

Rule-Based Ontology-Mediated Queries with Negation: An Algorithmic Characterization of First-order Rewritability

Andreas Pieris¹, Aldo Ricioppo²

¹University of Cyprus and University of Edinburgh

²University of Calabria

Abstract

The idea of using an ontology to enrich user queries with domain knowledge has attracted considerable attention from the database and KR communities during the last decade or so. The ontology and the user query can be conveniently seen as two components of one composite query, called ontology-mediated query (omq), while an omq language (OL, QL) collects all such omqs where the ontology is expressed using the ontology language OL and the user query comes from the query language QL. The evaluation problem for rule-based omq languages of the form (OL, CQ), where OL is a rule-based ontology language, i.e., it collects ontologies modelled using tuple-generating dependencies (a.k.a. existential rules), and CQ is the language of conjunctive queries, has been extensively studied in the literature. In particular, the notion of first-order rewritability of such languages, i.e., the property of being able to rewrite every omq from the language in question to an equivalent first-order query, has been studied in depth. Interestingly, every first-order rewritable tgds-based class of omqs, no matter how the underlying class of tgds is defined, can be handled via one uniform rewriting algorithm. More precisely, there is a resolution-based algorithm Rew such that, for every class C of tgds, the omq language (C, CQ) is first-order rewritable iff for every omq O from (C, CQ), Rew on input O terminates and returns a first-order rewriting of O. This provides an algorithmic characterization of when an omq language (C, CQ), where C is a class of tgds, is first-order rewritable. A question that has been recently posed is whether we can provide an analogous algorithmic characterization of when an omq language (C, CQ⁻), where C is a class of tgds and CQ⁻ is the language of conjunctive queries with the useful feature of negation, is first-order rewritable. We establish such a characterization for omq languages (C, CQ⁻), where C is a class of full tgds, that is, tgds without existentially quantified variables. An interesting consequence of our analysis is that first-order rewritability for positive and non-positive omq languages based on full tgds coincide. In other words, for every class C of full tgds, (C, CQ) is first-order rewritable iff (C, CQ⁻) is first-order rewritable. This allows us to systematically transfer existing first-order rewritability results for positive omq languages based on full tgds to non-positive ones.

Keywords

ontology-mediated queries, tuple-generating dependencies, conjunctive queries, negation, first-order rewritability

1. Introduction

Rule-based languages lie at the core of numerous areas of central importance to databases and artificial intelligence. It is widely accepted that tuple-generating dependencies (tgds), a.k.a. existential rules, form a prominent rule-based language, originally introduced as a unifying framework for database constraints [1]. Tgds are first-order sentences $\forall \bar{x} \forall \bar{y} (\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \psi(\bar{x}, \bar{z}))$, where ϕ (the body) and ψ (the head) are conjunctions of relational atoms, and they state that certain tuples in a database imply the presence of some other tuples in the database. The last decade or so, they have been employed for knowledge representation purposes, in particular, as an alternative way to model ontologies that are used to enrich user queries, typically conjunctive queries, with domain knowledge.

Ontology-Mediated Queries. The marriage of ontologies with user queries has been originally proposed in the context of ontology-based data integration, with the aim of facilitating access to data that is heterogeneous and incomplete [2]. The ontology and the user query can be conveniently seen as two components of one composite query, called ontology-mediated query (omq) [3]. In this context, the main problem of interest is to compute the answer to such an omq over a database. In particular,

Datalog 2.0 2026: 6th International Workshop on the Resurgence of Datalog in Academia and Industry

✉ apieris@inf.ed.ac.uk (A. Pieris); aldo.ricioppo@unical.it (A. Ricioppo)

ORCID 0000-0003-4779-3469 (A. Pieris); 0009-0007-5911-6015 (A. Ricioppo)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

given a database D and an omq $O = (\Sigma, Q)$, where Σ is a finite set of tgds (the ontology) and Q a conjunctive query (the user query), the goal is to compute the certain answers to O over D , denoted $\text{ans}(O, D)$, which are all the tuples of database elements that are answers to Q no matter on which model of D and Σ it is evaluated, that is, the answers to Q derived by every model of D and Σ .

Answering Omqs via Query Rewriting. Building ontology-aware database systems from scratch, with sophisticated optimization techniques, is a highly non-trivial task that requires a great effort. An alternative approach towards practical implementations of evaluating an omq over a database is to use conventional database systems. However, such systems are unaware of ontologies. This obstacle can be addressed by query rewriting: the omq $O = (\Sigma, Q)$ is rewritten into a new query Q_O , the so-called rewriting, which has the same answer as the omq O over all input databases, i.e., for every database D , $\text{ans}(O, D)$ coincides with the answer to Q_O over D . It is, of course, essential that the rewriting Q_O is expressed in a query language that can be handled by standard database systems. The typical target query language used in this setting is that of first-order queries.

