \Sigma^-}\{|\mathsf{exvar}(\sigma)|\} + \max_{\sigma \in \Sigma^-}\{|\mathsf{head}(\sigma)|\}\right) \cdot f_\mathbb{L}(\Sigma^-, q)$$

It is clear that $f_{\mathbb{L}_\mathsf{N}}$ is computable since $f_\mathbb{L}$ is computable. We proceed to show that:

Theorem 3.2. *$\mathbb{L}_\mathsf{N}$ enjoys the bounded witness property due to $f_{\mathbb{L}_\mathsf{N}}$.*

Proof. Consider a database D, a set $\Sigma \in \mathbb{L}_\mathsf{N}$, a CQ $q(\bar{x})$, and a tuple $c \in \mathbf{C}^{|\bar{x}|}$. We need to show that $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$ implies the existence of a sequence $(I_i)_{0 \le i \le n}$, with $n \le f_{\mathbb{L}_\mathsf{N}}(\Sigma, q)$, that is a prefix of a chase derivation of D w.r.t. Σ, and $\bar{c} \in q(I_n)$. By Lemma 3.3, we get that $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(q, D, \Sigma^-)$. Since, by construction, $\Sigma^- \in \mathbb{L}$, by Theorem 3.1 we conclude the following: $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$ implies the existence of a sequence $(J_i)_{0 \le i \le m}$, with $m \le f_\mathbb{L}(\Sigma^-, q)$, that is a prefix of a chase derivation of D w.r.t. Σ^-, and $\bar{c} \in q(J_n)$. Our goal is to convert $(J_i)_{0 \le i \le m}$ into the desired sequence $(I_i)_{0 \le i \le n}$, with $n \le f_{\mathbb{L}_\mathsf{N}}(\Sigma, q)$, that is a prefix of a chase derivation of D w.r.t. Σ, and $\bar{c} \in q(I_n)$, which in turn shows our claim.

Assume that $J_j\langle\sigma_j, h_j\rangle J_{j+1}$, for $j \in \{0, \ldots, m-1\}$, with σ_j being of the form

$$P(\bar{x}, \bar{y}) \rightarrow \exists z_1 \cdots \exists z_{k_j}\, R_1(\bar{x}_1, \bar{z}_1), \ldots, R_{\ell_j}(\bar{x}_{\ell_j}, \bar{z}_{\ell_j}).$$

We also assume that $J_{j+1} = J_j \cup h'_j(\mathsf{head}(\sigma_j))$, i.e., h'_j is the extension of h_j that has been employed during the application of (σ_j, h_j). We define the sequence of instances

$$I_0^0, I_0^1, \ldots, I_0^{k_0+\ell_0-1}, I_1^0, I_1^1, \ldots, I_1^{k_1+\ell_1-1}, \ldots, I_{m-1}^0, I_{m-1}^1, \ldots, I_{m-1}^{k_{m-1}+\ell_{m-1}-1}, I_m^0,$$

where

- for each $j \in \{0, \ldots, m\}$, $I_j^0 = J_j$,
- for each $j \in \{0, \ldots, m-1\}$ and $i \in \{1, \ldots, k_j\}$,

$$I_j^i = I_j^{i-1} \cup \left\{h'_j(\mathsf{Aux}^i_{\sigma_j}(\bar{x}, z_1, \ldots, z_i))\right\},$$

- for each $j \in \{0, \ldots, m-1\}$ and $i \in \{k_j+1, \ldots, k_j+\ell_j-1\}$,

$$I_j^i = I_j^{i-1} \cup \left\{h'_j(R_{i-k_j}(\bar{x}_i - k_j, \bar{z}_{i-k_j}))\right\}.$$

We proceed to show that the above is indeed the desired sequence of instances. For $j \in \{0, \ldots, m-1\}$, let $\sigma_j^1, \ldots, \sigma_j^{k_j+\ell_j}$ be the linear TGDs of $\mathsf{N}(\sigma_j)$, i.e., the TGDs obtained after normalizing σ_j, in the order that are presented in the definition of $\mathsf{N}(\sigma_j)$. In other words, for $i \in \{1, \ldots, k_j\}$, σ_j^i is the TGD with head $\mathsf{Aux}^i_{\sigma_j}(\bar{x}, z_1, \ldots, z_i)$, and for $i \in \{k_j+1, \ldots, k_j+\ell_j\}$, σ_j^i is the TGD with head $R_{i-k_j}(\bar{x}_{i-k_j}, \bar{z}_{i-k_j})$. It is not difficult to verify that, for each $j \in \{0, \ldots, m-1\}$, the following hold:

- $I_j^0\langle\sigma_j^1, h_j\rangle I_j^1$,
- for each $i \in \{1, \ldots, k_j-1\}$, $I_j^i\langle\sigma_j^{i+1}, h_j^{i+1}\rangle I_j^{i+1}$ with h_j^{i+1} being the restriction of h'_j over the variables $\bar{x} \cup \{z_1, \ldots, z_i\}$,

[8] In fact, even if we consider a set of TGDs over a schema of fixed arity, the normalized set will, in general, contain predicates of unbounded arity (see the auxiliary predicates).

- for each $i \in \{k_j, \ldots, k_j - \ell_j - 2\}$, $I_j^i\langle\sigma_j^{i+1}, h'_j\rangle I_j^{i+1}$, and
- $I_j^{k_j+\ell_j-1}\langle\sigma_j^{k_j+\ell_j}, h'_j\rangle I_{j+1}^0$.

Therefore, the sequence of instances defined above is a prefix of a chase derivation of D w.r.t. Σ^-, and $\bar{c} \in q(I_m^0)$. It remains to show that

$$\left(\max_{j\in\{0,\ldots,m-1\}}\{k_j\} + \max_{j\in\{0,\ldots,m-1\}}\{\ell_j\}\right)\cdot m \;\leq\; f_{\mathbb{L}_\mathsf{N}}(\Sigma, q).$$

Observe that

$$\begin{aligned}\max_{j\in\{0,\ldots,m-1\}}\{k_j\} &\leq \max_{\sigma\in\Sigma^-}\{|\mathsf{exvar}(\sigma)|\},\\ \max_{j\in\{0,\ldots,m-1\}}\{\ell_j\} &\leq \max_{\sigma\in\Sigma^-}\{|\mathsf{head}(\sigma)|\},\\ m &\leq f_{\mathbb{L}}(\Sigma^-, q),\end{aligned}$$

and the claim follows. □

4. Polynomial combined rewritability and linearity

We proceed to study the notion of polynomial combined rewritability in the case of linear TGDs. We say that a family of schemas $\mathbb{S}$ is of fixed arity if there exists an integer $k \geq 0$ such that, for every schema $\mathbf{S} \in \mathbb{S}$, it holds that $\mathsf{ar}(\mathbf{S}) \leq k$.

Theorem 4.1. *The following hold:*

1. $\mathbb{L}$ *is polynomially combined* FO*-rewritable.*
2. *For a family of schemas* $\mathbb{S}$ *of fixed arity,* $\mathbb{L}[\mathbb{S}]$ *is polynomially combined* $\exists\mathrm{FO}^+$*-rewritable.*

Since the evaluation problem for $\exists\mathbb{FO}^+$, that is, given a database D, an $\exists\mathrm{FO}^+$ query $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$, decide whether $\bar{c} \in q(D)$, is in NP, an immediate consequence of Theorem 4.1 is the following complexity result, which, as discussed in the Introduction, closes a complexity gap for the problem $\mathsf{OQA}(\mathbb{L})$ in the case of schemas of fixed arity. In fact, the following result is known for single-head (i.e., only one atom in the head) linear TGDs [23], but it cannot be straightforwardly transferred to multi-head (i.e., more than one atom in the head) linear TGDs.

Corollary 4.1. $\mathit{OQA}(\mathbb{L})$ *is* NP*-complete for schemas of fixed arity.*

The proof of Theorem 4.1 heavily relies on Proposition 4.1 below, which is actually the main technical result of this section. For a database D, we define

$$D_{01} \;=\; D \cup \{\mathsf{Zero}(0), \mathsf{One}(1)\},$$

where 0 and 1 are new constants not in $\mathsf{dom}(D)$, while Zero and One are new predicates not mentioned in D. In other words, D_{01} is obtained by adding to the domain of D the new constants 0 and 1 that we can access via the unary predicates Zero and One, respectively. Moreover, for a family of schemas $\mathbb{S}$, we write $\mathbb{L}_\mathsf{N}^\mathbb{S}$ for the class

$$\{\Sigma \in \mathbb{L}_\mathsf{N} \mid \Sigma^- \in \mathbb{L}[\mathbb{S}]\},$$

which essentially collects all the sets of linear TGDs in normal form that after "denormalizing" them we get a set of linear TGDs over a schema of $\mathbb{S}$.

Proposition 4.1. *The following hold:*

1. *There is a polynomial-time computable query rewriter* $f_\mathsf{Q} : \mathbb{CQ} \times \mathbb{L}_\mathsf{N} \to \mathbb{FO}$ *such that, for every database* D*, set* $\Sigma \in \mathbb{L}_\mathsf{N}$ *of TGDs, and CQ* q*,* $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ *with* $q_\Sigma = f_\mathsf{Q}(q, \Sigma)$*.*
2. *For a family of schemas* $\mathbb{S}$ *of fixed arity, there is a polynomial-time computable query rewriter* $f_\mathsf{Q}^\mathbb{S} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{L}_\mathsf{N}^\mathbb{S} \to \exists\mathbb{FO}^+$ *such that, for every* $D \in \mathbb{D}[\mathbb{S}]$*,* $\Sigma \in \mathbb{L}_\mathsf{N}^\mathbb{S}$*, and* $q \in \mathbb{CQ}[\mathbb{S}]$*,* $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ *with* $q_\Sigma = f_\mathsf{Q}^\mathbb{S}(q, \Sigma)$*.*

Before giving the proof of Proposition 4.1, let us explain how we get Theorem 4.1 by exploiting Proposition 4.1. For item (1), we need to show that there are polynomial-time computable database and query rewriters g_DB and g_Q, respectively,

such that, for every database D, set $\Sigma \in \mathbb{L}$, and CQ q, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_\Sigma)$ with $D_\Sigma = g_{\mathsf{DB}}(D, \Sigma)$ and $q_\Sigma = g_{\mathsf{Q}}(q, \Sigma)$. We define the database rewriter

$$g_{\mathsf{DB}}(D, \Sigma) \;=\; D_{01},$$

which is clearly computable in constant time,[9] and the query rewriter

$$g_{\mathsf{Q}}(q, \Sigma) \;=\; f_{\mathsf{Q}}(q, \mathsf{N}(\Sigma))$$

with f_{Q} being the polynomial-time computable rewriter provided by item (1) of Proposition 4.1; recall that $\mathsf{N}(\Sigma) \in \mathbb{L}_{\mathsf{N}}$ is the normalized version of Σ. Since, by Lemma 3.3, $\mathsf{N}(\Sigma)$ is computable in polynomial time, g_{Q} is also computable in polynomial time. Moreover, by item (1) of Proposition 4.1, $\mathsf{cert}(q, D, \mathsf{N}(\Sigma)) = q_\Sigma(D_\Sigma)$ with $D_\Sigma = g_{\mathsf{DB}}(D, \Sigma)$ and $q_\Sigma = g_{\mathsf{Q}}(q, \Sigma)$. By Lemma 3.3, $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(q, D, \mathsf{N}(\Sigma))$, and thus, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_\Sigma)$, as needed. Item (2) of Theorem 4.1 is shown via a similar argument, but relying on item (2) of Proposition 4.1.

The rest of the section is devoted to showing Proposition 4.1. Here is a roadmap of the rather long proof.

- We start in Section 4.1 by introducing our main technical tool, the so-called witness generator, which formalizes the intuitive idea that for linear TGDs, ontological query answering can be realized as a reachability problem on the directed graph (which is actually a forest due to linearity) that is formed by the parent relation of a chase derivation. In particular, given a database D, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$, we characterize the fact that $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$ via the existence of a witness generator for $q(\bar{c})$ w.r.t. D and Σ.
- The above characterization essentially tells us that the desired query rewriters should build first-order queries that check whether, on an input database D, there exists a witness generator for $q(\bar{d})$, with $\bar{d} \in \mathsf{dom}(D)^{|\bar{x}|}$ being a candidate certain answer, w.r.t. D and Σ. To this end, we need to encode such a witness generator in a way that a first-order query can talk about it. This is done in Section 4.2 by representing predicates and nulls via binary strings, i.e., strings consisting of 0 and 1, which in turn leads to the notion of binary witness generator.
- We then proceed in Section 4.3 and 4.4 to establish item (1) and (2), respectively, of Proposition 4.1 by constructing the desired query rewriters.

4.1. Witness generator

Before delving into the formal definitions, let us first illustrate the key ideas underlying the notion of witness generator via a simple example.

Example 4.1. Consider the database

$$D \;=\; \{P(a, b, c), P(b, c, d)\}$$

the set Σ consisting of the linear TGDs

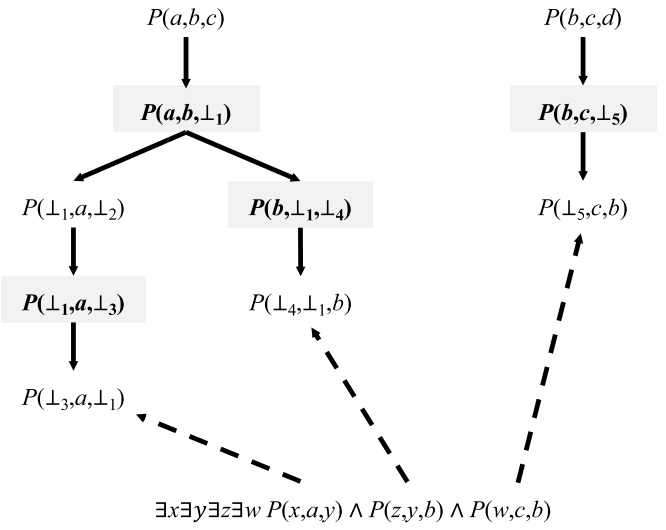
$$\beta \to \exists w\, P(x, y, w), \quad \beta \to \exists w\, P(z, x, w), \quad \beta \to P(z, y, x), \quad \beta \to \exists w\, P(y, z, w),$$

where $\beta = P(x, y, z)$, and the Boolean CQ

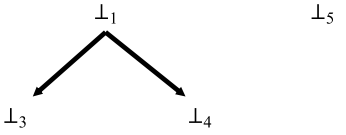
$$q \;=\; \exists x \exists y \exists z \exists w\, P(x, a, y) \wedge P(z, y, b) \wedge P(w, c, b).$$

As illustrated in the following figure, there is a homomorphism h (dashed arrows in the figure) that maps q to an initial segment of the forest formed due to the parent relation of a chase derivation δ of D w.r.t. Σ, and thus, $\mathsf{cert}(q, D, \Sigma) \neq \varnothing$.

[9] Notice that the result of $g_{\mathsf{DB}}(D, \Sigma)$ does not depend on Σ; we simply add to D the constants 0 and 1.



It is interesting to observe that the nulls occurring in $h(q)$, namely the nulls $\bot_1$, $\bot_3$, $\bot_4$ and $\bot_5$, form the rooted forest F



with the following properties; let $\nu(\bot)$ be the atom of $\mathsf{chase}_\delta(D, \Sigma)$, where the null $\bot$ has been invented (see shaded atoms above for $\nu(\bot_1)$, $\nu(\bot_3)$, $\nu(\bot_4)$, and $\nu(\bot_5)$):

- for every root node $\bot$, $\nu(\bot)$ is reachable from D,
- for every edge $(\bot, \bot')$, $\nu(\bot')$ is reachable from $\nu(\bot)$, and
- for every atom $\alpha \in h(q)$, there is a unique path π in F that contains all the nulls in α, and, assuming that the leaf node of π is $\bot$, α is reachable from $\nu(\bot)$.

Indeed, it is easy to see that $\nu(\bot_1)$ and $\nu(\bot_5)$ are reachable from D, $\nu(\bot_3)$ and $\nu(\bot_4)$ are reachable from $\nu(\bot_1)$, and finally, $h(P(x, a, y)) = P(\bot_3, a, \bot_1)$ is reachable from $\nu(\bot_3)$, $h(P(z, y, b)) = P(\bot_4, \bot_1, b)$ is reachable from $\nu(\bot_4)$, and $h(P(w, c, b)) = P(\bot_5, c, b)$ is reachable from $\nu(\bot_5)$. Roughly speaking, the triple consisting of

1. the homomorphism h, which maps q to $\mathsf{chase}_\delta(D, \Sigma)$,
2. the function ν, which gives the atoms of $\mathsf{chase}_\delta(D, \Sigma)$ where the nulls occurring in $h(q)$ were invented, and
3. the forest F, which encodes how the nulls of $h(q)$ depend on each other, as well as the order of their generation,

is what we call a witness generator for q w.r.t. D and Σ. The existence of such a triple allows us to generate a prefix of the chase derivation δ, i.e., a witness (hence the name "witness generator"), that entails the query q. □

Let us now formalize the intuition underlying the notion of witness generator described in Example 4.1. To this end, we first need to introduce some auxiliary notions.

Rooted Forests. A *rooted tree* T over a set V is a directed tree with V being the node set of T, where one node, denoted $\mathsf{root}(T)$, is 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 V 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 over a set V is a collection of rooted trees $T_1, \ldots, T_n$ over $V_1, \ldots, V_n$, respectively, where $\{V_1, \ldots, V_n\}$ is a partition of V. We define $\mathsf{root}(F) = \bigcup_{i \in [n]} \{\mathsf{root}(T_i)\}$. 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 $v \neq u$. Let $U \subseteq V$ such that, for $v, u \in U$, $v \preceq_F u$ or $u \preceq_F v$. It can be verified that there exists a unique $v \in U$, denoted $\max_F(U)$, such that $u \prec_F v$, for each $u \in U$.

Witness Generator Scheme. As discussed above, a witness generator for a Boolean CQ q w.r.t. a database D and a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$, is a triple consisting of a function h that maps $\mathsf{var}(q)$ to constants and nulls, a function ν that assigns to each null $\bot \in \mathsf{null}(h(q))$ an atom that corresponds to the atom in which $\bot$ has been invented, and a rooted forest F over $\mathsf{null}(h(q))$. Such a triple, though, in order to be a valid candidate for a witness generator, must satisfy certain properties: (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 $\bot \in \mathsf{null}(h(q))$, $\bot$ is the most recently generated null among the nulls in $\nu(\bot)$. These are formalized via the notion of witness generator scheme. Recall that, for a set T of terms and a schema $\mathbf{S}$, $\mathsf{B}(T, \mathbf{S})$ is the set that collects all the atoms that can be formed using terms of T and predicates of $\mathbf{S}$.

Definition 4.1 *(Witness Generator Scheme).* Consider a Boolean CQ q over a schema $\mathbf{S}$. A *witness generator scheme* for q is a triple (h, ν, F), where $h : \mathsf{var}(q) \to \mathbf{C} \cup \mathbf{N}$, $\nu : \mathsf{null}(h(q)) \to \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$, and F is a rooted forest over $\mathsf{null}(h(q))$, such that

1. for each $\alpha \in (h(q) \cup \{\nu(\bot) \mid \bot \in \mathsf{null}(h(q))\})$, and for each pair of nulls $\bot_1, \bot_2 \in (\mathsf{null}(\alpha) \cap \mathsf{null}(h(q)))$, $\bot_1 \preceq_F \bot_2$ or $\bot_2 \preceq_F \bot_1$, and
2. for each $\bot \in \mathsf{null}(h(q))$, $\bot = \max_F(\mathsf{null}(\nu(\bot)) \cap \mathsf{null}(h(q)))$. $\square$

Witness Generator. We are now ready to define the key notion of the witness generator. A witness generator is essentially a witness generator scheme that indeed gives rise to a witness of the given query. Consider a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ over a schema $\mathbf{S}$. We first need to inductively define the binary relation $\vdash^k_\Sigma$ over $\mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$, for $k \geq 0$, as follows:

- $\alpha \vdash^0_\Sigma \beta$ if $\alpha = \beta$, or $\{\alpha\}\langle\sigma, h\rangle\{\alpha, \beta\}$ with (σ, h) being a trigger for Σ on $\{\alpha\}$;
- $\alpha \vdash^k_\Sigma \beta$ if there exists an atom $\gamma \in \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$ such that $\mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta) \subseteq \mathsf{dom}(\gamma)$, and $\alpha \vdash^{k-1}_\Sigma \gamma$ and $\gamma \vdash^{k-1}_\Sigma \beta$.