The problem of rewriting omqs into first-order queries, when the ontology is a set of tgds and the user query a conjunctive query, has been extensively studied in the literature. This research effort led to several syntactic classes C of tgds that gave rise to first-order rewritable classes of omqs (C, CQ) , i.e., every omq (Σ, Q) of (C, CQ) , where Σ is a set of tgds from C and Q a query from the class of conjunctive queries CQ , admits a first-order rewriting, together with prototype systems illustrating the effectiveness of this approach; see, e.g., [4, 5, 6, 7]. Remarkably, every first-order rewritable tgd-based class of omqs, no matter how the underlying class of tgds is defined, can be handled via one uniform rewriting algorithm. More precisely, there is a resolution-based algorithm Rew such that, for every class C of tgds, the omq class (C, CQ) is first-order rewritable iff for every omq O from (C, CQ) , Rew on input O terminates and returns a first-order rewriting of O [7]. This provides an algorithmic characterization of when an omq class (C, CQ) , where C is a class of tgds, is first-order rewritable.

Adding Negation. Although conjunctive queries form a key tool for querying data, they are not able to express negation, and in particular, negation as failure, a very useful construct that allows us to express non-monotonic queries that naturally appear in real-life scenarios. It is thus very natural to consider classes of tgd-based omqs (C, CQ^-) , where the user query comes from the class of conjunctive queries with negation, denoted CQ^- . The problem of evaluating such non-positive omqs has also attracted attention in the literature (see, e.g., [8, 9, 10, 11]) and there are works that explicitly attacked the problem via first-order rewritability [4, 12].

An interesting question that arises in this context is whether an algorithmic characterization of when an omq class (C, CQ^-) , where C is a class of tgds, is first-order rewritable, in the spirit of the one for positive omqs from [7], can be established. In fact, this question has been recently posed in [12], where the authors made some first steps towards its settlement. It turned out from the analysis performed in [12] that establishing the desired algorithmic characterization is a highly non-trivial question. Our goal in this work is to provide such an algorithmic characterization for omq classes that are based on the central language of *full tgds*, that is, tgds without existentially quantified variables in their head.

Our Contribution. The outcome of our analysis can be summarized as follows:

1. We establish the desired algorithmic characterization of when an omq class (C, CQ^-) , where C is a class of full tgds, is first-order rewritable. This is done via a resolution-based algorithm that resembles the one for positive omqs, but with additional complications as it needs to carefully treat the negative part of the user query. Let us clarify that such complications have already been observed and handled in the rewriting algorithm proposed in [12].
2. A consequence of our technical analysis is that first-order rewritability for positive and non-positive omq classes based on full tgds coincide, i.e., for every class C of full tgds, (C, CQ) is first-order rewritable iff (C, CQ^-) is first-order rewritable. This allows us to transfer first-order rewritability results for positive omq classes to non-positive ones.

2. Preliminaries

Let $\mathbf{C}$ and $\mathbf{V}$ be disjoint countably infinite sets of *constants* and *variables*, respectively. We refer to constants and variables as *terms*. Given an integer $n > 0$, we write $[n]$ for the set of integers $\{1, \dots, n\}$.

Relational Databases. A *schema* $\mathbf{S}$ is a finite set of relation names (or predicates) with associated arity. We write R/n to say that R has arity $n > 0$; we may write $\text{ar}(R)$ for n . A (*relational*) *atom* α over $\mathbf{S}$ is an expression of the form $R(\bar{t})$, where $R/n \in \mathbf{S}$ and $\bar{t}$ is an n -tuple of terms; we may treat a tuple as the set of its elements. A *fact* is an atom that mentions only constants from $\mathbf{C}$. A (*relational*) *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$.

Homomorphisms. A *homomorphism* from a set of atoms A to a set of atoms B is a function h from the set of terms in A to the set of terms in B that is the identity on $\mathbf{C}$ with $R(h(\bar{t})) \in B$ for every $R(\bar{t}) \in A$. We write $A \rightarrow B$ to denote that there is a homomorphism from A to B .

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

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

where ϕ and ψ are (non-empty) conjunctions of atoms over $\mathbf{S}$ that mention only variables. For brevity, we write σ as $\phi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x})$ and use comma instead of $\wedge$ for joining atoms. We call ϕ and ψ the *body* and *head* of σ , denoted $\text{body}(\sigma)$ and $\text{head}(\sigma)$, respectively. Let FTGD be the family of all finite sets of full tgds; in the rest of the paper, a set of full tgds is always finite. A class $\mathbf{C}$ of full tgds is simply a subset of FTGD. A database D over $\mathbf{S}$ satisfies σ , written $D \models \sigma$, if, whenever $\phi(\bar{x}, \bar{y}) \rightarrow D$ via h , then $\psi(\bar{x}) \rightarrow D$ via h . The database D satisfies a set Σ of full tgds, written $D \models \Sigma$, if $D \models \sigma$ for each $\sigma \in \Sigma$. Given a database D over $\mathbf{S}$, a *model* of D and Σ is a (possibly infinite) set of facts $F \supseteq D$ such that $F \models \Sigma$; let $\text{mods}(D, \Sigma)$ be the set of models of D and Σ . A model M of D and Σ is *minimal* if there is no $M' \subsetneq M$ that is a model of D and Σ . It is well-known that a database D and a set Σ of full tgds have a *unique* minimal model, which we denote by $\text{mm}(D, \Sigma)$; in fact, $\text{mm}(D, \Sigma) = \bigcap_{F \in \text{mods}(D, \Sigma)} F$.

Conjunctive Queries with Negation. A *conjunctive query with negation* (ncq) Q over a schema $\mathbf{S}$ is an expression of the form

$$\text{Ans}(\bar{x}) :- R_1(\bar{y}_1), \dots, R_n(\bar{y}_n), \neg P_1(\bar{z}_1), \dots, \neg P_m(\bar{z}_m),$$

where $R_i(\bar{y}_i), P_j(\bar{z}_j)$ for $i \in [n], j \in [m]$, are atoms over $\mathbf{S}$, $\bar{x}$ are the *answer variables* of Q , each variable in $\bar{x}$ is mentioned in $\bar{y}_i$ for some $i \in [n]$, and, for each $j \in [m]$, each variable in $\bar{z}_j$ is mentioned in $\bar{y}_i$ for some $i \in [n]$; the latter condition means that the negation in Q is *safe*. We write $Q(\bar{x})$ to indicate that $\bar{x}$ are the answer variables of Q . When $\bar{x}$ is empty, Q is called *Boolean*. Let Q^+ be the set of atoms $\{R_1(\bar{y}_1), \dots, R_n(\bar{y}_n)\}$ and Q^- be the set of atoms $\{P_1(\bar{z}_1), \dots, P_m(\bar{z}_m)\}$. Note that if $Q^- = \emptyset$, then Q is simply a conjunctive query (cq). Let CQ^- be the set of all ncqs. We further write CQ for the set of all cqs. Given a database D , the *answer to Q over D* , denoted $Q(D)$, is the set of all tuples of constants $\bar{c}$ of length $|\bar{x}|$ such that (1) there is a homomorphism h from Q^+ to D with $h(\bar{x}) = \bar{c}$, and (2) for each $i \in [m]$, $P_i(h(\bar{z}_i)) \notin D$. For Boolean ncqs, we write $D \models Q$, and say that D *entails* Q , if $() \in Q(D)$; the empty tuple $()$ is the only possible answer to a Boolean ncq.

3. Ontology-Mediated Queries Based on Full Tgds

An *ontology-mediated query* (omq) over a schema $\mathbf{S}$ is a pair $O = (\Sigma, Q)$, where Σ is a set of full tgds over $\mathbf{S}$ (*the ontology*) and Q is an ncq over $\mathbf{S}$ (*the user query*). We say that O is Boolean if the query Q is Boolean. For a class $\mathbf{C} \subseteq \text{FTGD}$ of full tgds, we write $(\mathbf{C}, \text{CQ}^-)$ for the class of omqs (Σ, Q) , where $\Sigma \in \mathbf{C}$ and Q is an ncq. The class of omqs $(\mathbf{C}, \text{CQ})$ is defined analogously. Given a database D and an omq $O = (\Sigma, Q)$, the *answer to O over D* is defined as

$$\text{ans}(O, D) = Q(\text{mm}(D, \Sigma)),$$

i.e., we simply evaluate Q over the unique minimal model of D and Σ . The question that comes up is whether there is convenient way for computing the database $\text{mm}(D, \Sigma)$. To this end, we exploit the chase procedure, a useful algorithmic tool when reasoning with tuple-generating dependencies.

The Chase Procedure for Full Tgds. We first define a single chase application. A *trigger* for a set Σ of full tgds on a database D is a pair (σ, h) , where $\sigma \in \Sigma$ and h is a homomorphism from $\text{body}(\sigma)$ to D . An *application* of (σ, h) to D returns the database $D' = D \cup h(\text{head}(\sigma))$. The main idea of the chase is, starting from a database D , to apply distinct triggers for the set Σ of full tgds until a fixpoint is reached.