Intuitively, $\alpha \vdash^k_\Sigma \beta$ means that β is derivable from α in at most 2^k chase steps using TGDs from Σ. We further define the relation $\alpha \vdash^{k,\bot}_\Sigma \beta$, where $\bot$ is a null, as follows:

- $\alpha \vdash^{0,\bot}_\Sigma \beta$ if $\bot \in \mathsf{dom}(\beta) \setminus \mathsf{dom}(\alpha)$, and $\{\alpha\}\langle\sigma, h\rangle\{\alpha, \beta\}$ with (σ, h) being a trigger for Σ on $\{\alpha\}$;
- $\alpha \vdash^{k,\bot}_\Sigma \beta$ if there exists an atom $\gamma \in \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$ such that $\mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta) \subseteq \mathsf{dom}(\gamma)$, and $\alpha \vdash^{k-1}_\Sigma \gamma$ and $\gamma \vdash^{k-1,\bot}_\Sigma \beta$.

Roughly, $\alpha \vdash^{k,\bot}_\Sigma \beta$ means that β is derivable from α in at most 2^k chase steps using TGDs from Σ, and $\bot$ is invented in β. The notion of witness generator follows.

Definition 4.2 *(Witness Generator).* Consider a database D over a schema $\mathbf{S}$, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ of TGDs over $\mathbf{S}$, and a Boolean CQ q over $\mathbf{S}$. A *k-witness generator*, where $k \geq 0$, for q w.r.t. D and Σ is a witness generator scheme (h, ν, F) for q such that

1. For each $\bot \in \mathsf{root}(F)$, there exists $\alpha \in D$ such that $\alpha \vdash^{k,\bot}_\Sigma \nu(\bot)$.
2. For each edge $(\bot_1, \bot_2)$ of F, $\nu(\bot_1) \vdash^{k,\bot_2}_\Sigma \nu(\bot_2)$.
3. For each $\alpha \in h(q)$ with $\bot = \max_F(\mathsf{null}(\alpha))$, $\nu(\bot) \vdash^k_\Sigma \alpha$.
4. For each $\alpha \in h(q)$ with $\mathsf{null}(\alpha) = \varnothing$, there exists $\beta \in D$ such that $\beta \vdash^k_\Sigma \alpha$. $\square$

Note that for a witness generator (h, ν, F) for q w.r.t. D and Σ, it holds that the range of h is $\mathsf{dom}(D) \cup \mathbf{N}$ and the range of ν is $\mathsf{B}(\mathsf{dom}(D) \cup \mathbf{N}, \mathbf{S})$; otherwise, conditions (3) and (4) trivially fail. We proceed to characterize when a candidate answer is indeed a certain answer via the existence of an appropriate witness generator.

Lemma 4.1. *Consider a database D, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ of TGDs, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$. The following are equivalent:*

1. $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$.
2. *There exists a $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.*

Proof. (1) $\Rightarrow$ (2). By Theorem 3.2, $\mathbb{L}_{\mathsf{N}}$ enjoys the bounded witness property due to $f_{\mathbb{L}_{\mathsf{N}}}$. Thus, since $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$, there exists a prefix $(I_i)_{0 \leq i \leq n}$, with $n \leq \mathbb{L}_{\mathsf{N}}(\Sigma, q)$, of a chase derivation δ of D w.r.t. Σ such that $\bar{c} \in q(I_n)$. This in turn implies that there is a homomorphism h that maps $q(\bar{c})$ to I_n. For a null $\bot \in \mathsf{null}(h(q))$, there exists a *unique* integer $i \in [n]$ such that $\bot \in \mathsf{dom}(I_i) \setminus \mathsf{dom}(I_{i-1})$. We write $\alpha_\bot$ for the single atom in $I_i \setminus I_{i-1}$, that is, the atom where $\bot$ is invented according to δ, and we define the function $\nu : \mathsf{null}(h(q)) \to \mathsf{chase}_\delta(D, \Sigma)$ in such a way that $\nu(\bot) = \alpha_\bot$ for each $\bot \in \mathsf{null}(h(q))$. We finally define the rooted forest F over $\mathsf{null}(h(q))$ as follows:

- if $\mathsf{null}(h(q)) = \varnothing$, then F is the empty forest,
- if $\mathsf{null}(h(q)) = \{\bot\}$, then F consists of the single node $\bot$, and
- if $\mathsf{null}(h(q)) = \{\bot_1, \ldots, \bot_m\}$ for $m > 1$, then, for every $i, j \in \{1, \ldots, m\}$ with $i \neq j$, there exists an edge from $\bot_i$ to $\bot_j$ iff $\alpha_{\bot_i} \prec^{p,+}_\delta \alpha_{\bot_j}$, and there is no integer $k \in [m] \setminus \{i, j\}$ such that $\alpha_{\bot_i} \prec^{p,+}_\delta \alpha_{\bot_k}$ and $\alpha_{\bot_k} \prec^{p,+}_\delta \alpha_{\bot_j}$. Note that F is indeed a forest since $\prec^{p,+}_\delta$ forms a forest over $\mathsf{chase}_\delta(D, \Sigma)$ due to linearity.

We show that the triple (h, ν, F) is a $\lceil \log f_{\mathbb{L}_N}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ. The fact that (h, ν, F) is a witness generator scheme for q can be easily verified due to linearity. It remains to show that the four conditions in Definition 4.2 hold with $k = \lceil \log f_{\mathbb{L}_N}(\Sigma, q) \rceil$. To this end, it suffices to show that, for every two distinct atoms $\alpha, \beta \in \mathsf{chase}_\delta(D, \Sigma)$ such that $\alpha \prec_\delta^{p,+} \beta$, it holds that $\alpha \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_N}(\Sigma,q) \rceil} \beta$, and, if β is the atom where a null $\perp$ is invented, it holds that $\alpha \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_N}(\Sigma,q) \rceil, \perp} \beta$. Assume that

$$\alpha = \alpha_1 \prec_\delta^p \alpha_2 \prec_\delta^p \cdots \prec_\delta^p \alpha_\ell = \beta,$$

for $1 < \ell \leq f_{\mathbb{L}_N}(\Sigma, q)$, i.e., $\alpha_1, \ldots, \alpha_\ell$ is the path from α to β in the forest formed by $\prec_\delta^p$. It is easy to verify that, with $\ell' = 2^{\lceil \log f_{\mathbb{L}_N}(\Sigma,q) \rceil} - \ell + 1$, the sequence of atoms

$$\underbrace{\alpha_1, \ldots, \alpha_1}_{\ell'}, \alpha_1, \alpha_2, \ldots, \alpha_\ell$$

witnesses the fact that $\alpha \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_N}(\Sigma,q) \rceil} \beta$, and the fact that $\alpha \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_N}(\Sigma,q) \rceil, \perp} \beta$ in case β is the atom where $\perp$ is invented according to δ, as needed.

(2) $\Rightarrow$ (1). We first observe that for a quasi chase derivation $(I_i)_{0 \leq i \leq n}$, for $n \geq 0$, of D w.r.t. Σ, the following holds: for every chase derivation δ of D w.r.t. Σ, there exists a homomorphism h_δ that maps I_n to $\mathsf{chase}_\delta(D, \Sigma)$. This can be established via an easy inductive argument, which shows that, for every $i \in \{0, 1, \ldots, n\}$, there exists a homomorphism h_δ^i from I_i to $\mathsf{chase}_\delta(D, \Sigma)$, and, for $i > 0$, h_δ^i is compatible with h_δ^{i-1}, i.e., $h_\delta^{i-1} \cup h_\delta^i$ is well-defined. Therefore, h_δ can be defined as $\bigcup_{i=0}^n h_\delta^i$.

From the above observation, to show our claim, it suffices to show that the existence of a $\lceil \log f_{\mathbb{L}_N}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ implies the existence of a quasi chase derivation $(I_i)_{0 \leq i \leq n}$, for $n \geq 0$, of D w.r.t. Σ such that $\bar{c} \in q(I_n)$. Indeed, in such a case, since $q(\bar{c}) \to I_n$ and $I_n \to \mathsf{chase}_\delta(D, \Sigma)$, where δ is a chase derivation of D w.r.t. Σ, we get that $q(\bar{c}) \to \mathsf{chase}_\delta(D, \Sigma)$, and thus, $\bar{c} \in \mathsf{cert}(q, D, \Sigma)$, as needed. The rest of the proof is devoted to showing that (2) implies the existence of a quasi chase derivation $(I_i)_{0 \leq i \leq n}$, for $n \geq 0$, of D w.r.t. Σ such that $\bar{c} \in q(I_n)$.

Let $\Gamma = (h, \nu, F)$ be a $\lceil \log f_{\mathbb{L}_N}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ. The construction of the desired quasi chase derivation proceeds in three main steps:

- We first construct the forest F_Γ over atoms induced by Γ. As we shall see, the leaves of this forest are precisely the atoms of $h(q(\bar{c}))$.
- We then apply a renaming step over the atoms in F_Γ, which leads to the forest $F_\Gamma^\star$. This step ensures that different occurrences of the same null are semantically the same. Then, we show that the leaves of $F_\Gamma^\star$ are precisely the atoms of $h(q(\bar{c}))$, that is, the renaming step does not alter the atoms at the leaves of F_Γ.
- We finally convert $F_\Gamma^\star$ into a sequence of instances $(I_i)_{0 \leq i \leq n}$ that is a quasi chase derivation of D w.r.t. Σ (the latter is guaranteed by the renaming step), and $\bar{c} \in q(I_n)$ since the leaves of $F_\Gamma^\star$ are the atoms of $h(q(\bar{c}))$.

Step 1: Construction of F_Γ

We now explain how the rooted forest F_Γ is constructed. To this end, we first construct an auxiliary forest F'_Γ over the atoms $D \cup \{\nu(\perp) \mid \perp \in \mathsf{null}(h(q))\} \cup h(q)$ with the following edges (and only those edges). For brevity, let $k = \lceil \log f_{\mathbb{L}_N}(\Sigma, q) \rceil$.

- For each $\perp \in \mathsf{root}(F)$, assuming that $\alpha \in D$ is the atom provided by Γ with $\alpha \vdash_\Sigma^{k,\perp} \nu(\perp)$, there is an edge from α to $\nu(\perp)$.
- For each edge $(\perp_1, \perp_2)$ of F, there is an edge from $\nu(\perp_1)$ to $\nu(\perp_2)$.
- For each $\alpha \in h(q)$ with $\perp = \max_F(\mathsf{null}(\alpha))$, there is an edge from $\nu(\perp)$ to α.
- For each $\alpha \in h(q)$ with $\mathsf{null}(\alpha) = \varnothing$, assuming that $\beta \in D$ is the atom provided by Γ with $\beta \vdash_\Sigma^k \alpha$, there is an edge from β to α.

We now construct the forest F_Γ from F'_Γ as follows: replace each edge $e = (\alpha, \beta)$ of F'_Γ with the path of atoms $\alpha, \gamma_1^e, \ldots, \gamma_{2^k-1}^e, \beta$, where $\gamma_1^e, \ldots, \gamma_{2^k-1}^e$ is the sequence of atoms that witnesses $\alpha \vdash_\Sigma^k \beta$ (or $\alpha \vdash_\Sigma^{k,\perp} \beta$ in case $\beta = \nu(\perp)$). Let us clarify that, for distinct edges e, e' of F'_Γ, it might be the case that $\{\gamma_1^e, \ldots, \gamma_{2^k-1}^e\} \cap \{\gamma_1^{e'}, \ldots, \gamma_{2^k-1}^{e'}\}$ is non-empty. Nevertheless, the common atoms give rise to different nodes in F_Γ. This completes the construction of F_Γ. It is easy to see that the leaves of F_Γ are precisely the atoms of $h(q(\bar{c}))$ since the leaves of F'_Γ are precisely the atoms of $h(q(\bar{c}))$.

Step 2: Renaming of Nulls

The goal of this step is to ensure that different occurrences of a null in F_Γ are semantically the same. It might be the case that, for a certain null $\perp$, the subforest $F_\Gamma^\perp$ induced by $\perp$, that is, the subforest obtained by keeping only the nodes (i.e., atoms) containing $\perp$, is disconnected. In such a case, the occurrences of $\perp$ in different connected components of $F_\Gamma^\perp$ are semantically different, despite the fact that they have the same name. On the other hand, the occurrences of $\perp$ that

appear in the same connected component are semantically the same since they are invented and propagated via valid chase steps. We now explain how the renaming step works, which leads to the forest $F^\star_\Gamma$.

Let $\perp$ be a null in F_Γ, and let $T^\perp_1, \ldots, T^\perp_{m_\perp}$, for $m_\perp \geq 1$, be the connected components of the subforest $F^\perp_\Gamma$ induced by $\perp$. We consider the following two cases:

1. Assume that $\perp \notin \mathsf{null}(h(q(\bar{c})))$. We rename each occurrence of $\perp$ in $T^\perp_i$, for $i \in [m_\perp]$, into $\perp_i$.
2. Assume now that $\perp \in \mathsf{null}(h(q(\bar{c})))$. By construction, there exists $j_\perp \in [m_\perp]$ such that $T^\perp_{j_\perp}$ is rooted at $\nu(\perp)$. We rename each occurrence of $\perp$ in $T^\perp_i$, for $i \in [m_\perp] \setminus \{j_\perp\}$, into $\perp_i$. The only difference of this case is that the occurrences of $\perp$ in $T^\perp_{j_\perp}$ remain unchanged. This captures the intuition that these are the "real" occurrences of $\perp$ invented (in $\nu(\perp)$) and propagated via valid chase steps.

This completes the construction of $F^\star_\Gamma$. We proceed to establish the following claim:

Claim 4.1. *The following hold:*

1. *For a null $\perp$ in $F^\star_\Gamma$, the subforest $F^\star_\Gamma$ induced by $\perp$ is connected.*
2. *Let L be the set of atoms occurring as leaves in $F^\star_\Gamma$. Then $L = h(q(\bar{c}))$.*

Proof. It is easy to see that item (1) holds by construction. It remains to show item (2). Recall that the leaves of F_Γ are precisely the atoms of $h(q(\bar{c}))$. It is also clear that there is an 1-1 correspondence between the leaves of F_Γ and the leaves of $F^\star_\Gamma$ since the renaming step does not alter the structure of F_Γ. Thus, for a leaf node β in F_Γ, we can refer to its corresponding leaf node $\beta^\star$ in $F^\star_\Gamma$. Therefore, it suffices to show that, for each leaf node β in F_Γ, β and $\beta^\star$ are (syntactically) the same atom.

Consider an arbitrary leaf node β in F_Γ, and let $\alpha_1, \alpha_2, \ldots, \alpha_\ell, \beta$ be the unique path in F_Γ from a root node to β. We proceed to show that the renaming step does not alter β, which in turn implies that β and $\beta^\star$ are the same atom, as needed. The proof is by case analysis on whether β contains no null, one null, or more than one null:

1. Assume that $\mathsf{null}(\beta) = \varnothing$, which means that $\mathsf{dom}(\beta) \subseteq \mathsf{dom}(D)$. In this case, the claim holds trivially since during the renaming step constants from $\mathsf{dom}(D)$ are not renamed, and thus, $\beta = \beta^\star$.
2. Assume that $\mathsf{null}(\beta) = \{\perp\}$, i.e., β contains only one null. Since, by definition, Γ is a witness generator scheme for $q(\bar{c})$, we conclude that there exists an atom $\gamma \in \{\alpha_2, \ldots, \alpha_\ell\}$ such that $\gamma = \nu(\perp)$. Furthermore, since $\perp$ occurs in both β and γ, i.e., $\perp \in \mathsf{dom}(\beta) \cap \mathsf{dom}(\gamma)$, we get that $\perp$ occurs in every atom between γ and β. Thus, $\perp$ in β is not renamed during the renaming step, and thus, $\beta = \beta^\star$.
3. Assume that $\mathsf{null}(\beta) = \{\perp_1, \ldots, \perp_m\}$ for $m > 1$, i.e., β contains more than one null. Since Γ is a witness generator scheme for $q(\bar{c})$, the nulls $\perp_1, \ldots, \perp_m$ are comparable w.r.t. $\preceq_F$; let $\perp_1 \preceq_F \perp_2 \preceq_F \cdots \preceq \perp_m$. Furthermore, we conclude that, for each $i \in [m-1]$, $\nu(\perp_i)$ does not contain a null from $\{\perp_{i+1}, \ldots, \perp_m\}$. Therefore, there are atoms $\gamma_1, \ldots, \gamma_m \in \{\alpha_2, \ldots, \alpha_\ell\}$ such that $\gamma_i = \nu(\perp_i)$ for each $i \in [m]$, and the path from α_1 to β in F_Γ is of the form

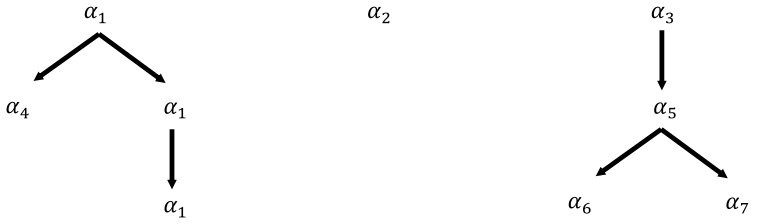
   $$\alpha_1, \ldots, \gamma_1, \ldots, \gamma_2, \ldots, \gamma_m, \ldots, \beta.$$

   Moreover, for each $i \in [m]$, the null $\perp_i \in \mathsf{dom}(\beta) \cap \mathsf{dom}(\gamma_i)$ occurs in every atom between γ_i and β. Therefore, none of the nulls $\perp_1, \ldots, \perp_m$ in β has been renamed during the renaming step, and thus, $\beta = \beta^\star$.

This completes the proof of Claim 4.1. □

Step 3: Construction of a quasi chase derivation

In this last step, we explain how $F^\star_\Gamma$ is converted into the desired quasi chase derivation of D w.r.t. Σ. Let s be the sequence of edges obtained by traversing $F^\star_\Gamma$ in a breadth-first fashion.[10] Let also s^- be the sequence of edges obtained from s by removing all the edges of the form (α, α), where $\alpha \in D$. For example, if $F^\star_\Gamma$ is of the form



[10] Here we simply need a mechanism to traverse $F^\star_\Gamma$. We could also traverse $F^\star_\Gamma$ in a depth-first fashion.

then

$$s = (\alpha_1, \alpha_4), (\alpha_1, \alpha_1), (\alpha_3, \alpha_5), (\alpha_1, \alpha_1), (\alpha_5, \alpha_6), (\alpha_5, \alpha_7)$$
$$s^- = (\alpha_1, \alpha_4), (\alpha_3, \alpha_5), (\alpha_5, \alpha_6), (\alpha_5, \alpha_7).$$

We inductively define a sequence of instances; let $s^- = e_1, \ldots, e_n$ for $n \geq 0$:

- $I_0 = D$, and
- for each $i \in [n]$, with $e_i = (\alpha, \beta)$, $I_i = I_{i-1} \cup \{\beta\}$.