Definition 1 (Chase Derivation). *Consider a database D and set Σ of full tgds. A chase derivation of D w.r.t. Σ is a sequence of databases $\delta = (D_i)_{0 \leq i \leq n}$, with $D = D_0$ and $n \geq 0$, such that:*

1. *for each $0 \leq i < n$, there is a trigger (σ_i, h_i) for Σ on D_i such that $D_i \langle \sigma_i, h_i \rangle D_{i+1}$,*
2. *for each $0 \leq i < j < n$, if $\sigma_i = \sigma_j$, then $h_i \neq h_j$, and*
3. *there is no trigger (σ, h) for Σ on D_n such that $(\sigma, h) \notin \{(\sigma_i, h_i)\}_{0 \leq i < n}$.*

The result of δ , denoted $\text{chase}_\delta(D, \Sigma)$, is the database D_n . ■

It is known that, for every two chase derivations δ_1 and δ_2 of D w.r.t. Σ , it holds that $\text{chase}_{\delta_1}(D, \Sigma) = \text{chase}_{\delta_2}(D, \Sigma)$. Thus, we can refer to *the* result of the chase of D w.r.t. Σ , denoted $\text{chase}(D, \Sigma)$. Interestingly, $\text{chase}(D, \Sigma)$ is precisely the database $\text{mm}(D, \Sigma)$, and thus, we get the following result:

Proposition 1. *For a database D and an omq (Σ, Q) , it holds that $\text{ans}(D, O) = Q(\text{chase}(D, \Sigma))$.*

3.1. First-Order Rewritability

Our objective is to study the central notion of first-order rewritability for omqs, which essentially states that we can compute the answer to an omq by relying on first-order query evaluation. This in turn implies that we can exploit standard database systems for evaluating omqs. Recall that a *first-order query* Q over a schema $\mathbf{S}$ is an expression of the form $\{(x_1, \dots, x_n) \mid \phi\}$, where ϕ is a (finite) first-order formula that uses predicates from $\mathbf{S}$ and mentions only constants and variables, $(x_1, \dots, x_n)$ is a tuple over the free variables of ϕ , and each free variable of ϕ occurs at least once in $(x_1, \dots, x_n)$. A first-order query of the form $\{() \mid \phi\}$, where ϕ is a sentence without free variables, is called *Boolean*. Given a database D over $\mathbf{S}$, the *answer to Q over D* , denoted $Q(D)$, is the set of all tuples of constants $(c_1, \dots, c_n)$ such that D is a model under the active domain semantics (i.e., the underlying domain consists of all the constants occurring in the atoms of D) of the sentence obtained from ϕ after replacing the variable x_i with the constant c_i for each $i \in [n]$. We typically refer to a Boolean first-order query $Q = \{() \mid \phi\}$ by simply using the sentence ϕ , and we write $D \models Q$ or $D \models \phi$ for the fact that $() \in Q(D)$. We are now ready to give the formal definition of first-order rewritability for omqs.

Definition 2 (First-Order Rewritability). *Consider an omq $O = (\Sigma, Q)$ over a schema $\mathbf{S}$. A first-order rewriting of O is a first-order query Q_O over $\mathbf{S}$ such that, for every database D over $\mathbf{S}$, $\text{ans}(O, D) = Q_O(D)$. A class $(C, CQ^\neg)$ of omqs, where $C \subseteq \text{FTGD}$, is first-order rewritable if every omq from $(C, CQ^\neg)$ has a first-order rewriting. A first-order rewriter for $(C, CQ^\neg)$ is a deterministic algorithm that takes as input a query O from $(C, CQ^\neg)$ and constructs a first-order rewriting of O .* ■

The main result of this work is an algorithmic characterization of when a class of omqs $(C, CQ^\neg)$, where $C \subseteq \text{FTGD}$, is first-order rewritable. In particular, we establish the following:

Theorem 1. *There exists a deterministic algorithm Rew such that, for every class $C \subseteq \text{FTGD}$ of full tgds, the following statements are equivalent:*

1. *$(C, CQ^\neg)$ is first-order rewritable.*
2. *Rew is a first-order rewriter for $(C, CQ^\neg)$.*

Let us stress that the above result, not only algorithmically characterizes when a class of omqs $(C, CQ^\neg)$, where C is a class of full tgds, is first-order rewritable, but it further states that this is done via one uniform algorithm that does not depend on the class of tgds. The direction (2) implies (1) of Theorem 1 holds trivially. Now, to establish the interesting direction (1) implies (2), let us assume for the moment that we have a deterministic algorithm Rew with the following property (the notion of first-order rewritability for a class of omqs (C, CQ) is defined in the same way as for $(C, CQ^\neg)$):

- ($\star$) For a class C of full tgds such that (C, CQ) is first-order rewritable the following holds: for every omq O from $(C, CQ^\neg)$ over a schema S , $\text{Rew}(O)$ terminates and returns a first-order query Q_O over S that is a first-order rewriting of O , i.e., for every database D over S , $\text{ans}(O, D) = Q_O(D)$.