It is now easy to see, due to Claim 4.1, that $(I_i)_{0 \leq i \leq n}$ is a quasi chase derivation of D w.r.t. Σ with $\bar{c} \in q(I_n)$. Let us conclude the proof by clarifying that $(I_i)_{0 \leq i \leq n}$ is not a prefix of a chase derivation of D w.r.t. Σ, as one might think, since triggers may repeat, i.e., the same trigger may be used in different applications. □

4.2. Binary witness generator

Lemma 4.1 essentially tells us that the desired query rewriters that are needed for establishing Proposition 4.1 should build first-order queries that check whether there is a $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ. To this end, we need to encode such a witness generator in a way that a first-order query can talk about it. The key idea underlying this encoding is to represent predicates and nulls via binary strings, i.e., strings consisting of 0 and 1; hence the name "binary witness generator". In the rest of the paper, we write o^k for the string $o \ldots o$ of length k, where o is some symbol and $k \geq 0$; whenever $k = 0$, o^k is the empty string.

Encoding/Decoding of Atoms. Consider a set $\Sigma \in \mathbb{L}_\mathsf{N}$ and a CQ $q(\bar{x})$ both over the schema $\mathbf{S} = \{P_1, \ldots, P_m\}$, where $m \geq 1$. We consider the integer

$$\tau \;=\; \left\lceil \log(|q| \cdot \mathsf{ar}(\mathbf{S}) \cdot f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) + 1) \right\rceil,$$

which is sufficiently large to ensure that all the nulls occurring in a witness induced by a $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator can be represented via binary strings of length τ. We define the set of binary strings

$$\mathbf{BS} \;=\; \left\{ 0^{i-1}10^{m-i} \mid i \in [m] \right\}$$

that collects the encodings of the predicates from $\mathbf{S}$. In particular, the predicate $P_i \in \mathbf{S}$ is encoded via the binary string $0^{i-1}10^{m-i}$. We further define the set of strings

$$\mathbf{BC} \;=\; \left\{ c^\tau \mid c \in \mathbf{C} \right\}$$

that collects the encodings of constants from $\mathbf{C}$. Note that a constant $c \in \mathbf{C}$ is represented via the string c^τ. The reason why c is represented via such a string of length τ, instead of c itself, is purely technical: we need all the terms, constants or nulls, in a witness induced by a $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator to be represented via strings of the same length. We finally define the set of binary strings

$$\mathbf{BN} \;=\; \{ b_1 \ldots b_\tau \mid b_i \in \{0, 1\} \text{ for each } i \in [\tau] \} \setminus \left\{ 0^\tau \right\}$$

that collects the encodings of null values that can appear in a witness induced by a $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator. Note that the string 0^τ is reserved for padding purposes and this is why is excluded from the set $\mathbf{BN}$; this will be made clear later.

Having the sets of strings $\mathbf{BS}$, $\mathbf{BC}$ and $\mathbf{BN}$ in place, we can now explain how an atom is encoded. Consider an atom $\alpha = P_i(t_1, \ldots, t_n) \in \mathsf{B}(\mathbf{C} \cup \mathbf{BC} \cup \mathbf{BN}, \mathbf{S})$, for some $i \in [m]$. The encoding of α, denoted $\mathsf{enc}(\alpha)$, is the string of length $(m + \mathsf{ar}(\mathbf{S}) \cdot \tau)$

$$0^{i-1}10^{m-i} \; \overbrace{0^\tau \ldots 0^\tau}^{\mathsf{ar}(\mathbf{S}) - \mathsf{ar}(P_i)} \; f(t_1) \ldots f(t_n),$$

where

$$f(t) \;=\; \begin{cases} t^\tau & \text{if } t \in \mathbf{C}, \\ t & \text{if } t \in (\mathbf{BC} \cup \mathbf{BN}). \end{cases}$$

Note that if $\mathsf{ar}(P_i) < \mathsf{ar}(\mathbf{S})$, then the first $\mathsf{ar}(\mathbf{S}) - \mathsf{ar}(P_i)$ positions of the encoded atom are padded with the special string 0^τ; this is the reason why $0^\tau \notin \mathbf{BN}$. Now, for a string t of length $(m + \mathsf{ar}(\mathbf{S}) \cdot \tau)$ of the form

$$0^{i-1}10^{m-i} \; \overbrace{0^\tau \ldots 0^\tau}^{\mathsf{ar}(\mathbf{S}) - \mathsf{ar}(P_i)} \; t_1 \ldots t_{\mathsf{ar}(P_i)},$$

where $t_j \in (\mathbf{BC} \cup \mathbf{BN})$ for each $j \in [\mathsf{ar}(P_i)]$, let $\mathsf{dec}(t)$ be the atom $P_i(t_1, \ldots, t_{\mathsf{ar}(P_i)})$, that is, the decoding of t. For an atom $\alpha \in \mathsf{B}(\mathbf{C} \cup \mathbf{BC} \cup \mathbf{BN}, \mathbf{S})$, let $\mathsf{bnull}(\alpha)$ be the set of strings $\mathsf{dom}(\alpha) \setminus (\mathbf{C} \cup \mathbf{BC})$, i.e., the binary strings in α that encode nulls; this notation extends to sets of atoms. The following example, which builds on Example 4.1, illustrates the encoding of atoms and how a binary witness generator looks like.

Example 4.2. Consider again the database

$$D = \{P(a, b, c), P(b, c, d)\}$$

the set Σ consisting of the linear TGDs

$$\beta \to \exists w\, P(x, y, w), \quad \beta \to \exists w\, P(z, x, w), \quad \beta \to P(z, y, x), \quad \beta \to \exists w\, P(y, z, w),$$

where $\beta = P(x, y, z)$, and the Boolean CQ

$$q = \exists x \exists y \exists z \exists w\, P(x, a, y) \wedge P(z, y, b) \wedge P(w, c, b)$$

introduced in Example 4.1. In the following, since Σ is already in normal form, we assume that Σ and $\mathsf{N}(\Sigma)$ coincide. Recall that the normal form for TGDs is discussed in Section 3. Observe that D, Σ and Q are over the schema $\mathbf{S} = \{P\}$. Clearly, $|\mathbf{S}| = 1$, $\mathsf{ar}(\mathbf{S}) = 3$, $|\Sigma| = 4$ and $|q| = 3$. Therefore, we have that

$$\begin{aligned} f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) &= |\Sigma| \cdot |\mathbf{S}| \cdot (2 \cdot |q| \cdot \mathsf{ar}(\mathbf{S}) + \mathsf{ar}(\mathbf{S}))^{\mathsf{ar}(\mathbf{S})} \\ &= 4 \cdot 1 \cdot (2 \cdot 3 \cdot 3 + 3)^3 \\ &= 37044, \end{aligned}$$

which in turn implies that

$$\begin{aligned} \tau &= \lceil \log(|q| \cdot \mathsf{ar}(\mathbf{S}) \cdot f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) + 1) \rceil \\ &= \lceil \log(3 \cdot 3 \cdot 37044 + 1) \rceil \\ &= 19. \end{aligned}$$

Therefore, we get that

$$\mathbf{BS} = \left\{0^{i-1}10^{m-i} \mid i \in [|\mathbf{S}|]\right\} = \{1\}$$

$$\mathbf{BC} = \left\{c^{19} \mid c \in \mathbf{C}\right\}$$

$$\mathbf{BN} = \{b_1, \ldots, b_{19} \mid b_i \in \{0, 1\} \text{ for } i \in [19]\} \setminus \{0^{19}\}.$$

Consider now the atom $\alpha = P(a, b, c)$. Since $|\mathbf{S}| = 1$ and $\mathsf{ar}(\mathbf{S}) - \mathsf{ar}(P) = 0$, its encoding $\mathsf{enc}(\alpha)$ is the string

$$1a^{19}b^{19}c^{19}.$$

Now, the atom $P(a, b, \bot_1)$ from Example 4.1, first has to be encoded as an atom of $\mathsf{B}(\mathbf{C} \cup \mathbf{BC} \cup \mathbf{BN}, \mathbf{S})$, namely $\alpha = P(a, b, t)$ with

$$t = 0^{18}1$$

being the first tuple of $\mathbf{BN}$ (according to the lexicographic order) that encodes $\bot_1$. Then, α can be encoded via the string

$$1a^{19}b^{19}0^{18}1.$$

The chase forest and rooted forest F of nulls from Example 4.1 remain structurally the same, with the key difference that atoms and nulls are encoded as described above. In particular, we have $h(q) = \{P(t_3, a, t_1), P(t_4, t_1, b), P(t_5, c, b)\}$, where

$$t_1 = 0^{18}1 \qquad t_3 = 0^{17}11 \qquad t_4 = 0^{16}100 \qquad t_5 = 0^{16}101.$$

This completes our example. □

Binary Witness Generator Scheme. The notion of binary witness generator scheme is defined in the same way as the notion of witness generator scheme (see Definition 4.1), with the difference that we use the binary encoding of terms and atoms discussed above.

Definition 4.3 *(Binary Witness Generator Scheme).* Consider a Boolean CQ q over a schema $\mathbf{S}$. A *binary witness generator scheme* for q is a triple (h, ν, F), where $h : \mathsf{var}(q) \to \mathbf{BC} \cup \mathbf{BN}$, $\nu : \mathsf{bnull}(h(q)) \to \{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\mathbf{BC} \cup \mathbf{BN}, \mathbf{S})\}$, and F is a rooted forest over $\mathsf{bnull}(h(q))$, such that:

1. For each $\alpha \in (h(q) \cup \{\mathsf{dec}(t) \mid t \in \mathsf{bnull}(h(q))\})$, and for each pair of distinct strings $t_1, t_2 \in (\mathsf{bnull}(\alpha) \cap \mathsf{bnull}(h(q)))$, $t_1 \preceq_F t_2$ or $t_2 \preceq_F t_1$.
2. For each $t \in \mathsf{bnull}(h(q))$, $t = \max_F(\mathsf{bnull}(\mathsf{dec}(\nu(t))) \cap \mathsf{bnull}(h(q)))$. □

Binary Witness Generator. We proceed to define the notion of binary witness generator. Consider a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ over a schema $\mathbf{S}$. We first need to define the binary relation $\vdash^k_\Sigma$, for $k \geq 0$, over encodings of atoms, i.e., over $\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\mathbf{BC} \cup \mathbf{BN}, \mathbf{S})\}$:

- $t \vdash^0_\Sigma u$ if $t = u$, or $\{\mathsf{dec}(t)\}\langle\sigma, h\rangle\{\mathsf{dec}(t), \mathsf{dec}(u)\}$ with (σ, h) being a trigger for Σ on $\{\mathsf{dec}(t)\}$;
- $t \vdash^k_\Sigma u$ if there is $v \in \{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\mathbf{BC} \cup \mathbf{BN}, \mathbf{S})\}$ such that $\mathsf{dom}(\mathsf{dec}(t)) \cap \mathsf{dom}(\mathsf{dec}(u)) \subseteq \mathsf{dom}(\mathsf{dec}(v))$, and $t \vdash^{k-1}_\Sigma v$ and $v \vdash^{k-1}_\Sigma u$.

Intuitively, $t \vdash^k_\Sigma u$ means that $\mathsf{dec}(u)$ is derivable from $\mathsf{dec}(t)$ in at most 2^k chase steps using TGDs from Σ. We further define the relation $\vdash^{k,\nu}_\Sigma$, where $\nu \in \mathbf{BN}$, as follows:

- $t \vdash^{0,\nu}_\Sigma u$ if $\nu \in \mathsf{dom}(\mathsf{dec}(u)) \setminus \mathsf{dom}(\mathsf{dec}(t))$, and $\{\mathsf{dec}(t)\}\langle\sigma, h\rangle\{\mathsf{dec}(t), \mathsf{dec}(u)\}$ with (σ, h) being a trigger for Σ on $\{\mathsf{dec}(t)\}$;
- $t \vdash^{k,\nu}_\Sigma u$ if there is $w \in \{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\mathbf{BC} \cup \mathbf{BN}, \mathbf{S})\}$ such that $\mathsf{dom}(\mathsf{dec}(t)) \cap \mathsf{dom}(\mathsf{dec}(u)) \subseteq \mathsf{dom}(\mathsf{dec}(w))$, and $t \vdash^{k-1}_\Sigma w$ and $w \vdash^{k-1,\nu}_\Sigma u$.

Roughly, $t \vdash^{k,\nu}_\Sigma u$ means that $\mathsf{dec}(u)$ is derivable from $\mathsf{dec}(t)$ in at most 2^k chase steps using TGDs from Σ, and ν is invented in $\mathsf{dec}(u)$. Let us clarify that during a chase step $\{\mathsf{dec}(t)\}\langle\sigma, h\rangle\{\mathsf{dec}(t), \mathsf{dec}(u)\}$ the invented nulls are strings from $\mathbf{BN}$.

Definition 4.4 *(Binary Witness Generator).* Consider a database D over $\mathbf{S}$, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ over $\mathbf{S}$, and a Boolean CQ q over $\mathbf{S}$. A *binary k-witness generator*, where $k \geq 0$, for q w.r.t. D and Σ is a binary witness generator scheme (h, ν, F) for q such that:

1. For each $t \in \mathsf{root}(F)$, there exists $\alpha \in D$ such that $\mathsf{enc}(\alpha) \vdash^{k,t}_\Sigma \nu(t)$.
2. For each edge (t_1, t_2) of F, $\nu(t_1) \vdash^{k,t_2}_\Sigma \nu(t_2)$.
3. For each $\alpha \in h(q)$ with $t = \max_F(\mathsf{bnull}(\alpha))$, $\nu(t) \vdash^k_\Sigma \mathsf{enc}(\alpha)$.
4. For each $\alpha \in h(q)$ s.t. $\mathsf{bnull}(\alpha) = \varnothing$, there is $\beta \in D$ with $\mathsf{enc}(\beta) \vdash^k_\Sigma \mathsf{enc}(\alpha)$. □

Note that for a binary witness generator (h, ν, F) for q w.r.t. D and Σ, the range of h is $\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}$, and the range of ν is $\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}, \mathbf{S})\}$; otherwise, conditions (3) and (4) trivially fail. The next easy lemma, which follows by definition, shows the correspondence between $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generators and binary $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generators.

Lemma 4.2. *Consider a database D, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ of TGDs, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$. The following are equivalent:*

1. *There exists a $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.*
2. *There exists a binary $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.*

4.3. Item (1) of Proposition 4.1

We now have all the ingredients for showing Proposition 4.1. We start with the first item, which we recall again here:

there is a polynomial-time computable query rewriter $f_{\mathsf{Q}} : \mathbb{CQ} \times \mathbb{L}_{\mathsf{N}} \to \mathbb{FO}$ such that, for a database D, $\Sigma \in \mathbb{L}_{\mathsf{N}}$, and CQ q, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ *with* $q_\Sigma = f_{\mathsf{Q}}(q, \Sigma)$.

By Lemmas 4.1 and 4.2, it suffices to define f_{Q} in such a way that, given a database D, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$, a CQ $q(\bar{x})$, and a tuple $\bar{c}$, the following are equivalent:

1. $\bar{c} \in q_\Sigma(D_{01})$ with $q_\Sigma = f_{\mathsf{Q}}(q, \Sigma)$.
2. There exists a binary $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.

Consider a set $\Sigma \in \mathbb{L}_{\mathsf{N}}$ and a CQ $q(\bar{x})$ both over the schema $\mathbf{S} = \{P_1, \ldots, P_m\}$, where $m \geq 1$. We proceed to define the first-order query $f_{\mathsf{Q}}(q, \Sigma)$. Henceforth, for brevity, we write ω for $\mathsf{ar}(\mathbf{S})$. We also assume that the CQ $q(\bar{x})$ is of the form

Table 1
U, **W** (possibly with subscripts) and **V** are τ-tuples of variables that represent terms, while **R** is an $|\mathbf{S}|$-tuple of variables that represents a predicate. $\mathbf{Y}_i$, $\mathbf{A}_i$ and $\mathbf{B}_i$ are variables of **Q**.

Shortcut	Definition
$\mathbf{U} = \mathbf{W}$	$\bigwedge_{i\in[\lvert\mathbf{U}\rvert]} (\mathbf{U}[i] = \mathbf{W}[i])$
$\mathbf{U} \prec \mathbf{W}$	$\bigvee_{i\in\{0,\ldots,\lvert\mathbf{U}\rvert-1\}} \big(\bigwedge_{j\in\{1,\ldots,i\}} \mathbf{U}[j] = \mathbf{W}[j] \wedge$ $\mathsf{Zero}(\mathbf{U}[i+1]) \wedge \mathsf{One}(\mathbf{W}[i+1])\big)$
$\mathsf{Dom}(\mathbf{U})$	$\bigwedge_{u\in\mathbf{U}} \neg(\mathsf{Zero}(u) \vee \mathsf{One}(u))$
$\mathsf{Null}(\mathbf{U})$	$\bigwedge_{u\in\mathbf{U}} (\mathsf{Zero}(u) \vee \mathsf{One}(u)) \ \wedge \ \bigvee_{u\in\mathbf{U}} \mathsf{One}(u)$
$\mathsf{QNull}(\mathbf{U})$	$\mathsf{Null}(\mathbf{U}) \wedge \big(\bigvee_{i\in[\ell]} \mathbf{U} = \mathbf{Y}_i\big)$
$\mathsf{Atom}(\mathbf{R}, \mathbf{U}_1, \ldots, \mathbf{U}_\omega)$	$\bigvee_{i\in[m]} \Big(\mathsf{One}(\mathbf{R}[i]) \ \wedge \ \bigwedge_{j\in[m]\setminus\{i\}} \mathsf{Zero}(\mathbf{R}[j]) \wedge$ $\bigwedge_{j\in[\omega-\mathsf{ar}(P_i)]} \mathbf{U}_j = (0)^\tau \ \wedge$ $\bigwedge_{j\in\{\omega-\mathsf{ar}(P_i)+1,\ldots,\omega\}} \big(\mathsf{Dom}(\mathbf{U}_j) \ \vee \ \mathsf{Null}(\mathbf{U}_j)\big)\Big)$
$\mathbf{U} \prec_1 \mathbf{W}$	$\neg(\mathbf{U} = \mathbf{W}) \wedge \big(\bigvee_{i\in[\ell-1]} \mathbf{U} = \mathbf{A}_i \wedge \mathbf{W} = \mathbf{B}_i\big)$
$\mathbf{U} \prec_{i+1} \mathbf{W}$	$\exists \mathbf{V} \, (\mathbf{U} \prec_i \mathbf{V} \wedge \mathbf{V} \prec_1 \mathbf{W})$
$\mathbf{U} \prec \mathbf{W}$	$\bigvee_{i\in[\ell-1]} \mathbf{U} \prec_i \mathbf{W}$
$\mathsf{Min}(\mathbf{U})$	$\big(\bigvee_{\mathbf{V}\in\mathbb{A}} \mathbf{U} = \mathbf{V}\big) \wedge \neg\big(\bigvee_{\mathbf{V}\in\mathbb{B}} \mathbf{U} = \mathbf{V}\big)$
$\mathsf{Max}(\mathbf{U}, \mathbf{W}_1, \ldots, \mathbf{W}_\omega)$	$\big(\bigvee_{i\in[\omega]} \mathbf{U} = \mathbf{W}_i\big) \wedge$ $\bigwedge_{i\in[\omega]} (\mathsf{QNull}(\mathbf{W}_i) \rightarrow \mathbf{U} = \mathbf{W}_i \vee \mathbf{W}_i \prec \mathbf{U})$

$$q(x_1, \ldots, x_k) \ = \ \exists y_1 \cdots \exists y_\ell \, (P_{a_1}(\bar{t}_1) \wedge \cdots \wedge P_{a_n}(\bar{t}_n)).$$

The first-order query $q_\Sigma = f_{\mathsf{Q}}(q, \Sigma)$ is the conjunction of the following three components; let D be the input database on which q_Σ will be evaluated:

Component 1. Guess a candidate binary witness generator (h, ν, F) such that, for each $i \in [k]$, $h(x_i) \in \{c^\tau \mid c \in \mathsf{dom}(D)\}$.
Component 2. Assuming that $h(x_i) = c_i^\tau$, for each $i \in [k]$, verify that (h, ν, F) is a binary witness generator scheme for $q(\bar{c})$, where $\bar{c} = (c_1, \ldots, c_k)$.
Component 3. Finally, verify that (h, ν, F) is a binary $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.

Before giving the syntactic shape of the query $f_{\mathsf{Q}}(q, \Sigma)$, let us introduce the variables, and their underlying meaning, that we will use:

- For each $i \in [k]$, $\mathbf{X}_i = x_i^1, \ldots, x_i^\tau$ represents the string from $\{c^\tau \mid c \in \mathsf{dom}(D)\}$ to which the free variable x_i of q is mapped to via h.
- For each $i \in [\ell]$, $\mathbf{Y}_i = y_i^1, \ldots, y_i^\tau$ represents the string from $\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}$ to which the variable y_i of q is mapped to via h.
- For each $i \in [\ell]$, $\mathcal{A}_i = \mathbf{P}_i, \mathbf{T}_i^1, \ldots, \mathbf{T}_i^\omega$, where $\mathbf{P}_i = p_i^1, \ldots, p_i^{|\mathbf{S}|}$, and $\mathbf{T}_i^j$ a τ-tuple of distinct variables, for each $j \in [\omega]$, represents the string from

  $$\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}, \mathbf{S})\}$$

  to which $h(y_i)$ is mapped to via ν whenever $h(y_i) \in \mathbf{BN}$.
- For each $i \in [\ell - 1]$, the pair $(\mathbf{A}_i, \mathbf{B}_i)$, where $\mathbf{A}_i, \mathbf{B}_i$ are τ-tuples of distinct variables, represents an edge in the rooted forest F over $\mathsf{bnull}(h(q))$. For brevity, we write $\mathbb{A}$ and $\mathbb{B}$ for the sets $\{\mathbf{A}_1, \ldots, \mathbf{A}_{\ell-1}\}$ and $\{\mathbf{B}_1, \ldots, \mathbf{B}_{\ell-1}\}$, respectively.

We define **Q** as the set that collects all the variables introduced above apart from the variables $x_1^1, \ldots, x_k^1$. The general syntactic shape of q_Σ is

$$\exists \mathbf{Q} \ (\mathsf{Triple}(\mathbf{Q}) \ \wedge \ \mathsf{WitnessGeneratorScheme}(\mathbf{Q}) \ \wedge \ \mathsf{WitnessGenerator}(\mathbf{Q})) \, ,$$

where the subquery Triple corresponds to Component 1 discussed above, the subquery WitnessGeneratorScheme to Component 2, and the subquery WitnessGenerator to Component 3. In what follows, whenever we write Name(**Q**) we mean that the free variables of the subquery Name are among **Q**. With the aim of simplifying the definition of q_Σ, we use some useful shortcuts given in Table 1; as usual, by $\mathbf{X}[i]$ we refer to the i-th element of the tuple **X**. The intuitive meaning of those shortcuts follows:

- $\mathbf{U} = \mathbf{W}$ checks whether the τ-tuples **U** and **W** are equal.

- $\mathbf{U} < \mathbf{W}$ checks whether the binary string of length τ that corresponds to $\mathbf{U}$, which can be seen as a binary number, is strictly less than the binary string (or binary number) that corresponds to $\mathbf{W}$.
- Dom($\mathbf{U}$) checks whether $\mathbf{U}$ corresponds to an element of **BC**. This is done by checking that $\mathbf{U}$ does not contain 0 or 1.
- Null($\mathbf{U}$) checks whether $\mathbf{U}$ corresponds to an element of **BN**.
- QNull($\mathbf{U}$) checks whether $\mathbf{U}$ corresponds to an element of **BN** that occurs in the image (via h) of the query, i.e., corresponds to a null value and is the image of an existentially quantified variable y_i of q.
- Atom($\mathbf{R}, \mathbf{U}_1, \ldots, \mathbf{U}_\omega$) checks whether $\mathbf{R}, \mathbf{U}_1, \ldots, \mathbf{U}_\omega$ corresponds to the encoding of some atom, or, in other words, whether it corresponds to an element of $\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\mathbf{BC} \cup \mathbf{BN}, \mathbf{S})\}$.
- $\mathbf{U} \prec_1 \mathbf{W}$ checks whether $(\mathbf{U}, \mathbf{W})$ corresponds to an edge in the forest F.
- $\mathbf{U} \prec_{i+1} \mathbf{W}$ checks if w is reachable in F from u via a path of length $i+1$, assuming that $\mathbf{U}$ and $\mathbf{W}$ correspond to the strings u and w, respectively.
- $\mathbf{U} \prec \mathbf{W}$ checks if w is reachable in F from u via some path, assuming that $\mathbf{U}$ and $\mathbf{W}$ correspond to the strings u and w, respectively. Note that such a path is of length at most $\ell - 1$ since ℓ is the maximum number of strings from **BN** that can appear in the image of the query q (there are ℓ existentially quantified variables in the query), and thus, the maximum number of nodes in F.
- Min($\mathbf{U}$) checks whether $\mathbf{U}$ corresponds to a root node u in F, which boils down to checking that u does not have any incoming edge in F.
- Max($\mathbf{U}, \mathbf{W}_1, \ldots, \mathbf{W}_\omega$) checks whether u corresponds to the greatest element (w.r.t. $\preceq_F$) among $w_1, \ldots, w_\omega$, assuming that $\mathbf{U}, \mathbf{W}_1, \ldots, \mathbf{W}_\omega$ correspond to the strings $u, w_1, \ldots, w_\omega$, respectively.

We are now ready to define the queries Triple($\mathbf{Q}$), WitnessGeneratorScheme($\mathbf{Q}$) and WitnessGenerator($\mathbf{Q}$), which will give rise to the desired FO query q_Σ.

The Subquery Triple($\mathbf{Q}$)**.** The goal of this subquery is to perform some consistency checks to ensure that indeed the guessed triple is of the right syntactic form.

We first need to ensure that the free variables of q are mapped to strings of $\{c^\tau \mid c \in \mathsf{dom}(D)\}$, while the existentially quantified variables are mapped to strings of $\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}$. This is achieved via the query VariableConsistency($\mathbf{Q}$):

$$\bigwedge_{i \in [k]} \mathrm{Dom}(\mathbf{X}_i) \wedge \bigwedge_{j \in \{2,\ldots,\tau\}} x_i^1 = x_i^j \wedge \bigwedge_{i \in [\ell]} \left(\left(\mathrm{Dom}(\mathbf{Y}_i) \wedge \bigwedge_{j \in \{2,\ldots,\tau\}} y_i^1 = y_i^j \right) \vee \mathrm{Null}(\mathbf{Y}_i) \right).$$

We also need check that the tuples $\mathcal{A}_i$ indeed represent strings from $\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}, \mathbf{S})\}$, done via the query AtomConsistency($\mathbf{Q}$):

$$\bigwedge_{i \in [\ell]} \mathrm{Atom}(\mathcal{A}_i).$$

We then check that ν is a (total) function via the query TotalFunction($\mathbf{Q}$):

$$\bigwedge_{i,j \in [\ell]} \left(\mathrm{Null}(\mathbf{Y}_i) \wedge \mathrm{Null}(\mathbf{Y}_j) \wedge \mathbf{Y}_i = \mathbf{Y}_j \rightarrow \mathcal{A}_i = \mathcal{A}_j \right) \wedge \bigwedge_{i \in [\ell]} \left(\mathrm{Null}(\mathbf{Y}_i) \rightarrow \mathbf{Y}_i = \mathbf{T}_i^\omega \right).$$

The above query essentially checks the following: (i) for every two existentially quantified variables y_i, y_j in q that are mapped via h to the same string from **BN**, it holds that $\nu(h(y_i)) = \nu(h(y_j))$ (i.e., ν is a function), and (ii) for each existentially quantified variable y_i in q, if $h(y_i) \in \mathbf{BN}$, then $h(y_i)$ is indeed invented in the atom represented by $\mathcal{A}_i$ (i.e., ν is total); recall that $\mathbf{T}_i^\omega$ is the last term of $\mathcal{A}_i$ that represents $h(y_i)$.

We check that F is a rooted forest over $\mathsf{bnull}(h(q))$ via the query RootedForest($\mathbf{Q}$):

$$\bigwedge_{i \in [\ell-1]} (\mathrm{QNull}(\mathbf{A}_i) \wedge \mathrm{QNull}(\mathbf{B}_i)) \wedge \bigwedge_{i \in [\ell]} \left(\mathrm{Null}(\mathbf{Y}_i) \rightarrow \bigvee_{\mathbf{U} \in \mathbb{A} \cup \mathbb{B}} \mathbf{Y}_i = \mathbf{U} \right) \wedge \bigwedge_{i \in [\ell-1]} \mathbf{A}_i < \mathbf{B}_i \wedge \bigwedge_{i,j \in [\ell-1]} \left(\mathbf{B}_i = \mathbf{B}_j \rightarrow \mathbf{A}_i = \mathbf{A}_j \right).$$

Recall that the pair $(\mathbf{A}_i, \mathbf{B}_i)$ represents an edge in F. Hence, the above query checks that (i) the nodes of F are tuples from $\mathsf{bnull}(h(q))$, (ii) each tuple of $\mathsf{bnull}(h(q))$ occurs in F, and (iii) the edges $(\mathbf{A}_1, \mathbf{B}_1), \ldots, (\mathbf{A}_{\ell-1}, \mathbf{B}_{\ell-1})$ indeed form a rooted forest.

Consequently, Triple($\mathbf{Q}$) is defined as the first-order query:

$$\mathrm{VariableConsistency}(\mathbf{Q}) \wedge \mathrm{AtomConsistency}(\mathbf{Q}) \wedge \mathrm{TotalFunction}(\mathbf{Q}) \wedge \mathrm{RootedForest}(\mathbf{Q}).$$

The Subquery WitnessGeneratorScheme($\mathbf{Q}$)**.** We now check that the guessed triple represents a binary witness generator scheme for $q(\bar{c})$, where $\bar{c} = (h(x_1^1), \ldots, h(x_k^1))$. But let us first introduce some auxiliary notation.

For each $i \in [n]$ (recall that $P_{a_i}(\bar{t}_i)$ is an atom of q), $\mathbf{TT}_i$ is defined as the tuple

$$\overbrace{\underbrace{0,\ldots,0}_{\tau},\ldots,\underbrace{0,\ldots,0}_{\tau}}^{\omega-\mathsf{ar}(P_{a_i})}, f(\bar{t}_i[1]),\ldots,f(\bar{t}_i[\mathsf{ar}(P_{a_i})]),$$

where

$$f(t) = \begin{cases} \underbrace{t,\ldots,t}_{\tau} & \text{if } t \in \mathbf{C}, \\ t^1,\ldots,t^\tau & \text{if } t \in \bar{x} \cup \bar{y}. \end{cases}$$

Moreover, we define

$$\mathbf{TT}_i^j = \begin{cases} \underbrace{0,\ldots,0}_{\tau} & \text{if } j \in \{1,\ldots,\omega-\mathsf{ar}(P_{a_i})\}, \\ f(\bar{t}_i[j]) & \text{if } j \in \{\omega-\mathsf{ar}(P_{a_i})+1,\ldots,\omega\}. \end{cases}$$

For example, if $\omega = 4$, $\bar{t}_i = c, y_3, y_1$, and $\tau = 2$, then $\mathbf{TT}_i = 0,0,c,c,y_3^1,y_3^2,y_1^1,y_1^2$, while $\mathbf{TT}_i^1 = 0,0$, $\mathbf{TT}_i^2 = c,c$, $\mathbf{TT}_i^3 = y_3^1,y_3^2$ and $\mathbf{TT}_i^4 = y_1^1,y_1^2$. We are now ready to proceed with the queries that check for the two conditions of Definition 4.3.

The first condition *"for each $\alpha \in (h(q) \cup \{\mathsf{dec}(\nu(t)) \mid t \in \mathsf{bnull}(h(q))\})$, and for each pair of distinct strings $t_1, t_2 \in (\mathsf{bnull}(\alpha) \cap \mathsf{bnull}(h(q)))$, $t_1 \preceq_F t_2$ or $t_2 \preceq_F t_1$"* is checked via the query $\mathsf{WGS}_1(\mathbf{Q})$:

$$\bigwedge_{i\in[n]}\bigwedge_{j,r\in[\omega]}\Big(\mathrm{Null}(\mathbf{TT}_i^j) \wedge \mathrm{Null}(\mathbf{TT}_i^r) \rightarrow (\mathbf{TT}_i^j = \mathbf{TT}_i^r) \vee (\mathbf{TT}_i^j \prec \mathbf{TT}_i^r) \vee (\mathbf{TT}_i^r \prec \mathbf{TT}_i^j)\Big) \wedge$$

$$\bigwedge_{i\in[\ell]}\bigwedge_{j,r\in[\omega]}\Big(\mathrm{QNull}(\mathbf{T}_i^j) \wedge \mathrm{QNull}(\mathbf{T}_i^r) \rightarrow (\mathbf{T}_i^j = \mathbf{T}_i^r) \vee (\mathbf{T}_i^j \prec \mathbf{T}_i^r) \vee (\mathbf{T}_i^r \prec \mathbf{T}_i^j)\Big).$$

The first conjunction of implications checks that, for each $\alpha \in h(q)$, every two distinct strings of $\mathsf{bnull}(\alpha)$ are comparable w.r.t. $\preceq_F$. The second conjunction of implications takes care of the atoms $\{\nu(t) \mid t \in \mathsf{bnull}(h(q))\}$. In particular, for each such atom α, it checks whether every two distinct strings of $\mathsf{bnull}(\alpha)$ are comparable w.r.t. $\preceq_F$.

The second condition of Definition 4.3, namely *"for each $t \in \mathsf{bnull}(h(q))$, $t = \max_F(\mathsf{bnull}(\mathsf{dec}(\nu(t))) \cap \mathsf{bnull}(h(q)))$"*, is checked via the query $\mathsf{WGS}_2(\mathbf{Q})$:

$$\bigwedge_{i\in[\ell]}\Big(\mathrm{QNull}(\mathbf{T}_i^\omega) \rightarrow \mathrm{Max}(\mathbf{T}_i^\omega,\mathbf{T}_i^1,\ldots,\mathbf{T}_i^\omega)\Big).$$

Note that the query $\mathsf{WGS}_2(\mathbf{Q})$ assumes the following: for each existentially quantified variable y_i in q, if $h(y_i) \in \mathbf{BN}$, then $h(y_i)$ is invented in the atom $\mathsf{dec}(\nu(h(y_i)))$, i.e., $\mathbf{Y}_i = \mathbf{T}_i^\omega$. This is guaranteed by the query TotalFunction($\mathbf{Q}$) defined above.

Hence, WitnessGeneratorScheme($\mathbf{Q}$) is defined as the first-order query

$$\mathsf{WGS}_1(\mathbf{Q}) \wedge \mathsf{WGS}_2(\mathbf{Q}).$$

The Subquery WitnessGenerator($\mathbf{Q}$)**.** We finally check whether the guessed triple (h,ν,F) is a binary $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma,q)\rceil$-witness generator for $q(\bar{c})$ w.r.t. the input database D and Σ, where $\bar{c} = (h(x_1^1),\ldots,h(x_k^1))$. Note that the check whether it is a witness generator scheme is taken care by the previous query. Therefore, the remaining task is to check the four conditions given in Definition 4.4.

We assume, for the moment, that we have available the subqueries π and π_G. Intuitively speaking, given $\mathcal{A},\mathcal{B}$ that represent the strings $t_\mathcal{A}$ and $t_\mathcal{B}$ from $\{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN},\mathbf{S})\}$, respectively,

$$\pi(\mathcal{A},\mathcal{B}) \equiv t_\mathcal{A} \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma,q)\rceil} t_\mathcal{B} \qquad \text{and} \qquad \pi_G(\mathcal{A},\mathcal{B}) \equiv t_\mathcal{A} \vdash_\Sigma^{\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma,q)\rceil,t} t_\mathcal{B},$$

where $t \in \mathbf{BN}$ is the last string of length τ in $t_\mathcal{B}$; π and π_G are defined below.

The first condition of Definition 4.4, namely *"for each $t \in \mathsf{root}(F)$, there exists $\alpha \in D$ such that $\mathsf{enc}(\alpha) \vdash_\Sigma^{k,t} \nu(t)$"*, is checked via the query $\mathsf{WG}_1(\mathbf{Q})$:

$$\bigwedge_{i\in[\ell]}\Big(\mathrm{QNull}(\mathbf{T}_i^\omega)\ \wedge\ \mathrm{Min}(\mathbf{T}_i^\omega)\ \rightarrow\ \exists p_1\ldots\exists p_m\exists s_1\cdots\exists s_\omega$$
$$\mathrm{Atom}(p_1,\ldots,p_m,(s_1)^\tau,\ldots,(s_\omega)^\tau)\ \wedge$$
$$\bigvee_{j\in[m]}\Big(\mathrm{One}(p_j)\ \rightarrow\ P_j(s_{\omega-\mathsf{ar}(P_j)+1},\ldots,s_\omega)\ \wedge$$
$$\pi_G(p_1,\ldots,p_m,(s_1)^\tau,\ldots,(s_\omega)^\tau,\mathcal{A}_i)\Big)\Big).$$

The second condition, namely *"for each edge (t_1,t_2) of F, $\nu(t_1)\vdash^{k,t_2}_\Sigma \nu(t_2)$"*, is checked via the query $\mathrm{WG}_2(\mathbf{Q})$:

$$\bigwedge_{i,j\in[\ell]}\Big(\mathrm{QNull}(\mathbf{T}_i^\omega)\ \wedge\ \mathrm{QNull}(\mathbf{T}_j^\omega)\ \wedge\ (\mathbf{T}_i^\omega\prec_1\mathbf{T}_j^\omega)\ \rightarrow\ \pi_G(\mathcal{A}_i,\mathcal{A}_j)\Big).$$

The third condition, i.e., *"for each $\alpha\in h(q)$ with $t=\max_F(\mathsf{bnull}(\alpha))$, $\nu(t)\vdash^k_\Sigma \mathsf{enc}(\alpha)$"*, is checked via the query $\mathrm{WG}_3(\mathbf{Q})$:

$$\bigwedge_{i\in[n]}\bigwedge_{j\in[\ell]}\Big(\mathrm{QNull}(\mathbf{T}_j^\omega)\ \wedge\ \mathrm{Max}(\mathbf{T}_j^\omega,\mathbf{TT}_i)\ \rightarrow\ \pi(\mathcal{A}_j,\underbrace{0,\ldots,0}_{a_i-1},1,\underbrace{0\ldots,0}_{m-a_i},\mathbf{TT}_i)\Big).$$

Finally, the fourth condition, namely *"for each $\alpha\in h(q)$ s.t. $\mathsf{null}(\alpha)=\varnothing$, there is $\beta\in D$ with $\mathsf{enc}(\beta)\vdash^k_\Sigma\mathsf{enc}(\alpha)$"*, is checked via the query $\mathrm{WG}_4(\mathbf{Q})$:

$$\bigwedge_{i\in[n]}\Big(\bigwedge_{j\in[\omega]}\mathrm{Dom}(\mathbf{TT}_i^j)\ \rightarrow\ \exists p_1\ldots\exists p_m\exists s_1\cdots\exists s_\omega$$
$$\mathrm{Atom}(p_1,\ldots,p_m,(s_1)^\tau,\ldots,(s_\omega)^\tau)\ \wedge$$
$$\bigvee_{j\in[m]}\Big(\mathrm{One}(p_j)\ \rightarrow\ P_j(s_{\omega-\mathsf{ar}(P_j)+1},\ldots,s_\omega)\ \wedge$$
$$\pi(p_1,\ldots,p_m,(s_1)^\tau,\ldots,(s_\omega)^\tau,\underbrace{0,\ldots,0}_{a_i-1},1,\underbrace{0\ldots,0}_{m-a_i},\mathbf{TT}_i)\Big)\Big).$$

Consequently, WitnessGenerator($\mathbf{Q}$) is defined as the first-order query

$$\mathrm{WG}_1(\mathbf{Q})\ \wedge\ \mathrm{WG}_2(\mathbf{Q})\ \wedge\ \mathrm{WG}_3(\mathbf{Q})\ \wedge\ \mathrm{WG}_4(\mathbf{Q}).$$

It remains to define the crucial subqueries π and π_G.