With such an algorithm in place, it is then easy to establish the direction (1) implies (2). Indeed,

$$\begin{array}{ll}
(C, CQ^\neg) \text{ is first-order rewritable} & \\
CQ \subseteq CQ^\neg & (C, CQ) \text{ is first-order rewritable} \\
\Rightarrow & \\
\text{by } (\star) & \text{For every } O \in (C, CQ^\neg), \text{Rew}(O) \text{ terminates and returns a first-order rewriting of } O \\
\Rightarrow & \\
\text{by definition} & \text{Rew is a first order-rewriter for } (C, CQ^\neg).
\end{array}$$

Consequently, to establish the direction (1) implies (2) of Theorem 1, it suffices to devise an algorithm that enjoys property ($\star$). Section 4 is devoted to this interesting task. But before we proceed further with this task, let us conclude this section by observing that an algorithm Rew with the property ($\star$) has another interesting consequence, that is, first-order rewritability for classes of positive omqs and first-order rewritability for classes of non-positive omqs are equivalent notions. This in turn allows us to systematically transfer first-order rewritability results for classes of positive omqs to classes of non-positive ones. The formal result follows:

Theorem 2. *For every class $C \subseteq \text{FTGD}$ of full tgds, the following statements are equivalent:*

1. (C, CQ) is first-order rewritable.
2. $(C, CQ^\neg)$ is first-order rewritable.

Since CQ is contained in $CQ^\neg$, the direction (2) implies (1) holds trivially. Conversely, (1) implies (2) is a consequence of the fact that, as discussed above, the first-order rewritability of (C, CQ) implies that Rew is a first-order rewriter for $(C, CQ^\neg)$, which means that $(C, CQ^\neg)$ is first-order rewritable.

4. A Uniform First-Order Rewriter

We now establish the existence of a deterministic algorithm with the property ($\star$), which shows (1) implies (2) in Theorem 1, as well as Theorem 2. To this end, we are going to show the following:

Proposition 2. *There exists a non-deterministic algorithm Rew such that, for every class $C \subseteq \text{FTGD}$ of full tgds with (C, CQ) being first-order rewritable the following holds: for every omq O from $(C, CQ^\neg)$, every execution of $\text{Rew}(O)$ terminates and returns a first-order rewriting of O .*

Since the property stated in Proposition 2 holds for *every execution* of the non-deterministic algorithm Rew , we then obtain the existence of a deterministic algorithm with the property ($\star$): simply fix a certain order of the non-deterministic choices of Rew , which will lead to the desired deterministic algorithm. The rest of the section is devoted to discussing the proof of Proposition 2. We first present a resolution-based non-deterministic algorithm, called QueryRewrite , and then show that it enjoys the property stated in Proposition 2.

4.1. The Non-Deterministic Algorithm QueryRewrite

As for positive omqs, the high-level idea of the rewriting algorithm is to start from the positive part of the given ncq and simulate backwards the chase steps that lead to its image in the result of the chase via resolution steps that use the full tgds as rewriting rules until we reach the input database; the latter is implemented using the standard notion of unifier among sets of atoms. This rewriting process should lead to an equivalent first-order query that can be directly evaluated over the input database. But now, unlike the positive case, we need to carefully treat the negative part of the query.

We start by recalling the standard notion of unifier among sets of atoms, which is the building block of our algorithm, and then discuss the algorithm itself, which is essentially an iterative application of resolution steps. Without loss of generality, we assume that full tgds have only one atom in their head. Indeed, a set Σ of full tgds can be equivalently rewritten as a set of full tgds with only one atom in their head by simply replacing each full tgd $\phi(\bar{x}, \bar{y}) \rightarrow R_1(\bar{x}_1), \dots, R_n(\bar{x}_n)$ of Σ with the full tgds $\phi(\bar{x}, \bar{y}) \rightarrow R_i(\bar{x}_i)$ for $i \in [n]$. Moreover, for clarity, we focus on Boolean ncqs, but everything can be extended to non-Boolean ncqs at the expense of slightly more complex definitions and proofs. In what follows in this section, we silently assume, without explicitly saying it, that ncqs are Boolean. Finally, by abuse of notation, we may treat a conjunction of atoms as the set of its atoms; it will be clear from the context when a conjunction of atoms is treated as a set of atoms.

Unifiers and Resolvents. We say that two sets of atoms A and B *unify* if there is a substitution θ , which is the identity on $\mathbf{C}$, called *unifier for A and B* , such that $\theta(A) = \theta(B)$. A *most general unifier* (mgu) for A and B is a unifier $\theta_{A,B}$ for A and B such that, for each unifier θ for A and B , $\theta = \theta' \circ \theta_{A,B}$ for some substitution θ' . It is well-known that if two sets of atoms unify, then there is a unique mgu (modulo variable renaming). We can further assume that the mgu θ for A and B is such that $\theta(A) = \theta(B)$ mentions only variables occurring in A and B ; in what follows, we always use mgus with the latter property.