The Subqueries π and π_G. Assume, for the moment, that we have the query $\pi_i(\mathcal{A},\mathcal{B},s)$, where $\mathcal{A},\mathcal{B}$ represent the strings $t_\mathcal{A}$ and $t_\mathcal{B}$ from $\{\mathsf{enc}(\alpha)\mid\alpha\in\mathsf{B}(\{c^\tau\mid c\in\mathsf{dom}(D)\}\cup\mathbf{BN},\mathbf{S})\}$, that states the following: $t_\mathcal{A}\vdash^i_\Sigma t_\mathcal{B}$ if $s=0$, and $t_\mathcal{A}\vdash^{i,t}_\Sigma t_\mathcal{B}$, with t being the last substring of length τ in $t_\mathcal{B}$, if $s=1$. Then, we define the crucial queries as:

$$\pi(\mathcal{A},\mathcal{B})=\exists s\ \Big(\pi_{\lceil\log f_{\mathbb{L}_\mathsf{N}}(\Sigma,q)\rceil}(\mathcal{A},\mathcal{B},s)\wedge\mathrm{Zero}(s)\Big)$$
$$\pi_G(\mathcal{A},\mathcal{B})=\exists s\ \Big(\pi_{\lceil\log f_{\mathbb{L}_\mathsf{N}}(\Sigma,q)\rceil}(\mathcal{A},\mathcal{B},s)\wedge\mathrm{One}(s)\Big).$$

Let us now proceed with the formal definition of π_i. To this end, given two tuples $\mathcal{A},\mathcal{B}$ that represent the strings $t_\mathcal{A}$ and $t_\mathcal{B}$ from $\{\mathsf{enc}(\alpha)\mid\alpha\in\mathsf{B}(\{c^\tau\mid c\in\mathsf{dom}(D)\}\cup\mathbf{BN},\mathbf{S})\}$, we need a way to check whether $\mathsf{dec}(t_\mathcal{B})$ can be obtained from $\mathsf{dec}(t_\mathcal{A})$ via a single chase step that uses a linear TGD $\sigma\in\Sigma$. This is achieved via the subquery $\widehat{\sigma}(\mathcal{AB},s)$, where $\mathcal{AB}$ denotes the tuple $\mathcal{A},\mathcal{B}$ (i.e., the tuple obtained by appending $\mathcal{B}$ after $\mathcal{A}$) that has the following meaning: if $s=0$, then $\mathsf{dec}(t_\mathcal{B})$ can be obtained from $\mathsf{dec}(t_\mathcal{A})$ by applying σ, and if $s=1$, then in addition the string of length τ from $\mathbf{BN}$ occurring at the last position of $\mathsf{dec}(t_\mathcal{B})$ is invented in $\mathsf{dec}(t_\mathcal{B})$. Before giving the formal definition, we need an auxiliary function.

Assume that σ is a linear TGD of the form

$$R(t_{\omega+2-\mathsf{ar}(R)},\ldots,t_{\omega+1})\ \rightarrow\ P(t_{2\omega+3-\mathsf{ar}(P)},\ldots,t_{2\omega+2}).$$

Let $\mathit{Ind}=\{1,\ldots,\omega+1-\mathsf{ar}(R),\omega+2,\ldots,2\omega+2-\mathsf{ar}(P)\}$. These are the positions in the encodings of $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$ where either the encoding of the predicates, or the special string 0^τ occurs. In other words, in those positions a non-valid term appears. We can now define the function $\xi_\sigma:[2\omega+2]\rightarrow[2\omega+2]$ as follows:

$$\xi_\sigma(i) = \begin{cases} i & \text{if } i \in \mathit{Ind}, \\ i & \text{if } t_i \notin \{t_{i+1}, \ldots, t_{2\omega+2}\} \setminus \{t_j \mid j \in \mathit{Ind}\}, \\ \mu_i & \text{otherwise}, \end{cases}$$

where $\mu_i = \min\{j \mid j \in \{i+1, \ldots, 2\omega+2\} \setminus \mathit{Ind} \text{ and } t_i = t_j\}$. In words, given the position i of a variable v occurring in σ, $\xi_\sigma(i)$ is the next position (from left-to-right) where the same variable v (if any) occurs in σ. For example, assuming that $\omega = 3$ and

$$\sigma \;=\; R(x_1, x_2) \;\rightarrow\; \exists x_3\, P(x_2, x_1, x_3),$$

we have that $\xi_\sigma(1) = 1$, $\xi_\sigma(2) = 2$, $\xi_\sigma(3) = 7$, $\xi_\sigma(4) = 6$, $\xi_\sigma(5) = 5$, $\xi_\sigma(6) = 6$, $\xi_\sigma(7) = 7$, and $\xi_\sigma(8) = 8$.

We are now ready to define $\hat{\sigma}$. In what follows, $\mathcal{A}$ and $\mathcal{B}$ should be seen as tuples consisting of $\omega+1$ tuples, where the first one is of length $|\mathbf{S}|$, while the rest are of length τ, and by $\mathcal{A}[i]$ or $\mathcal{B}[i]$ we refer to the i-th tuple of $\mathcal{A}$ and $\mathcal{B}$, respectively. We also write $\mathcal{AB}[i]$ for the i-th tuple of $\mathcal{AB}$. We consider the following two cases where σ contains or not an existentially quantified variable.

- If σ contains an existentially quantified variable, then $\widehat{\sigma}(\mathcal{AB}, s)$ is defined as

$$\mathrm{Null}(\mathcal{B}[\omega+1]) \;\wedge\; (\mathrm{Zero}(s) \vee \mathrm{One}(s)) \;\wedge \bigwedge_{i \in [2\omega+2]} \mathcal{AB}[i] = \mathcal{AB}[\xi_\sigma(i)] \;\wedge \bigwedge_{i \in \{2,\ldots,\omega+1\}} \neg(\mathcal{A}[i] = \mathcal{B}[\omega+1]).$$

- If σ does not contain an existentially quantified variable, then $\widehat{\sigma}(\mathcal{AB}, s)$ is

$$\mathrm{Zero}(s) \;\wedge \bigwedge_{i \in [2\omega+2]} \mathcal{AB}[i] = \mathcal{AB}[\xi_\sigma(i)].$$

We now have all the ingredients to define π_i. This is done inductively as follows. The query $\pi_0(\mathcal{A}, \mathcal{B}, s)$ is defined as

$$\mathcal{A} = \mathcal{B} \;\vee \bigvee_{\sigma \in \Sigma} \hat{\sigma}(\mathcal{AB}, s).$$

Note that the shortcut $\mathcal{A} = \mathcal{B}$ is not defined in Table 1, but it can be defined in the same way as $\mathbf{U} = \mathbf{W}$. Then, $\pi_{i+1}(\mathcal{A}, \mathcal{B}, s)$ is defined as

$$\begin{aligned} &\exists \mathcal{C}\, (\mathrm{Atom}(\mathcal{C}) \wedge \\ &\bigwedge_{i \in \{2,\ldots,\omega+1\}} \left(\bigvee_{j \in \{2,\ldots,\omega+1\}} \mathcal{A}[i] = \mathcal{B}[j] \;\rightarrow\; \bigvee_{j \in \{2,\ldots,\omega+1\}} \mathcal{A}[i] = \mathcal{C}[j] \right) \wedge \\ &\forall \mathcal{D} \forall \mathcal{E} \forall s' \Big(\mathrm{Atom}(\mathcal{D}) \;\wedge\; \mathrm{Atom}(\mathcal{E}) \;\wedge\; \Big(\big(\mathcal{D} = \mathcal{A} \;\wedge\; \mathcal{E} = \mathcal{C} \;\wedge\; \mathrm{Zero}(s')\big) \;\vee \\ &\big(\mathcal{D} = \mathcal{C} \;\wedge\; \mathcal{E} = \mathcal{B} \;\wedge\; s = s'\big) \Big) \rightarrow \pi_i(\mathcal{D}, \mathcal{E}, s') \Big). \end{aligned}$$

It checks whether there exists a string $t_\mathcal{C} \in \{\mathsf{enc}(\alpha) \mid \alpha \in \mathsf{B}(\{c^\tau \mid c \in \mathsf{dom}(D)\} \cup \mathbf{BN}, \mathbf{S})\}$, represented by $\mathcal{C}$, such that, assuming $\mathcal{A}, \mathcal{B}$ correspond to the strings $t_\mathcal{A}, t_\mathcal{B}$, respectively, $t_\mathcal{A} \vdash_\Sigma^{2^i} t_\mathcal{C}$ and $t_\mathcal{C} \vdash_\Sigma^{2^i} t_\mathcal{B}$ if $s = 0$, and $t_\mathcal{C} \vdash_\Sigma^{2^i, t} t_\mathcal{B}$ if $s = 1$, with t being the last substring of length τ in $t_\mathcal{B}$.

The definition of the query $f_\mathsf{Q}(q, \Sigma)$ is now complete. It can be verified that the following technical lemma holds – in fact, it follows by construction – which essentially states the correctness of the query rewriter f_Q.

Lemma 4.3. *Consider a database D, a set $\Sigma \in \mathbb{L}_\mathsf{N}$ of TGDs, a CQ $q(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$. Let $q_\Sigma = f_\mathsf{Q}(q, \Sigma)$. The following hold:*

1. *$\bar{c} \in q_\Sigma(D_{01})$ iff there exists a binary $\lceil \log f_{\mathbb{L}_\mathsf{N}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.*
2. *q_Σ can be computed in polynomial time.*

By combining Lemmas 4.1, 4.2 and 4.3, we get that there exists a polynomial-time computable query rewriter f_Q such that, for every database D, $\Sigma \in \mathbb{L}_\mathsf{N}$, and CQ q, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ with $q_\Sigma = f_\mathsf{Q}(q, \Sigma)$, and item (1) of Proposition 4.1 follows.

4.4. Item (2) of Proposition 4.1

Let us now proceed with the second item of Proposition 4.1, which we recall here:

for a family of schemas $\mathbb{S}$ of fixed arity, there is a polynomial-time computable query rewriter $f_{\mathsf{Q}}^{\mathbb{S}} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{L}_{\mathsf{N}}^{\mathbb{S}} \to \exists\mathbb{F}\mathbb{O}^{+}$ such that for every $D \in \mathbb{D}[\mathbb{S}]$, $\Sigma \in \mathbb{L}_{\mathsf{N}}^{\mathbb{S}}$, and $q \in \mathbb{CQ}[\mathbb{S}]$, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ with $q_\Sigma = f_{\mathsf{Q}}^{\mathbb{S}}(q, \Sigma)$.

By Lemmas 4.1 and 4.2, it suffices to define $f_{\mathsf{Q}}^{\mathbb{S}}$ in such a way that, for a database $D \in \mathbb{D}[\mathbb{S}]$, a set $\Sigma \in \mathbb{L}_{\mathsf{N}}^{\mathbb{S}}$ of TGDs, a CQ $q(\bar{x}) \in \mathbb{CQ}[\mathbb{S}]$, and a tuple $\bar{c}$, the following are equivalent; we can assume, without loss of generality, that D, Σ, and q are over the same schema $\mathbf{S} \in \mathbb{S}$ that collects all the predicates occurring in D, Σ, and q:

1. $\bar{c} \in q_\Sigma(D_{01})$ with $q_\Sigma = f_{\mathsf{Q}}^{\mathbb{S}}(q, \Sigma)$.
2. There exists a binary $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q) \rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ.

Consider a set $\Sigma \in \mathbb{L}_{\mathsf{N}}^{\mathbb{S}}$ and a CQ $q(\bar{x}) \in \mathbb{CQ}[\mathbb{S}]$, and assume that both Σ and q are over the same schema $\mathbf{S} \in \mathbb{S}$. The $\exists\mathrm{FO}^{+}$ query $f_{\mathsf{Q}}^{\mathbb{S}}(q, \Sigma)$ is

$$\exists \mathbf{Q} \left(\mathrm{Triple}^{+}(\mathbf{Q}) \wedge \mathrm{WitnessGeneratorScheme}^{+}(\mathbf{Q}) \wedge \mathrm{WitnessGenerator}^{+}(\mathbf{Q}) \right)$$

with $\mathrm{N}^{+}(\mathbf{Q})$, where $\mathrm{N} \in \{\mathrm{Triple}, \mathrm{WitnessGeneratorScheme}, \mathrm{WitnessGenerator}\}$, being an adaptation of the query $\mathrm{N}(\mathbf{Q})$ defined in the previous section. In fact, the goal is to convert the FO query $\mathrm{N}(\mathbf{Q})$ into an equivalent $\exists\mathrm{FO}^{+}$ query $\mathrm{N}^{+}(\mathbf{Q})$, but without losing the crucial property that it can be constructed in polynomial time.

The Subqueries $\mathrm{Triple}^{+}(\mathbf{Q})$ and $\mathrm{WitnessGeneratorScheme}^{+}(\mathbf{Q})$. We first observe that $\mathrm{Triple}(\mathbf{Q})$ and $\mathrm{WitnessGeneratorScheme}(\mathbf{Q})$ can be transformed in polynomial time into equivalent queries $\mathrm{Triple}^{\mathsf{nnf}}(\mathbf{Q})$ and $\mathrm{WitnessGeneratorScheme}^{\mathsf{nnf}}(\mathbf{Q})$ in *negation normal form*, that is, the negation operator is only applied to atoms, and the only other allowed Boolean operators are $\wedge$ and $\vee$. It is easy to verify that the only reasons why the obtained queries in negation normal form are not positive are:

- expressions of the form $\neg(s = s')$, i.e., inequalities, where the witnesses for the variables s, s' can only be 0 and 1 (not constants from the input database), and
- atoms of the form $\neg\mathrm{Zero}(s)$ or $\neg\mathrm{One}(s)$.

Since the variables s and s' in expressions $\neg(s = s')$ can only take the values 0 and 1, we can simply replace this kind of inequalities with the positive expression

$$\left(\mathrm{Zero}(s) \vee \mathrm{Zero}(s')\right) \wedge \left(\mathrm{One}(s) \vee \mathrm{One}(s')\right).$$

We can also replace $\neg\mathrm{Zero}(s)$ and $\neg\mathrm{One}(s)$ with the equivalent positive expression

$$\mathrm{One}(s) \vee \mathrm{ADom}(s) \qquad \text{and} \qquad \mathrm{Zero}(s) \vee \mathrm{ADom}(s),$$

respectively, where $\mathrm{ADom}(s)$ is a subquery that computes the active domain of the input database, i.e., the set of constants occurring in the input database. More precisely, the query $\mathrm{ADom}(s)$ is defined as

$$\bigvee_{R \in \mathbf{S}} \bigvee_{i \in [\mathrm{ar}(R)]} \exists s_1 \cdots \exists s_{i-1} \exists s_{i+1} \cdots \exists s_{\mathrm{ar}(R)}\, R(s_1, \ldots, s_{i-1}, s, s_{i+1}, \ldots, s_{\mathrm{ar}(R)}).$$

The above replacements in $\mathrm{Triple}^{\mathsf{nnf}}(\mathbf{Q})$ and $\mathrm{WitnessGeneratorScheme}^{\mathsf{nnf}}(\mathbf{Q})$ lead to the positive existential queries $\mathrm{Triple}^{+}(\mathbf{Q})$ and $\mathrm{ProofGeneratorScheme}^{+}(\mathbf{Q})$.

The Subquery $\mathrm{WitnessGenerator}^{+}(\mathbf{Q})$. We now convert $\mathrm{WitnessGenerator}(\mathbf{Q})$ into an equivalent positive existential query by applying the following three steps:

- We first redefine the subqueries π and π_G by relying on a different definition of $\pi_{\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma,q) \rceil}(\mathcal{A}, \mathcal{B}, s)$. In particular, the query $\pi_0(\mathcal{A}, \mathcal{B}, s)$ is defined as

  $$\mathcal{A} = \mathcal{B} \vee \bigvee_{\sigma \in \Sigma} \hat{\sigma}(\mathcal{A}\mathcal{B}, s),$$

  which is actually the same as before, and then $\pi_{i+1}(\mathcal{A}, \mathcal{B}, s)$ is defined as

  $$\exists \mathcal{C} \exists s' \Bigg(\mathrm{Atom}(\mathcal{C}) \wedge \bigwedge_{i \in \{2, \ldots, \omega+1\}} \mathrm{Null}(\mathcal{A}[i]) \rightarrow \left(\bigvee_{j \in \{2, \ldots, \omega+1\}} \mathcal{A}[i] = \mathcal{B}[j] \rightarrow \bigvee_{j \in \{2, \ldots, \omega+1\}} \mathcal{A}[i] = \mathcal{C}[j] \right) \wedge \pi_i(\mathcal{A}, \mathcal{C}, s') \wedge \mathrm{Zero}(s') \wedge \pi_i(\mathcal{C}, \mathcal{B}, s) \Bigg).$$

Note that during the propagation check we focus on null values (see the expression Null($\mathcal{A}[i]$) that checks first whether $\mathcal{A}[i]$ encodes a null), which in turn guarantees that the equalities used are among variables that can only be instantiated with the values 0 and 1. This can be done without affecting the correctness of the construction since, assuming that $t_{\mathcal{A}}, t_{\mathcal{B}}, t_{\mathcal{C}}$ are the atom encodings represented by $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$, respectively, if $\mathsf{dec}(t_{\mathcal{C}})$ does not contain a string from **BC** that occurs in both $\mathsf{dec}(t_{\mathcal{A}})$ and $\mathsf{dec}(t_{\mathcal{B}})$, then $\pi_i(\mathcal{C}, \mathcal{B}, s)$ is trivially evaluated to false. Let WitnessGenerator*(**Q**) be the query defined as WitnessGenerator(**Q**) but using the new definition of π and π_G given above.
Before we proceed further, it is important to stress that, since $\mathbb{S}$ is of fixed arity, $\pi_{\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma,q)\rceil}(\mathcal{A}, \mathcal{B}, s)$ can be constructed in polynomial time. Without assuming that the arity is fixed, the construction of $\pi_{\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma,q)\rceil}(\mathcal{A}, \mathcal{B}, s)$ takes exponential time. This is precisely the reason why this simpler construction, which avoids the use of universally quantified variables, cannot be used in the proof of item (1) of Proposition 4.1.

- We now observe that WitnessGenerator*(**Q**) can be transformed into an equivalent query WitnessGenerator$^{\mathsf{nnf}}$(**Q**) in negation normal form, where the negation operator is only applied to equalities ($s = s'$), where the witnesses for the variables s, s' can only be 0 and 1, and atoms of the form Zero(s) or One(s).
- We finally convert WitnessGenerator$^{\mathsf{nnf}}$(**Q**) into the positive existential query WitnessGenerator^{+}(**Q**) by eliminating negation as explained above: simply replace $\neg(s = s')$ with $(\mathrm{Zero}(s) \vee \mathrm{Zero}(s')) \wedge (\mathrm{One}(s) \vee \mathrm{One}(s'))$, $\neg\mathrm{Zero}(s)$ with $\mathrm{One}(s) \vee \mathrm{ADom}(s)$, and $\neg\mathrm{One}(s)$ with $\mathrm{Zero}(s) \vee \mathrm{ADom}(s)$.

It is not a difficult task to verify that the following technical lemma holds, which essentially states the correctness of the query rewriter $f^{\mathbb{S}}_{\mathsf{Q}}$.

Lemma 4.4. *Let $\mathbb{S}$ be a family of schemas of fixed arity. Consider a database $D \in \mathbb{D}[\mathbb{S}]$, a set $\Sigma \in \mathbb{L}^{\mathbb{S}}_{\mathsf{N}}$ of TGDs, a CQ $q(\bar{x}) \in \mathbb{CQ}[\mathbb{S}]$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{x}|}$. With $q_\Sigma = f^{\mathbb{S}}_{\mathsf{Q}}(q, \Sigma)$ the following statements hold*[11]*:*

1. $\bar{c} \in q_\Sigma(D_{01})$ *iff there exists a binary $\lceil \log f_{\mathbb{L}_{\mathsf{N}}}(\Sigma, q)\rceil$-witness generator for $q(\bar{c})$ w.r.t. D and Σ*
2. q_Σ *is an $\exists\mathrm{FO}^+$ query that can be computed in polynomial time.*

By Lemmas 4.1, 4.2 and 4.4, we conclude that, for a family of schemas $\mathbb{S}$ of fixed arity, there exists a polynomial-time computable query rewriter $f^{\mathbb{S}}_{\mathsf{Q}} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{L}^{\mathbb{S}}_{\mathsf{N}} \to \exists\mathbb{FO}^+$ such that, for every $D \in \mathbb{D}[\mathbb{S}]$, $\Sigma \in \mathbb{L}^{\mathbb{S}}_{\mathsf{N}}$, and $q \in \mathbb{CQ}[\mathbb{S}]$, $\mathsf{cert}(q, D, \Sigma) = q_\Sigma(D_{01})$ with $q_\Sigma = f^{\mathbb{S}}_{\mathsf{Q}}(q, \Sigma)$, as needed. This completes the proof of Proposition 4.1.

5. Polynomial combined rewritability and guardedness

We now concentrate on the notion of polynomial combined rewritability for guarded TGDs. A family of schemas $\mathbb{S}$ is of fixed size if there exists an integer $k \geq 0$ such that, for every schema $\mathbf{S} \in \mathbb{S}$, $|\mathbf{S}| \leq k$ and $\mathsf{ar}(\mathbf{S}) \leq k$. Our goal is to show the following:

Theorem 5.1. *For a family of schemas $\mathbb{S}$ of fixed size, $\mathbb{G}[\mathbb{S}]$ is polynomially combined $\exists\mathrm{FO}^+$-rewritable.*

Recall that the evaluation problem for $\exists\mathbb{FO}^+$ is in NP. Therefore, an immediate consequence of Theorem 5.1 is the following complexity result, which, as discussed in the Introduction, closes a complexity gap for $\mathsf{OQA}(\mathbb{G})$ in the case of schemas of fixed size. In fact, the following result is known for single-head guarded TGDs [17], but it cannot be straightforwardly transferred to multi-head guarded TGDs.

Corollary 5.1. *$\mathsf{OQA}(\mathbb{G})$ is* NP*-complete for schemas of fixed size.*

The proof of Theorem 5.1 exploits Proposition 5.1 below, which is the main technical result of this section. It essentially provides a polynomial-time combined reduction from ontological query answering under guarded TGDs over schemas of fixed size, to ontological query answering under linear TGDs over schemas of fixed arity. Before introducing Proposition 5.1, we need some auxiliary notions.

A *TGD rewriter* is a function that takes as input a set of TGDs and outputs a set of TGDs. For families of schemas $\mathbb{S}$ and $\mathbb{S}'$, we say that $\mathbb{G}[\mathbb{S}]$ is *polynomially combined reducible to* $\mathbb{L}[\mathbb{S}']$, written $\mathbb{G}[\mathbb{S}] \leq_{\mathsf{pc}} \mathbb{L}[\mathbb{S}']$, if there exist

- a polynomial-time computable database rewriter $f^{\mathbb{S}}_{\mathsf{DB}} : \mathbb{D}[\mathbb{S}] \times \mathbb{G}[\mathbb{S}] \to \mathbb{D}[\mathbb{S}']$,
- a polynomial-time computable TGD rewriter $f^{\mathbb{S}}_{\mathsf{T}} : \mathbb{G}[\mathbb{S}] \to \mathbb{L}[\mathbb{S}']$, and
- a polynomial-time computable query rewriter $f^{\mathbb{S}}_{\mathsf{Q}} : \mathbb{CQ}[\mathbb{S}] \times \mathbb{G}[\mathbb{S}] \to \mathbb{CQ}[\mathbb{S}']$,

[11] We assume, without loss of generality, that D, Σ, and q are over the same schema $\mathbf{S} \in \mathbb{S}$.

such that, for every $D \in \mathbb{D}[\mathbb{S}]$, $\Sigma \in \mathbb{G}[\mathbb{S}]$, and $q \in \mathbb{CQ}[\mathbb{S}]$, it holds that $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(f^{\mathbb{S}}_{\mathsf{Q}}(q, \Sigma), f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma), f^{\mathbb{S}}_{\mathsf{T}}(\Sigma))$. We show the following:

Proposition 5.1. *For a family of schemas $\mathbb{S}$ of fixed size, there is a family of schemas $\mathbb{S}'$ of fixed arity such that $\mathbb{G}[\mathbb{S}] \leq_{\mathsf{pc}} \mathbb{L}[\mathbb{S}']$.*

Observe that polynomial combined $\exists\mathsf{FO}^+$-rewritability is closed under $\leq_{\mathsf{pc}}$. Since, by Theorem 4.1, $\mathbb{L}[\mathbb{S}']$ is polynomially combined $\exists\mathsf{FO}^+$-rewritable, Proposition 5.1 immediately implies Theorem 5.1. The rest of the section is devoted to showing Proposition 5.1. This is done in three main steps:

1. We define the database, TGD and query rewriters $f^{\mathbb{S}}_{\mathsf{DB}}$, $f^{\mathbb{S}}_{\mathsf{T}}$ and $f^{\mathbb{S}}_{\mathsf{Q}}$, respectively.
2. We then establish their correctness, that is, for every $D \in \mathbb{D}[\mathbb{S}]$, $\Sigma \in \mathbb{G}[\mathbb{S}]$, and $q \in \mathbb{CQ}[\mathbb{S}]$, $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(f^{\mathbb{S}}_{\mathsf{Q}}(q, \Sigma), f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma), f^{\mathbb{S}}_{\mathsf{T}}(\Sigma))$.
3. We finally show that $f^{\mathbb{S}}_{\mathsf{DB}}$, $f^{\mathbb{S}}_{\mathsf{T}}$ and $f^{\mathbb{S}}_{\mathsf{Q}}$ are polynomial-time computable.

Before proceeding with the above three steps, we first need to introduce a notion that is central when reasoning with guarded TGDs, known as the type of an atom. This notion will play an important role in all of the above steps.

5.1. Guarded types

Consider a database D and a set $\Sigma \in \mathbb{G}$. Let $\delta = (I_i)_{i \geq 0}$ be a (finite or infinite) chase derivation of D w.r.t. Σ with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$. Given an atom $\alpha \in \mathsf{chase}_\delta(D, \Sigma)$, its *$\delta$-type (w.r.t. D and Σ)*, is the set of atoms

$$\mathsf{type}_\delta(\alpha) \;=\; \{\beta \in \mathsf{chase}_\delta(D, \Sigma) \mid \mathsf{dom}(\beta) \subseteq \mathsf{dom}(\alpha)\}$$

that collects all the atoms in the result of δ that mention only terms occurring in α. The key property of the type has been shown in [17] for single-head guarded TGDs, and it can be easily extended to multi-head guarded TGDs. It roughly states that the set of atoms in $\mathsf{chase}_\delta(D, \Sigma)$ that can be derived from α (used as a guard) is determined by its δ-type. To make this precise we need some auxiliary notions.

The *guarded parent relation* of δ, denoted $\prec^{gp}_\delta$, is a binary relation on $\mathsf{chase}_\delta(D, \Sigma)$ such that $\alpha \prec^{gp}_\delta \beta$ iff there exists $i \geq 0$ with $\alpha = h_i(\mathsf{guard}(\sigma_i))$ and $\beta \in I_{i+1} \setminus I_i$. Let $\prec^{gp,+}_\delta$ be the transitive closure of $\prec^{gp}_\delta$. Note that $\prec^{gp}_\delta$ forms a forest with the atoms of D being the roots. We can now define the notion of projection of δ, which will allow us to state the key property of the type. Consider an atom $\alpha \in \mathsf{chase}_\delta(D, \Sigma)$, and let $(i_j)_{j>0}$ be the sequence of indices with $0 \leq i_1 < i_2 < i_3 < \cdots$ such that, for each $\ell \geq 0$, $\ell \in \{i_j\}_{j>0}$ iff $h_\ell(\mathsf{guard}(\sigma_\ell)) = \alpha$ or $\alpha \prec^{gp,+}_\delta h_\ell(\mathsf{guard}(\sigma_\ell))$. Simply stated, $(i_j)_{j>0}$ collects all the applications in δ, in ascending order, that use α or a $\prec^{gp}_\delta$-descendant of α as the guard. The *α-projection of δ*, denoted $\delta[\alpha]$, is the sequence of instances $(J_i)_{i \geq 0}$, with $J_0 = \mathsf{type}_\delta(\alpha)$, and, for each $j > 0$,

$$J_j \;=\; J_{j-1} \cup \left\{\beta \in I_{i_j+1} \mid h_{i_j}(\mathsf{guard}(\sigma_{i_j})) \prec^{gp}_\delta \beta\right\}.$$

We can now state the key property of the notion of type, which relies on guardedness:

Lemma 5.1. *Consider a database D and a set $\Sigma \in \mathbb{G}$. For a chase derivation δ of D w.r.t. Σ and an atom $\alpha \in \mathsf{chase}_\delta(D, \Sigma)$, $\delta[\alpha]$ is a chase derivation of $\mathsf{type}_\delta(\alpha)$ w.r.t. Σ.*

Proof. Let $\delta = (I_i)_{i \geq 0}$ with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$. By definition, $\delta[\alpha] = (J_i)_{i \geq 0}$, with $J_0 = \mathsf{type}_\delta(\alpha)$, and, for each $j > 0$,

$$J_j \;=\; J_{j-1} \cup \left\{\beta \in I_{i_j+1} \mid h_{i_j}(\mathsf{guard}(\sigma_{i_j})) \prec^{gp}_\delta \beta\right\},$$

where $(i_j)_{j>0}$ is the sequence of indices, with $0 \leq i_1 < i_2 < i_3 < \cdots$ such that, for each $\ell \geq 0$, $\ell \in \{i_j\}_{j>0}$ iff $h_\ell(\mathsf{guard}(\sigma_\ell)) = \alpha$ or $\alpha \prec^{gp,+}_\delta h_\ell(\mathsf{guard}(\sigma_\ell))$. We proceed to show the following three statements, which in turn imply that $\delta[\alpha]$ is a chase derivation of $\mathsf{type}_\delta(\alpha)$ w.r.t. Σ, as needed:

1. For each $j > 0$, $J_{j-1} \langle \sigma_{i_j}, h_{i_j} \rangle J_j$.
2. For $j, k > 0$ such that $j \neq k$, $(\sigma_{i_j}, h_{i_j}) \neq (\sigma_{i_k}, h_{i_k})$.
3. For each $j \geq 0$, and for every trigger (σ, h) for Σ on J_j, there exists $k \geq j$ such that $(\sigma, h) = (\sigma_{i_{k+1}}, h_{i_{k+1}})$.

Item (1) is a consequence of the following observations, which in turn hold due to the fact that Σ is guarded; for brevity, for a guarded TGD σ, we write $\mathsf{side}(\sigma)$ for the side atoms of σ, i.e., the set of atoms $\mathsf{body}(\sigma) \setminus \{\mathsf{guard}(\sigma)\}$:

- $h_{i_1}(\mathsf{side}(\sigma_{i_1})) \subseteq \mathsf{type}_\delta(\alpha)$.
- For each $j > 1$, $h_{i_j}(\mathsf{side}(\sigma_{i_j})) \subseteq \mathsf{type}_\delta(\alpha) \cup \{\beta \in I_{i_{j-1}+1} \mid \alpha \prec^{gp,+}_\delta \beta\}$.

Items (2) and (3) hold since, by hypothesis, δ is a chase derivation. Indeed, by contradiction, if (2) (resp. (3)) does not hold, then δ applies a certain trigger more than once (resp., is not fair), which contradicts the fact that it is a chase derivation. □

With the notion of type in place, we are now ready to proceed with the proof of Proposition 5.1. In the rest of the section, let $\mathbb{S}$ be a family of schemas of fixed size.

5.2. The database, TGD and query rewriters

Before we define the desired rewriters, we first need to introduce some auxiliary technical notions that are crucial for the definitions.

Auxiliary Notions. We need to encode the equality type of a guard atom (i.e., which of its positions mention the same term), as well as the equality type of the atoms occurring in the type of such a guard. This is done via the notion of **S**-type, where **S** is a schema. The equality type of an atom can be encoded via an atom that mentions only integer values. For example, the equality type of $R(a, b, a, c)$ can be encoded via the atom $R(1, 2, 1, 3)$. Formally, an **S**-*type* τ is a pair (α, T), where $\alpha = R(t_1, \ldots, t_n)$, $R \in \mathbf{S}$, $t_1 = 1$, $t_i \in \{t_1, \ldots, t_{i-1}, t_{i-1} + 1\}$ for $i \in \{2, \ldots, n\}$, and $T \subseteq \mathsf{B}(\{t_1, \ldots, t_n\}, \mathbf{S}) \setminus \{\alpha\}$. We write $\mathsf{guard}(\tau)$ for the atom α, $\mathsf{atoms}(\tau)$ for the set of atoms $(\{\alpha\} \cup T)$, and $\mathsf{ar}(\tau)$ for the maximum integer occurring in $\mathsf{guard}(\tau)$. We further write $\mathbf{S}^+$ for the schema $\mathbf{S} \cup \{[\tau] \mid \tau \text{ is an } \mathbf{S}\text{-type}\}$, where $[\tau]$ is a new predicate not in **S** of arity $\mathsf{ar}(\tau)$; it is clear that $\mathsf{ar}(\mathbf{S}) = \mathsf{ar}(\mathbf{S}^+)$. The family of schemas $\mathbb{S}^+$ is defined as $\{\mathbf{S}^+ \mid \mathbf{S} \in \mathbb{S}\}$. It is clear that $\mathbb{S}^+$ is of fixed arity since $\mathbb{S}$ is of fixed size (and thus, of fixed arity). We say that a tuple $\bar{u} = (u_1, \ldots, u_n)$ is isomorphic to $\bar{t} = (t_1, \ldots, t_n)$, written $\bar{u} \simeq \bar{t}$, if $u_i = u_j$ iff $t_i = t_j$. Given a tuple $\bar{u}$ such that $\bar{u} \simeq \bar{t}$, the *instantiation* of τ with $\bar{u}$, denoted $\tau(\bar{u})$, is the set of atoms obtained from $\mathsf{atoms}(\tau)$ after replacing t_i with u_i. The *projection* of τ over $P \subseteq \{1, \ldots, \mathsf{ar}(\tau)\}$ is

$$\Pi_P(\tau) \;=\; \{\beta \in \mathsf{atoms}(\tau) \mid \mathsf{dom}(\beta) \subseteq P\}.$$

The *completion* of an instance I over **S** w.r.t. a set of TGDs Σ over **S** is the instance

$$\mathsf{complete}(I, \Sigma) \;=\; \{R(\bar{u}) \mid R \in \mathbf{S} \text{ and } \bar{u} \in \mathsf{cert}(R(\bar{x}), I, \Sigma)\},$$

where $\bar{u} = (u_1, \ldots, u_n)$ and $\bar{x} = (x_1, \ldots, x_n)$. Note that the completion of an instance w.r.t. a set of TGDs relies on the notion of certain answers. At first glance, this might seem circular since we are trying to devise a machinery for computing certain answers via the combined approach to FO-rewritability. However, the definition of completion only needs to compute the certain answers of atomic full queries, that is, CQs consisting of a single atom without existentially quantified variables, which is a much simpler task. Further details on the problem of computing the certain answers to atomic full queries, known as instance checking, are given in Section 5.4, where the polynomial-time computability of the rewriters is shown.

The Database Rewriter. Given a database $D \in \mathbb{D}[\mathbb{S}]$ and a set $\Sigma \in \mathbb{G}[\mathbb{S}]$, and assuming that both D and Σ are over a schema $\mathbf{S} \in \mathbb{S}$,

$$f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma) \;=\; \left\{ [\tau](\bar{c}) \;\middle|\; \begin{array}{l} R(\bar{c}) \in D \\ \tau \text{ is an } \mathbf{S}\text{-type of the from } (R(\bar{t}), T) \text{ with } \bar{c} \simeq \bar{t} \\ \tau(\bar{c}) \subseteq \mathsf{complete}(D, \Sigma) \end{array} \right\}.$$

It is clear that $f^{\mathbb{S}}_{\mathsf{DB}}$ is a database rewriter of the form $\mathbb{D}[\mathbb{S}] \times \mathbb{G}[\mathbb{S}] \to \mathbb{D}[\mathbb{S}^+]$. A simple example that illustrates the notion of the database rewriter introduced above follows.

Example 5.1. Let $\mathbf{S} = \{P, Q, R, S, T\} \in \mathbb{S}$. Consider the database

$$D \;=\; \{R(a, a, b, c)\}$$

over **S** and the set Σ of guarded TGDs over **S** consisting of

$$\sigma \;=\; P(x, y, x, u, w), S(x, u) \to \exists z_1 \exists z_2\, R(u, y, x, z_1), T(z_1, z_2, x),$$
$$\sigma' \;=\; R(x, x, y, z) \to Q(x, z).$$

It is clear that the pair

$$\tau \;=\; (R(1, 1, 2, 3), \{Q(1, 3)\})$$

is an **S**-type with $(a, a, b, c) \simeq (1, 1, 2, 3)$, and there is no other such **S**-type. Moreover, it is easy to verify that

$$\{R(a, a, b, c), Q(a, c)\} \;\subseteq\; \mathsf{complete}(D, \Sigma),$$

which in turn implies that $f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma) = \{[\tau](a, a, b, c)\}$. In simple words, $[\tau](a, a, b, c)$ is a compact encoding of the guard $R(a, a, b, c)$ and its type. □

The TGD Rewriter. Given a set $\Sigma \in \mathbb{G}[\mathbb{S}]$ over a schema $\mathbf{S} \in \mathbb{S}$, $f_{\mathsf{T}}^{\mathbb{S}}(\Sigma)$ is defined as the union of the sets of linear TGDs Σ_{tg} and Σ_{ex}, where

- Σ_{tg} is the so-called *type generator*, which is responsible for generating new $\mathbf{S}$-types from existing ones, and
- Σ_{ex} is the so-called *expander*, which is responsible for expanding a derived $\mathbf{S}$-type τ, i.e., it explicitly constructs the atoms over $\mathbf{S}$ encoded by τ.

The type generator is defined as follows. For each $\sigma \in \Sigma$

$$\varphi(\bar{x}, \bar{y}) \rightarrow \exists z_1 \cdots \exists z_m \, R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$$

with $\mathsf{guard}(\sigma) = G(\bar{u})$ and $(\bar{x} \cup \{z_i\}_{i \in [m]})$ being the variables occurring in $\mathsf{head}(\sigma)$, and for every $\mathbf{S}$-type τ such that there is a homomorphism h from $\varphi(\bar{x}, \bar{y})$ to $\mathsf{atoms}(\tau)$ and $h(G(\bar{u})) = \mathsf{guard}(\tau)$, we add to Σ_{tg} the linear TGD

$$[\tau](\bar{u}) \rightarrow \exists z_1 \cdots \exists z_m \, [\tau_1](\bar{u}_1), \ldots, [\tau_n](\bar{u}_n),$$

where, for each $i \in \{1, \ldots, n\}$, $[\tau_i]$ is defined as follows. Let f be the function from the variables in $\mathsf{head}(\sigma)$ to the natural numbers such that

$$f(t) = \begin{cases} h(t) & \text{if } t \in \bar{x}, \\ \mathsf{ar}(\mathbf{S}) + i & \text{if } t = z_i. \end{cases}$$

With $\bar{u}_i = (u_i^1, \ldots, u_i^{\mathsf{ar}(R_i)})$ and $\alpha_i = R_i(f(u_i^1), \ldots, f(u_i^{\mathsf{ar}(R_i)}))$, we define the set

$$T_i = \{\beta \in \mathsf{complete}(I, \Sigma) \mid \mathsf{dom}(\beta) \subseteq \mathsf{dom}(\alpha_i)\} \setminus \{\alpha_i\}$$

with

$$I = \{\alpha_1, \ldots, \alpha_n\} \cup \Pi_{\{h(x) \mid x \in \bar{x}\}}(\tau).$$

Note that (α_i, T_i) is not a proper $\mathbf{S}$-type since the integers in atoms do not appear in the correct order. Let ρ be the renaming function that renames the integers in atoms in order to appear in increasing order starting from 1 (e.g., $\rho(R(2,2,4,1)) = R(1,1,2,3)$; the formal definition is omitted since it is clear what the function ρ does). We then define τ_i as the $\mathbf{S}$-type $(\rho(\alpha_i), \rho(T_i))$.