Consider now a set of atoms A and a full tgd σ that does not share variables with A . A *(most general) unifier of A with σ* is a pair $\tau = (A', \theta)$, where $\emptyset \subsetneq A' \subseteq A$ and θ is a (most general) unifier of A' with $\{\text{head}(\sigma)\}$. An mgu $\tau = (A', \theta)$ of A with σ essentially indicates that the subset A' of A can be satisfied in the result of the chase by applying the tgd σ . Thus, we can simulate this chase application backwards by resolving A' using σ , which is done by simply replacing A' in A with $\text{body}(\sigma)$ and properly updating the variables according to θ . Such a resolution step is formalized via the notion of (σ, τ) -resolvent. In particular, the (σ, τ) -resolvent of A , denoted $\text{res}(A, \sigma, \tau)$, is defined as the set of atoms

$$\{\theta(\beta) \mid \beta \in (A \setminus A') \cup \text{body}(\sigma)\}.$$

Sentences in Rewrite Form. First-order sentences of the following form, which we call *rewrite form*, will play a crucial role in our development:

$$\exists \bar{x} \left(\Phi_1(\bar{x}) \wedge \Phi_2(\bar{y}) \wedge \bigwedge_i \neg (\exists \bar{z}_i (\Phi_3^i(\bar{z}_i, \bar{w}_i))) \right),$$

where

- $\Phi_1(\bar{x})$ is a non-empty conjunction of atoms with constants of $\mathbf{C}$ and variables of $\bar{x}$,
- $\Phi_2(\bar{y})$ is a possibly empty conjunction of inequalities over $\mathbf{C}$ and $\bar{y} \subseteq \bar{x}$, and
- $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ is a possibly empty disjunction of conjunctions of atoms with constants of $\mathbf{C}$ and variables of $\bar{z}_i \cup \bar{w}_i$ with $\bar{x} \cap \bar{z}_i = \emptyset$ and $\bar{w}_i \subseteq \bar{x}$.

Consider a first-order sentence Φ in rewrite form as above and a full tgd σ . A *(most general) unifier of Φ with σ* is a pair $\tau = (A, \theta)$ that is a (most general) unifier of $\Phi_1(\bar{x})$ with σ . As discussed above, an mgu $\tau = (A, \theta)$ of Φ with σ essentially indicates that the set of atoms A of $\Phi_1(\bar{x})$ can be satisfied in

```

if  $Q^- = \emptyset$  then
  |  $Rew := \left\{ \exists \bar{x} \left( \bigwedge_{\alpha \in Q} \alpha \right) \right\}$ 
else
  |  $\Pi := \{ \pi \mid \pi \text{ is a proper partition of terms}(Q^-) \}$ 
  |  $Rew := \{ \Phi_{Q,\pi} \mid \pi \in \Pi \text{ and } \Phi_{Q,\pi} \text{ is satisfiable} \}$ 
  | if  $Rew = \emptyset$  then
  | | return  $\exists x_1 \cdots \exists x_n (P(x_1, \dots, x_n) \wedge \neg P(x_1, \dots, x_n))$ 
  |
  |  $i := 0$ 
  | repeat
  | |  $Temp := Rew$ 
  | | forall  $\Phi \in Rew$  do
  | | | forall  $\sigma \in \Sigma$  do
  | | | |  $i := i + 1$ 
  | | | | forall mgu  $\tau$  of  $\Phi$  with  $\sigma^i$  do
  | | | | |  $\Phi' := \text{res}(\Phi, \sigma^i, \tau)$ 
  | | | | | if  $\Phi'$  is not  $Rew$ -blocked then
  | | | | | |  $Rew := Rew \cup \{ \Phi' \}$ 
  | |
  | until  $Temp = Rew$ ;
return  $\bigvee_{\Phi \in Rew} \Phi$ 

```

Algorithm 1: QueryRewrite(Σ, Q)

the result of the chase by applying the tgd σ , and thus, we can simulate this chase application via a resolution step. Formally, the (σ, τ) -*resolvent* of Φ , denoted $\text{res}(\Phi, \sigma, \tau)$, is defined as the sentence

$$\exists \bar{u} \left(\bigwedge_{\beta \in \text{res}(\Phi_1(\bar{x}), \sigma, \tau)} \beta \wedge \Phi_2(\theta(\bar{y})) \wedge \bigwedge_i \neg (\exists \bar{z}_i (\Phi_3^i(\bar{z}_i, \theta(\bar{w}_i)))) \right),$$

where $\bar{u}$ collects all the variables in the formula apart from those in Ψ_3^i that are existentially quantified. It is important to stress that $\text{res}(\Phi, \sigma, \tau)$ is a sentence in rewrite form. More precisely, a resolution step does not lead to an unsafe sentence, i.e., a sentence where the negative part (including the inequalities) has a free variable that does not appear in the positive part. This is ensured by the definition of full tgds, and in particular, by the fact that every variable in $\text{head}(\sigma)$ occurs also in $\text{body}(\sigma)$.