The expander constructs the guard atom of τ, for each $\mathbf{S}$-type τ. To this end, for each $\mathbf{S}$-type τ, we add to Σ_{ex} the linear TGD

$$[\tau](x_1, \ldots, x_k) \rightarrow R(x_1, \ldots, x_k),$$

where R is the k-ary predicate of $\mathsf{guard}(\tau)$. It is clear that $f_{\mathsf{T}}^{\mathbb{S}}$ is indeed a TGD rewriter of the form $\mathbb{G}[\mathbb{S}] \rightarrow \mathbb{L}[\mathbb{S}^+]$. A simple example that illustrates the notion of the TGD rewriter introduced above follows.

Example 5.2. Let $\Sigma = \{\sigma, \sigma'\}$ be the set of guarded TGDs given in Example 5.1. Consider the $\mathbf{S}$-type

$$\tau = (P(1,2,1,2,3), \{S(1,2), S(1,1)\}).$$

It is easy to verify that $h = \{x \mapsto 1, y \mapsto 2, u \mapsto 2, w \mapsto 3\}$ is a homomorphism from $\mathsf{body}(\sigma)$ to $\mathsf{atoms}(\tau) = \{P(1,2,1,2,3), S(1,2), S(1,1)\}$ and $h(P(x,y,x,u,w)) = P(1,2,1,2,3)$. Therefore, the linear TGD

$$[\tau](x, y, x, u, w) \rightarrow \exists z_1 \exists z_2 \, [\tau_1](u, y, x, z_1), [\tau_2](z_1, z_2, x)$$

with

$$\tau_1 = (R(1,1,2,3), \{S(2,1), S(2,2), Q(1,3)\}) \quad \text{and} \quad \tau_2 = (T(1,2,3), \varnothing)$$

belongs to the type generator Σ_{tg}. Moreover, the linear TGDs

$$[\tau](x_1, \ldots, x_5) \rightarrow P(x_1, \ldots, x_5),$$
$$[\tau_1](x_1, \ldots, x_4) \rightarrow R(x_1, \ldots, x_4),$$
$$[\tau_2](x_1, x_2, x_3) \rightarrow T(x_1, x_2, x_3)$$

belong to the expander Σ_{ex}. □

The Query Rewriter. Due to the expander Σ_{ex} defined above, which explicitly constructs the atoms over the original schema $\mathbf{S} \in \mathbb{S}$ encoded by an $\mathbf{S}$-type τ, we can leave the query untouched. In other words, for $q \in \mathbb{CQ}[\mathbb{S}]$ and $\Sigma \in \mathbb{G}[\mathbb{S}]$,

$$f_{\mathsf{Q}}^{\mathbb{S}}(q, \Sigma) = q.$$

5.3. Correctness of the rewriters

We now proceed to establish the correctness of the database, TGD and query rewriters defined above. In fact, we need to show the following:

Lemma 5.2. *Consider a database $D \in \mathbb{D}[\mathbb{S}]$, a set $\Sigma \in \mathbb{G}[\mathbb{S}]$ of TGDs, and a CQ $q \in \mathbb{CQ}[\mathbb{S}]$. It holds that* $\mathsf{cert}(q, D, \Sigma) = \mathsf{cert}(q, f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma), f^{\mathbb{S}}_{\mathsf{T}}(\Sigma))$.

The proof of Lemma 5.2 relies on the following technical lemma; in fact, Lemma 5.2 is an immediate consequence of Lemma 5.3 shown below. We say that two instances I, J are homomorphically equivalent if $I \to J$ and $J \to I$. We also write $I_{|\mathbf{S}}$ for the restriction of I over a schema $\mathbf{S}$, i.e., the set of atoms $\{R(\bar{t}) \in I \mid R \in \mathbf{S}\}$. For brevity, we simply write $D^\star$ and $\Sigma^\star$ for $f^{\mathbb{S}}_{\mathsf{DB}}(D, \Sigma)$ and $f^{\mathbb{S}}_{\mathsf{T}}(\Sigma)$, respectively.

Lemma 5.3. *Consider a database $D \in \mathbb{D}[\mathbb{S}]$ and a set $\Sigma \in \mathbb{G}[\mathbb{S}]$ both over $\mathbf{S} \in \mathbb{S}$. Let δ be a chase derivation of D w.r.t. Σ and δ' be a chase derivation of $D^\star$ w.r.t. $\Sigma^\star$. It holds that* $\mathsf{chase}_\delta(D, \Sigma)$ *and* $\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}}$ *are homomorphically equivalent.*

Proof. We first need to establish two auxiliary technical results that reveal the relationship between $\mathsf{chase}_\delta(D, \Sigma)$ and $\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$. For an atom $\alpha \in \mathsf{chase}_\delta(D, \Sigma)$, let γ_α be the function that renames the arguments of α into integers in an increasing order; e.g., if $\alpha = R(a, \bot, a, b)$, then $\gamma_\alpha(\alpha) = R(1, 2, 1, 3)$. The definition of γ_α is obvious and is omitted. We also write τ^δ_α for the $\mathbf{S}$-type $(\gamma_\alpha(\alpha), \gamma_\alpha(\mathsf{type}_\delta(\alpha) \setminus \{\alpha\}))$. We are now ready to present and show the first auxiliary technical result.

Lemma 5.4. *Let δ be a chase derivation of D w.r.t. Σ and δ' be a chase derivation of $D^\star$ w.r.t. $\Sigma^\star$. There is a homomorphism h such that $R(\bar{t}) \in \mathsf{chase}_\delta(D, \Sigma)$ implies $[\tau^\delta_{R(\bar{t})}](h(\bar{t})) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$.*

Proof. Assume that $\delta = (I_i)_{i \geq 0}$. We show by induction that, for each $k \geq 0$, there is h_k such that $R(\bar{t}) \in I_k$ implies $[\tau^\delta_{R(\bar{t})}](h_k(\bar{t})) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$, whereas, for $k > 0$, h_k is compatible with h_{k-1}. This implies that the claim holds with $h = \bigcup_{i \geq 0} h_i$.

Base Case. Consider an atom $R(\bar{t}) \in I_0 = D$. By construction, $[\tau^\delta_{R(\bar{t})}](\bar{t}) \in D^\star$. Since $D^\star \subseteq \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$, the claim follows with h_0 being the identity on $\mathsf{dom}(D)$.

Inductive Step. Assume now that $R(\bar{t})$ has been generated during the k-the step for $k > 0$, i.e., $R(\bar{t}) \in I_k \setminus I_{k-1}$. Assume also that $I_{k-1} \langle \sigma, \mu \rangle I_k$ with (σ, μ) being a trigger for Σ on I_{k-1}; let σ be of the form

$$\varphi(\bar{x}, \bar{y}) \ \rightarrow \ \exists z_1 \cdots \exists z_m \, R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$$

with $\mathsf{guard}(\sigma) = G(\bar{u})$, and $(\bar{x} \cup \{z_i\}_{i \in [m]})$ be the variables occurring in $\mathsf{head}(\sigma)$. Clearly, μ is a homomorphism that maps $\varphi(\bar{x}, \bar{y})$ to I_{k-1}, and $I_k = I_{k-1} \cup \mu'(\mathsf{head}(\sigma))$. By the induction hypothesis, we conclude that

$$[\tau^\delta_{G(\mu(\bar{u}))}](h_{k-1}(\mu(\bar{u}))) \ \in \ \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star).$$

Observe that $(\gamma_{G(\mu(\bar{u}))} \circ \mu)$ maps $\varphi(\bar{x}, \bar{y})$ to $\mathsf{atoms}(\tau^\delta_{G(\mu(\bar{u}))})$ with $\gamma_{G(\mu(\bar{u}))}(\mu(G(\bar{u}))) = \mathsf{guard}(\tau^\delta_{G(\mu(\bar{u}))})$. Therefore, by construction, $\Sigma^\star$ contains a linear TGD σ_L of the form

$$[\tau^\delta_{G(\mu(\bar{u}))}](\bar{u}) \ \rightarrow \ \exists z_1 \cdots \exists z_m \, [\tau_1](\bar{u}_1), \ldots, [\tau_n](\bar{u}_n).$$

Furthermore, by Lemma 5.1, we can conclude that, for each $i \in [n]$, $\tau_i = \tau^\delta_{R_i(\mu'(\bar{u}_i))}$. Observe that $\lambda = h_{k-1} \circ \mu$ maps $\mathsf{body}(\sigma_L)$ to $\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$. Therefore, there exists an extension λ' of λ that maps $\{[\tau_i](\bar{u}_i)\}_{i \in [n]}$ to $\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$. Since $R(\bar{t}) = R_i(\mu'(\bar{u}_i))$ for some $i \in [n]$, we conclude that $[\tau^\delta_{R(\bar{y})}](\lambda'(\bar{u}_i)) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$. Let

$$h_k \ = \ h_{k-1} \ \cup \ \{\mu'(z_i) \mapsto \lambda'(z_i)\}_{i \in [m]};$$

note that, if σ has no existentially quantified variables, then $h_k = h_{k-1}$. It is clear that h_k is well-defined since none of the $\mu'(z_1), \ldots, \mu'(z_m)$ occurs in the domain of h_{k-1}. It is also easy to verify that $[\tau^\delta_{R(\bar{t})}](h_k(\bar{t})) = [\tau^\delta_{R(\bar{t})}](\lambda'(\bar{u}_i)) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$. □

We now proceed with the second technical claim needed for establishing Lemma 5.3. Let us note that, by construction, if $[\tau](\bar{t}) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$, where $\mathsf{guard}(\tau) = R(\bar{u})$, then $\bar{t} \simeq \bar{u}$. This means that, given an atom $[\tau](\bar{t}) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$, we can always refer to the instantiation of τ with $\bar{t}$, i.e., $\tau(\bar{t})$ is well-defined.

Lemma 5.5. *Let δ be a chase derivation of D w.r.t. Σ and δ' be a chase derivation of $D^\star$ w.r.t. $\Sigma^\star$. There is a homomorphism h such that $[\tau](\bar{t}) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)$ implies $h(\tau(\bar{t})) \subseteq \mathsf{chase}_\delta(D, \Sigma)$.*

Proof. Assume that $\delta' = (I_i)_{i \geq 0}$. We show by induction that, for each $k \geq 0$, there exists h_k such that $[\tau](\bar{t}) \in I_k$ implies $h_k(\tau(\bar{t})) \subseteq \mathsf{chase}_\delta(D, \Sigma)$, whereas, for $k > 0$, h_k is compatible with h_{k-1}. This implies that the claim holds with $h = \bigcup_{i \geq 0} h_i$.

Base Case. Consider $[\tau](\bar{t}) \in I_0 = D^\star$. By construction, $\tau(\bar{t}) \subseteq \mathsf{complete}(D, \Sigma)$. Since $\mathsf{complete}(D, \Sigma) \subseteq \mathsf{chase}_\delta(D, \Sigma)$, h_0 is the identity on $\mathsf{dom}(D)$.

Inductive Step. Assume now that $[\tau](\bar{t})$ has been generated during the k-th step for $k > 0$, i.e., $[\tau](\bar{t}) \in I_k \setminus I_{k-1}$. Assume also that $I_{k-1}\langle \sigma, \mu \rangle I_k$ with (σ, μ) being a trigger for $\Sigma^\star$ on I_{k-1}; let σ be of the form

$$[\tau](\bar{u}) \rightarrow \exists z_1 \cdots \exists z_m \, [\tau_1](\bar{u}_1), \ldots, [\tau_n](\bar{u}_n).$$

Clearly, μ is a homomorphism that maps $[\tau'](\bar{u})$ to I_{k-1}, and $I_k = I_{k-1} \cup \mu'(\mathsf{head}(\sigma))$. By the induction hypothesis, we conclude that

$$h_{k-1}(\tau'(\mu(\bar{u}))) \subseteq \mathsf{chase}_\delta(D, \Sigma).$$

By construction, there exists a TGD $\sigma_G \in \Sigma$ of the form

$$\varphi(\bar{x}, \bar{y}) \rightarrow \exists z_1 \cdots \exists z_m \, R_1(\bar{u}_1), \ldots, R_n(\bar{u}_n)$$

with $\mathsf{guard}(\sigma) = G(\bar{u})$ such that $\varphi(\bar{x}, \bar{y})$ is mapped to $\mathsf{atoms}(\tau')$ via a homomorphism λ, while $\lambda(G(\bar{u})) = \mathsf{guard}(\tau')$. Clearly, there is an isomorphism γ from $\mathsf{atoms}(\tau')$ to $\tau'(\mu(\bar{u}))$, which implies that $\mu = \gamma \circ \lambda$. Thus, $\nu = h_{k-1} \circ \mu$ maps $\varphi(\bar{x}, \bar{y})$ to $\mathsf{chase}_\delta(D, \Sigma)$. Therefore, there is an extension ν' of ν that maps $\{R_i(\bar{u}_i)\}_{i \in [n]}$ to $\mathsf{chase}_\delta(D, \Sigma)$. By Lemma 5.1, we can conclude that, for $i \in [n]$, $\tau_i = \tau^\delta_{R_i(\nu'(\bar{u}_i))}$. Let

$$h_k = h_{k-1} \cup \{\mu'(z_i) \mapsto \nu'(z_i)\}_{i \in [m]};$$

note that if σ has no existentially quantified variables, then $h_k = h_{k-1}$. It is clear that h_k is well-defined since none of the $\mu'(z_1), \ldots, \mu'(z_m)$ occurs in the domain of h_{k-1}. It is also easy to verify that $h_k(\tau(\bar{t})) = \tau_i(\nu'(\bar{u}_i))$ for some $i \in [n]$. Since

$$\tau_i(\nu'(\bar{u}_i)) = \mathsf{type}_\delta(R_i(\nu'(\bar{u}_i))) \subseteq \mathsf{chase}_\delta(D, \Sigma),$$

we conclude that $h_k(\tau(\bar{t})) \subseteq \mathsf{chase}_\delta(D, \Sigma)$, and the claim follows. □

Having Lemmas 5.4 and 5.5 in place, we can now conclude the proof of Lemma 5.3. We first show that

$$\mathsf{chase}_\delta(D, \Sigma) \rightarrow \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}}.$$

Let h be the homomorphism provided by Lemma 5.4. Due to the expander in $\Sigma^\star$, that is, the set of TGDs of the form $[\tau](\bar{x}) \rightarrow R(\bar{x})$, where τ is an $\mathbf{S}$-type and $R \in \mathbf{S}$, we can conclude that, for each atom $R(\bar{t}) \in \mathsf{chase}_\delta(D, \Sigma)$, $R(h(\bar{t})) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}}$. Therefore, $h(\mathsf{chase}_\delta(D, \Sigma)) \subseteq \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}}$, as needed.

We finally show that

$$\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}} \rightarrow \mathsf{chase}_\delta(D, \Sigma).$$

Let h be the homomorphism provided by Lemma 5.5. It is clear that

$$h\left(\bigcup_{[\tau](\bar{t}) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)} \tau(\bar{t}) \right) \subseteq \mathsf{chase}_\delta(D, \Sigma).$$

Since, by construction,

$$\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}} \subseteq \bigcup_{[\tau](\bar{t}) \in \mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)} \tau(\bar{t}),$$

we get that $h(\mathsf{chase}_{\delta'}(D^\star, \Sigma^\star)_{|\mathbf{S}}) \subseteq \mathsf{chase}_\delta(D, \Sigma)$, and the claim follows. □

As already said, Lemma 5.2, which establishes the correctness of the database, TGD and query rewriters defined above, is an immediate consequence of Lemma 5.3.

5.4. Polynomial-time computability of the rewriters

We finally proceed to establish that the rewriters defined above are polynomial-time computable. Actually, we only need to concentrate on the database and TGD rewriters since the query rewriter $f_{\mathsf{Q}}^{\mathbb{S}}$ is trivially computable in polynomial time. Let us recall that the family of schemas $\mathbb{S}$ is of fixed size, that is, for every schema $\mathbf{S} \in \mathbb{S}$, $|\mathbf{S}|$ and $\mathsf{ar}(\mathbf{S})$ are bounded by some integer. This is crucial for the validity of the next lemma:

Lemma 5.6. *The functions $f_{\mathsf{DB}}^{\mathbb{S}}$ and $f_{\mathsf{T}}^{\mathbb{S}}$ are polynomial-time computable.*

Observe that the definitions of both $f_{\mathsf{DB}}^{\mathbb{S}}$ and $f_{\mathsf{T}}^{\mathbb{S}}$ heavily rely on instance checking, which is a simpler version of ontological query answering where the query is an atomic full query, that is, a single atom without existentially quantified variables. Indeed, the notion of completion, which is used in the definitions of $f_{\mathsf{DB}}^{\mathbb{S}}$ and $f_{\mathsf{T}}^{\mathbb{S}}$, exploits instance checking. Therefore, to show Lemma 5.6, we first need to pinpoint the complexity of instance checking defined as follows; let $\mathbb{C}$ a class of TGDs:

PROBLEM :	$\mathsf{IC}(\mathbb{C})$
INPUT :	A database D, a set $\Sigma \in \mathbb{C}$, a CQ $R(\bar{x})$, and $\bar{c} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$.
QUESTION :	Does $\bar{c} \in \mathsf{cert}(R(\bar{x}), D, \Sigma)$?

It is clear that an $\mathbf{S}$-type, for a schema $\mathbf{S} \in \mathbb{S}$, depends only on $\mathbf{S}$. Since $\mathbb{S}$ is of fixed size, only a constant number of $\mathbf{S}$-types for schemas $\mathbf{S} \in \mathbb{S}$ can be formed that can be constructed in constant time. This implies that, if instance checking for $\mathbb{G}[\mathbb{S}]$ is feasible in polynomial time, then Lemma 5.6 follows, which, together with Lemma 5.2, establishes Proposition 5.1. The rest of the section is devoted to showing the following:

Lemma 5.7. *$IC(\mathbb{G}[\mathbb{S}])$ is in* PTIME.

Before we proceed further, let us clarify that the above result has been shown in [17] assuming single-head guarded TGDs, i.e., TGDs with only one atom in the head, via a sophisticated alternating algorithm that uses logarithmic space. However, the result from [17] cannot be straightforwardly transferred to multi-head guarded TGDs with an arbitrary conjunction of atoms in the head. The fact that we need to directly deal with multi-heads causes non-trivial complications that require novel ideas. Those complications were not an issue for single-head TGDs. To better understand the different nature of the two settings, let us stress that in the case of single-head guarded TGDs, by fixing the size of the underlying schema we implicitly fix the whole set of TGDs. Indeed, the number of different guarded non-isomorphic TGD-bodies that can be formed over a schema of fixed size is constant, which in turn implies that we can only have a constant number of different single-head guarded TGDs (up to variable renaming). This is not true for multi-head guarded TGDs since we can have an unbounded number of different TGDs with the same guarded body due to the unguarded multi-heads.

We now establish Lemma 5.7 via a novel alternating algorithm that is designed to directly operate on multi-head guarded TGDs. But first we need an auxiliary result.

Pivotal Atoms. Our alternating algorithm exploits the existence of some special atoms, called pivotal, for guarded subsets of the result of a derivation. Roughly, to check if a guarded set of atoms Q is in the result of some chase derivation δ, it suffices to check whether Q is in the result of the α-projection of δ with α being a pivotal atom for Q.