The Algorithm QueryRewrite. We now have all the notions needed to discuss our query rewriting algorithm, depicted in Algorithm 1, that takes as input an omq (Σ, Q) , where Σ is a set of full tgds and Q a Boolean ncq, and exhaustively applies resolution steps. It first checks whether Q has any negative atoms. If this is not the case, then it builds the first-order sentence in rewrite form $\exists \bar{x} \left(\bigwedge_{\alpha \in Q} \alpha \right)$, where $\bar{x}$ are the variables occurring in Q , that is kept in the set Rew . Otherwise, it proceeds to build several first-order sentences in rewrite form, which will be kept in Rew , by rewriting the negative atoms using the procedure AtomRewrite, depicted in Algorithm 2. Let us discuss the details of this step.

Let $\preceq_{\text{lex}}$ be some lexicographic order over $\mathbf{C} \cup \mathbf{V}$ such that all the constants precede the variables. For a finite set T of constants and variables, we write $[T]$ for the term of T that precedes all other terms of T according to $\preceq_{\text{lex}}$. A partition $\pi = \{T_1, \dots, T_m\}$ of the set $\text{terms}(Q^-)$ of terms occurring in Q^- is called *proper* if, for each $i \in [m]$, T_i contains at most one constant, i.e., two different constants occurring in Q^- appear in different sets of π . Essentially, a proper partition π of $\text{terms}(Q^-)$ fixes the way that the variables in Q^- should be mapped to the constants of the input database, i.e., two variables should be mapped to the same constant iff they occur in the same set of π . Furthermore, if a variable x occurs in a set of π that has a constant c , then x should be mapped to c . To implement this intuitive idea, we define the function μ_π from the set $\text{terms}(Q)$ of terms in Q to itself that is the identity on

```

 $U := \{t_1, \dots, t_n\} \cap \mathbf{V}$ 
 $Rew := \{\{R(t_1, \dots, t_n)\}\}$ 
 $i := 0$ 
repeat
   $Temp := Rew$ 
  forall  $A \in Rew$  do
    forall  $\sigma \in \Sigma$  do
       $i := i - 1$ 
      forall mgu  $\tau = (\cdot, \theta)$  of  $A$  with  $\sigma^i$  such that  $\theta$  is the identity on  $U$  do
         $A' := \text{res}(A, \sigma^i, \tau)$ 
        if  $A'$  is not  $(Rew, U)$ -blocked then
           $Rew := Rew \cup \{A'\}$ 
until  $Temp = Rew$ ;
return  $\bigvee_{A \in Rew} \bigwedge_{\alpha \in A} \alpha$ 

```

Algorithm 2: AtomRewrite($\Sigma, R(t_1, \dots, t_n)$)

terms(Q) $\setminus$ terms(Q^-), and for each $t \in \text{terms}(Q^-)$, $t \in T_i$ implies $\mu_\pi(t) = [T_i]$.

For a proper partition $\pi = \{T_1, \dots, T_m\}$ of terms(Q^-), the algorithm builds the sentence $\Phi_{Q,\pi}$

$$\exists \bar{x}_\pi \left(\bigwedge_{\alpha \in Q^+} \mu_\pi(\alpha) \wedge \bigwedge_{1 \leq i < j \leq m} [T_i] \neq [T_j] \wedge \bigwedge_{\alpha \in Q^-} \neg (\exists \bar{z}_\alpha (\text{AtomRewrite}(\Sigma, \mu_\pi(\alpha)))) \right),$$

where $\bar{x}_\pi$ are the variables occurring in Q^+ but not in Q^- plus the variables of $\{[T_1], \dots, [T_m]\} \cap \mathbf{V}$, and $\bar{z}_\alpha$ are the variables occurring in $\text{AtomRewrite}(\Sigma, \mu_\pi(\alpha))$ but not in $\bar{x}_\pi$. Let us say that $\text{AtomRewrite}(\Sigma, \mu_\pi(\alpha))$ exhaustively rewrites $\{\mu_\pi(\alpha)\}$ by considering each full tgd $\sigma \in \Sigma$ and mgu $(\cdot, \theta)$ of $\{\mu_\pi(\alpha)\}$ with σ , where θ is the identity on the variables in $\mu_\pi(\alpha)$ since, as said above, the partition π states that the variables $[T_i]$ and $[T_j]$ for $i \neq j$ cannot be unified. Observe though that during the resolution step we actually consider the full tgd σ^i obtained from σ by renaming each variable x in σ to x^i . This is a mechanism that allows us to avoid undesirable clash among variables during a resolution step. At each resolution step, the produced set of atoms A' is added to Rew only if there is no set of atoms $B \in Rew$ such that $B \rightarrow A$ via a homomorphism that is the identity on U , where U is the set of variables in $\mu_\pi(\alpha)$, in which case we say that A' is not (Rew, U) -blocked. In essence, A' is added to Rew if it is general enough, namely there is no $B \in Rew$ that is more general than A since in this case A will be redundant. The sentence $\Phi_{Q,\pi}$ is kept in the set Rew if it is satisfiable, i.e., if there exists a database D such that $D \models \Phi_{Q,\pi}$. If none of those sentences is satisfiable, i.e., $Rew = \emptyset$, then there is no database D such that $() \in \text{ans}(O, D)$, and thus, the algorithm returns $\exists x_1 \dots \exists x_n (P(x_1, \dots, x_n) \wedge \neg P(x_1, \dots, x_n))$ for some predicate P/n of the underlying schema, that is, a simple unsatisfiable sentence over the underlying schema.