Consider a database D and a set of TGDs $\Sigma \in \mathbb{G}$, both over a schema $\mathbf{S}$. A finite set $Q \subseteq \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$ is *guarded* if there is $\alpha \in Q$ such that $\mathsf{dom}(\alpha) = \mathsf{dom}(Q)$. We say that Q is *ungrounded* if each of its atoms contains at least one null of $\mathbf{N}$. Consider now a chase derivation $\delta = (I_i)_{i \geq 0}$ of D w.r.t. Σ with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$ and a set of atoms $Q \subseteq \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$ that is guarded and ungrounded. An atom α is called *δ-pivotal for Q* if $\alpha = h_i(\mathsf{guard}(\sigma_i))$, for some $i \geq 0$, such that $\mathsf{null}(Q) \not\subseteq \mathsf{dom}(I_i)$, while $\mathsf{null}(Q) \subseteq \mathsf{dom}(I_{i+1})$. In simple words, a δ-pivotal atom for Q is an atom of $\mathsf{chase}_\delta(D, \Sigma)$ in which the nulls in Q occur together for the first time according to δ. The key lemma concerning pivotal atoms follows:

Lemma 5.8. *Consider a database D and a set of TGDs $\Sigma \in \mathbb{G}$, both over a schema $\mathbf{S}$. Let δ be a chase derivation of D w.r.t. Σ and $Q \subseteq \mathsf{B}(\mathbf{C} \cup \mathbf{N}, \mathbf{S})$ be a guarded and ungrounded set of atoms. The following are equivalent:*

1. $Q \subseteq \mathsf{chase}_\delta(D, \Sigma)$.
2. *There exists* exactly one *δ-pivotal atom $\alpha \in \mathsf{chase}_\delta(D, \Sigma)$ for Q such that $Q \subseteq \mathsf{chase}_{\delta[\alpha]}(\mathsf{type}_\delta(\alpha), \Sigma)$.*

Proof. Clearly, (2) $\Rightarrow$ (1) since $\mathsf{chase}_{\delta[\alpha]}(\mathsf{type}_\delta(\alpha), \Sigma) \subseteq \mathsf{chase}_\delta(D, \Sigma)$; $\delta[\alpha]$ is a chase derivation of $\mathsf{type}_\delta(\alpha)$ w.r.t. Σ due to Lemma 5.1. It remains to show (1) $\Rightarrow$ (2). Let $\gamma \in Q$ be the atom that contains all the terms of $\mathsf{dom}(Q)$ and $\alpha, \beta \in \mathsf{chase}_\delta(D, \Sigma)$ be atoms such that $\alpha \prec_\delta^{gp} \beta$, $\beta \prec_\delta^{gp,+} \gamma$, $\mathsf{null}(\alpha) \not\subseteq \mathsf{null}(\beta)$, and $\mathsf{null}(\beta) \subseteq \mathsf{null}(\gamma)$. By guardedness, we conclude that α is *the* δ-pivotal atom for Q, and, for each $\alpha' \in Q$, it holds that $\alpha \prec_\delta^{gp,+} \alpha'$ or $\alpha' \in \mathsf{type}_\delta(\alpha)$. Hence, $Q \subseteq \mathsf{chase}_{\delta[\alpha]}(\mathsf{type}_\delta(\alpha), \Sigma)$. □

We now have all the ingredients that are needed for introducing our new alternating algorithm, which will lead to the desired complexity result stated in Lemma 5.7.

Algorithm 1: Instance Checking.

Entail$(D, \Sigma, R(\bar{x}), \bar{c})$

if $R(\bar{c}) \in D$ **then**
  return Accept
else
  guess $\sigma \in \Sigma$ and $h : \mathsf{dom}(\mathsf{body}(\sigma)) \to \mathsf{dom}(D) \cup \Delta^{\star}_{\Sigma}$
  if $R(\bar{c}) \notin h(\mathsf{head}(\sigma))$ **then**
    return Reject
  else
    return Proof$(h(\mathsf{body}(\sigma)))$

Proof(Q)

$Q' := Q \setminus \left\{ R(\bar{c}) \in Q \mid \bar{c} \in \mathbf{C}^{\mathsf{ar}(R)} \right\}$
$N := \mathsf{null}(Q')$
universally do:
▷ **universally select** every atom $R(\bar{c}) \in Q \setminus Q'$
  return Entail$(D, \Sigma, R(\bar{x}), \bar{c})$
▷ **if** $N = \varnothing$ **then**
  return Accept
else
  guess (D, Σ)-step (σ, h, s, T) for $s_{\Sigma}(\sigma, h)$
  if $N \subseteq \mathsf{null}(h(\mathsf{guard}(\sigma)))$ or $N \not\subseteq \{s^1, \ldots, s^{\ell_\sigma}\}$ **then**
    return Reject
  else
    universally do:
    ▷ **universally select** every atom $\beta \in Q' \setminus T$
      return Reach(σ, h, s, T, β)
    ▷ **return** Proof(T)

Reach(σ, h, s, T, β)

guess an atom $\alpha \in [\![\sigma, h, s]\!]$
if $\alpha = \beta$ **then**
  return Accept
else
  $S := s_{\Sigma}(\sigma, h) \cup \{s\} \cup s_{\Sigma}(\mathsf{null}(\beta))$
  guess (D, Σ)-step (σ', h', s', T') for S
  if $h'(\mathsf{guard}(\sigma')) \neq \alpha$ **then**
    return Reject
  else
    universally do:
    ▷ **return** Reach$(\sigma', h', s', T', \beta)$
    ▷ **universally select** every atom $\beta' \in T' \setminus T$
      return Reach$(\sigma, h, s, T, \beta')$

The Alternating Algorithm

Consider a database D, a set Σ of guarded TGDs, a CQ $R(\bar{x})$, and a tuple $\bar{c} \in \mathsf{dom}(D)^{\mathsf{ar}(R)}$. We assume that D, Σ, and $R(\bar{x})$ are over a schema $\mathbf{S} \in \mathbb{S}$. Checking whether $\bar{c} \in \mathsf{cert}(R(\bar{x}), D, \Sigma)$ boils down to checking whether $R(\bar{c}) \in \mathsf{chase}_{\delta}(D, \Sigma)$ for some chase derivation δ of D w.r.t. Σ. Lemma 5.8 suggests the following strategy for the latter task: find $\sigma \in \Sigma$ and a function h from the variables in $\mathsf{body}(\sigma)$ to $\mathsf{dom}(D) \cup \mathbf{N}$ such that $R(\bar{c}) \in h(\mathsf{head}(\sigma))$, and check whether there exists a δ-pivotal atom for the ungrounded subset of $h(\mathsf{body}(\sigma))$, for some chase derivation δ of D w.r.t. Σ, while for each ground atom in $h(\mathsf{body}(\sigma))$ recursively apply the above. Our alternating algorithm, which is described in detail next, performs the above steps in parallel universal computations, which ensures that only logarithmic space is needed. Recall that polynomial time coincides with alternating logarithmic space [26].

Some Preparation. We first introduce some auxiliary notions. Since we can use only logarithmic space, we cannot explicitly store all the atoms generated via a single chase step since we deal with unguarded multi-heads; note that this could be possible in the case of single-head TGDs. Therefore, we need a way to compactly represent such a set of atoms, while such a representation should take only logarithmic space. This is done via a so-called (D, Σ)-step, which is a compact representation of the guard atom of a chase step, together with its type, and the atoms that are generated via this chase step.

Let $\Delta_{\Sigma} = \{\bot_1, \ldots, \bot_{2 \cdot (\mathsf{ar}(\Sigma)+1)}\}$ be the set that collects all the different *sorts* of nulls that are needed to perform our check using a bounded number of nulls, which in turn will ensure that we use only logarithmic space. Let $\Delta^{\star}_{\Sigma} = \{s^j \mid s \in$

$\Delta_\Sigma\}_{j\in[\ell_\Sigma]} \subseteq \mathbf{N}$, where $\ell_\Sigma = \max_{\sigma\in\Sigma}\{\ell_\sigma\}$ with ℓ_σ being the number of existential variables in σ. A (D,Σ)-*step* for a set $S \subsetneq \Delta_\Sigma$ is a tuple (σ, h, s, T), where

- $\sigma \in \Sigma$,
- $h : \mathsf{dom}(\mathsf{body}(\sigma)) \to \mathsf{dom}(D) \cup \Delta^\star_\Sigma$,
- $s \in \Delta_\Sigma \setminus S$, and
- $h(\mathsf{body}(\sigma)) \subseteq T \subseteq \mathsf{B}(\mathsf{dom}(h(\mathsf{guard}(\sigma))), \mathbf{S})$.

Such a (D,Σ)-step should be interpreted as follows. The triple (σ, h, s) encodes the set of atoms $h'(\mathsf{head}(\sigma))$, where h' extends h as follows: if $z_1, \ldots, z_k$ are the existential variables of σ, then $h'(z_i) = s^i$, i.e., the sort s and the variable z_i uniquely determine the null that is assigned to z_i. We refer to this set of atoms as $[\![\sigma, h, s]\!]$. Note that the fresh nulls of sort s in $[\![\sigma, h, s]\!]$ do not occur in $h(\mathsf{guard}(\sigma))$ since $h(\mathsf{guard}(\sigma))$ does not contain a null of sort s. The set T corresponds to the type of $h(\mathsf{guard}(\sigma))$. For a set $N \subseteq \Delta^\star_\Sigma$, it would be useful to be able to extract the set of sorts of nulls occurring in N, i.e., the set $s_\Sigma(N) = \{s \mid s^j \in N\} \subseteq \Delta_\Sigma$. For brevity, we simply write $s_\Sigma(\sigma, h)$ instead of the formal $s_\Sigma(\mathsf{null}(h(\mathsf{body}(\sigma))))$.

Description of the Algorithm. Our alternating algorithm is presented in Algorithm 1. Recall that alternation is an extension of non-determinism where, apart from existential steps, one can also perform universal steps where several independent computations can run in parallel. Indeed, Algorithm 1 is an alternating one since, not only existential guesses are performed, but also universal steps. The latter is expressed by either using the key word *universally do*, followed by a list of steps starting with the symbol ▷ that should be executed in parallel, or the key word *universally select*, which essentially means that, for each element of the set from which we select elements, we execute in parallel the command that follows. Here is a detailed description of our algorithm:

- The procedure Entail implements the strategy discussed above: find $\sigma \in \Sigma$ and a function h that maps $\mathsf{var}(\mathsf{body}(\sigma))$ to $\mathsf{dom}(D) \cup \Delta^\star_\Sigma$ with $R(\bar{c}) \in h(\mathsf{head}(\sigma))$, and check, via the procedure Proof, that $h(\mathsf{body}(\sigma))$ is derivable via some chase derivation. Note that we cannot afford to use an unlimited number of nulls since we have limited space. The nulls of $\Delta^\star_\Sigma$ are enough for performing this check.
- The procedure Proof checks whether a set of atoms Q is derivable via some chase derivation. Each ground atom of Q is proved in a parallel universal computation by recursively calling the procedure Entail. The remaining atoms form a guarded and ungrounded set Q'; if $\mathsf{null}(Q)$ is empty, which means that Q' is empty, then the algorithm accepts. In a parallel universal computation, it checks whether there is a δ-pivotal atom α for Q', for some chase derivation δ, such that Q' is derivable by applying chase steps starting from the δ-type of α (this is enough due to Lemma 5.8; notice that here we simply exploit the existence of a δ-pivotal atom). The guessed (D,Σ)-step (σ, h, s, T), actually $h(\mathsf{guard}(\sigma))$, corresponds to the δ-pivotal atom for Q', and the algorithm performs in parallel universal computations the following checks: (i) each atom β of Q', which is not already in the δ-type of $h(\mathsf{guard}(\sigma))$ given by T, is derivable by applying chase steps starting from T, which is done in parallel universal computations by calling the procedure Reach, and (ii) T is indeed the δ-type of $h(\mathsf{guard}(\sigma))$, which is done by recursively calling the procedure Proof.
- The procedure Reach actually checks whether, for some chase derivation δ of T w.r.t. Σ, $h(\mathsf{guard}(\sigma)) \prec_\delta^{gp,+} \beta$. It starts by guessing an atom α from $[\![\sigma, h, s]\!]$. If α is the atom that we are targeting, namely β, then it accepts; otherwise, it proceeds to check whether $\alpha \prec_\delta^{gp,+} \beta$. Due to Lemma 5.1, to check whether $\alpha \prec_\delta^{gp,+} \beta$, it suffices to consider only the δ-type of α. In fact, the guessed (D,Σ)-step (σ', h', s', T') provides the trigger to apply at the next step, that is, (σ', h') with $h'(\mathsf{guard}(\sigma')) = \alpha$, the sort of the fresh nulls that will appear in the generated atoms, that is, s', and the δ-type of α, that is, T'. It remains to verify that (i) $\alpha \prec_\delta^{gp,+} \beta$, and (ii) T' is indeed the δ-type of α. The former check is done by recursively calling the procedure Reach with input (σ', h', s', T'). For the latter check, one may suggest that the algorithm can simply call $\mathsf{Proof}(T')$. It should not be overlooked, though, that β and T' may share null values, and this fact should be preserved. However, by calling $\mathsf{Proof}(T')$ in a parallel universal computation, we loose this connection between β and T', which may lead to unsound results. The key observation is that $T' \setminus T$ is a guarded and ungrounded set of atoms that has the same δ-pivotal atom as $Q' \setminus T$ (the set of atoms from which β is coming from), which, by item (2) of Lemma 5.8, is unique; this atom is $h(\mathsf{guard}(\sigma))$. Hence, to prove T', the algorithm recursively calls for each atom $\beta' \in T' \setminus T$, in a parallel universal computation, $\mathsf{Reach}(\sigma, h, s, T, \beta')$.

By Lemmas 5.1 and 5.8, the algorithm Entail is correct, i.e., $\mathsf{Entail}(D, \Sigma, R(\bar{x}), \bar{c})$ accepts iff $R(\bar{c}) \in \mathsf{chase}_\delta(D, \Sigma)$, for some chase derivation δ of D w.r.t. Σ. Furthermore, at each step of its computation, the algorithm uses logarithmic space for storing elements of $\mathsf{dom}(D)$ and auxiliary pointers; recall that $\mathbf{S}$ is coming from the family of schemas $\mathbb{S}$ of fixed size. Since polynomial time coincides with alternating logarithmic space, Lemma 5.7 follows.

6. Conclusion

We considered the classes of linear and guarded existential rules, and investigated the limits of the polynomial combined approach to FO rewritability. We proved that it can be successfully applied to (i) linear existential rules when the rewritten query can use the full power of first-order queries, (ii) to linear existential rules when the arity of the underlying schema is fixed, and the rewritten query is a positive existential first-order query, and (iii) to guarded existential rules when the

underlying schema is fixed, and the rewritten query is a positive existential first-order query. As immediate corollaries, we get that ontological query answering for linear existential rules over schemas of fixed arity and guarded existential rules over fixed schemas is in NP. This closes a gap in the complexity picture of ontological query answering under linear and guarded existential rules, which passed unnoticed until recently when it was brought to our attention by a colleague of ours [24] and also discussed in [25].

The results of this work are, for the moment, of theoretical nature, and they simply tell us that the exploitation of conventional database systems for ontological query answering purposes in the case of linear and guarded existential rules is, in principle, possible. We do not claim that they will directly lead to efficient algorithms. A smart implementation and an evaluation of the obtained rewritings is the obvious next step.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

Gottlob is a Royal Society Research Professor and acknowledges support by the Royal Society in the context of the project "RAISON DATA" (project reference: RP/R1/201074). Part of Gottlob's work was done while visiting the University of Cyprus. Manna has been supported by MISE under the project "S2BDW" (F/050389/01-03/X32) - Horizon 2020 PON I&C2014-20, by Regione Calabria under the project "DLV LargeScale" (CUP J28C17000220006) - POR Calabria 2014-20, and by "PNRR MUR project PE0000013-FAIR, Spoke 9 - Green-aware AI – WP9.1". Pieris was supported by the EPSRC grant EP/S003800/1 "EQUID".

References

[1] G. Gottlob, M. Manna, A. Pieris, Polynomial combined rewritings for existential rules, in: KR (2014), 2014.
[2] G. Gottlob, M. Manna, A. Pieris, Polynomial rewritings for linear existential rules, in: IJCAI (2015), 2015, pp. 2992–2998.
[3] G. Gottlob, M. Manna, A. Pieris, Multi-head guarded existential rules over fixed signatures, in: KR (2020), 2020.
[4] S. Abiteboul, M. Arenas, P. Barceló, M. Bienvenu, D. Calvanese, C. David, R. Hull, E. Hüllermeier, B. Kimelfeld, L. Libkin, W. Martens, T. Milo, F. Murlak, F. Neven, M. Ortiz, T. Schwentick, J. Stoyanovich, J. Su, D. Suciu, V. Vianu, K. Yi, Research directions for principles of data management (abridged), SIGMOD Rec. 45 (2016) 5–17.
[5] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173.
[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: the DL-Lite family, J. Autom. Reason. 39 (2007) 385–429.
[7] A. Calì, G. Gottlob, T. Lukasiewicz, A general datalog-based framework for tractable query answering over ontologies, J. Web Semant. 14 (2012) 57–83.
[8] G. Gottlob, G. Orsi, A. Pieris, Query rewriting and optimization for ontological databases, ACM Trans. Database Syst. 39 (2014) 25:1–25:46.
[9] G. Gottlob, S. Kikot, R. Kontchakov, V.V. Podolskii, T. Schwentick, M. Zakharyaschev, The price of query rewriting in ontology-based data access, Artif. Intell. 213 (2014) 42–59.
[10] M. Bienvenu, S. Kikot, R. Kontchakov, V.V. Podolskii, M. Zakharyaschev, Ontology-mediated queries: combined complexity and succinctness of rewritings via circuit complexity, J. ACM 65 (2018) 28:1–28:51.
[11] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ envelope, in: IJCAI (2005), 2005, pp. 364–369.
[12] C. Lutz, D. Toman, F. Wolter, Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system, in: IJCAI (2009), 2009, pp. 2070–2075.
[13] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to query answering in DL-Lite, in: KR (2010), 2010.
[14] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to ontology-based data access, in: IJCAI (2011), 2011, pp. 2656–2661.
[15] C. Feier, D. Carral, G. Stefanoni, B. Cuenca Grau, I. Horrocks, The combined approach to query answering beyond the OWL 2 profiles, in: IJCAI (2015), 2015, pp. 2971–2977.
[16] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, A. Pieris, Datalog+/-: a family of logical knowledge representation and query languages for new applications, in: LICS (2010), 2010, pp. 228–242.
[17] A. Calì, G. Gottlob, M. Kifer, Taming the infinite chase: query answering under expressive relational constraints, J. Artif. Intell. Res. 48 (2013) 115–174.
[18] J.-F. Baget, M. Leclère, M.-L. Mugnier, E. Salvat, On rules with existential variables: walking the decidability line, Artif. Intell. 175 (2011) 1620–1654.
[19] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: semantics and query answering, Theor. Comput. Sci. 336 (2005) 89–124.
[20] T. Lukasiewicz, M.V. Martinez, A. Pieris, G.I. Simari, From classical to consistent query answering under existential rules, in: AAAI (2015), 2015, pp. 1546–1552.
[21] A. Calì, G. Gottlob, A. Pieris, Ontological query answering under expressive entity-relationship schemata, Inf. Syst. 37 (2012) 320–335.
[22] N. Leone, M. Manna, G. Terracina, P. Veltri, Fast query answering over existential rules, ACM Trans. Comput. Log. 20 (2019) 12:1–12:48.
[23] D.S. Johnson, A.C. Klug, Testing containment of conjunctive queries under functional and inclusion dependencies, J. Comput. Syst. Sci. 28 (1984) 167–189.
[24] M. Benedikt, Personal communication, 2018.
[25] K. Kappelmann, Decision procedures for guarded logics, CoRR, arXiv:1911.03679 [abs], 2019.
[26] C.H. Papadimitriou, Computational Complexity, Addison-Wesley, 1994.