Once Rew is computed (and is non-empty), the algorithm proceeds to exhaustively rewrite the positive part of each first-order sentence of Rew via resolution steps, which will lead to the final rewriting as a disjunction of sentences in rewrite form. In particular, for each sentence $\Phi \in Rew$ and full tgd σ , the algorithm considers all the mgus of Φ with σ^i , and then adds to Rew the (σ^i, τ) -resolvent of Φ providing that it is not Rew -blocked. Before formalizing the notion of Rew -blocking, let us note that here, as in AtomRewrite, we consider the full tgd σ^i obtained from σ by renaming each variable x in σ to x^i in order to avoid undesirable clash among variables during a resolution step. The fact that there is no clash among variables introduced via a resolution step on the positive part and variables introduced during the execution of AtomRewrite, it is ensured by the fact that here we use positive indices to rename variables, whereas in AtomRewrite we use negative indices.

Coming back to the notion of Rew -blocking, we add in Rew a sentence Φ produced by a resolution step only if it is general enough, i.e., there is no sentence in Rew that is more general than Φ since in

this case Φ is redundant. Formally, we say that a sentence

$$\Phi = \exists \bar{x} \left(\Phi_1(\bar{x}) \wedge \Phi_2(\bar{y}) \wedge \bigwedge_{i \in [n]} \neg (\exists \bar{z}_i (\Phi_3^i(\bar{z}_i, \bar{w}_i))) \right)$$

is *Rew-blocked* if there exists a sentence

$$\Psi = \exists \bar{x}' \left(\Psi_1(\bar{x}') \wedge \Psi_2(\bar{y}') \wedge \bigwedge_{i \in [m]} \neg (\exists \bar{z}'_i (\Psi_3^i(\bar{z}'_i, \bar{w}'_i))) \right)$$

in *Rew* such that:

- $\Psi_1(\bar{x}') \rightarrow \Phi_1(\bar{x})$ via h such that $h(x) \neq h(y)$ for each inequality $x \neq y$ in $\Psi_2(\bar{y}')$,
- for each inequality $x \neq y$ in $\Psi_2(\bar{y}')$, the inequality $h(x) \neq h(y)$ occurs in $\Phi_2(\bar{y})$, and
- for each $i \in [m]$ and disjunct $\alpha_1 \wedge \dots \wedge \alpha_k$ of $\Psi_3^i(\bar{z}'_i, \bar{w}'_i)$, there exists $j \in [n]$ such that $h(\alpha_1) \wedge \dots \wedge h(\alpha_k)$ is a disjunct of $\Phi_3^j(\bar{z}_j, \bar{w}_j)$.

This completes the description of the algorithm *QueryRewrite*, which is indeed non-deterministic since there is no fixed order in which we apply the resolution steps starting from the input ncq.

4.2. Termination of QueryRewrite

We proceed to establish the termination of every execution of *QueryRewrite* whenever the input is an omq from (C, CQ^-) , where C is an arbitrary class of full tgds, assuming that (C, CQ) is first-order rewritable. To this end, we first establish that the first-order rewritability of (C, CQ) implies the termination of every execution of *QueryRewrite* on every input from (C, CQ) .

Lemma 1. *Consider a class $C \subseteq \text{FTGD}$ of full tgds. If (C, CQ) is first-order rewritable, then, for every omq O from (C, CQ) , every execution of *QueryRewrite*(O) terminates.*

The proof of Lemma 1 relies on the celebrated Rossman's Theorem [13], which states that a first-order sentence is preserved under homomorphisms on all databases iff it is equivalent to an existential-positive first-order sentence, and the fact that omqs from (FTGD, CQ) are preserved under homomorphisms on all databases. Having the above lemma in place, we can establish the desired termination result:

Lemma 2. *Consider a class $C \subseteq \text{FTGD}$ of full tgds. If (C, CQ) is first-order rewritable, then, for every omq O from (C, CQ^-) , every execution of *QueryRewrite*(O) terminates.*

The above lemma is shown by proving the contrapositive. Assume that there exists an omq $O = (\Sigma, Q)$ from (C, CQ^-) such that there exists an execution of *QueryRewrite*(O) that does not terminate. By exploiting this fact, we would like to show that there exists a cq (without negation) $\hat{Q}$ such that there is an execution of *QueryRewrite*($\hat{O}$), where $\hat{O} = (\Sigma, \hat{Q})$, that does not terminate. Since $\hat{O}$ belongs to (C, CQ) , by Lemma 1 we get that (C, CQ) is not first-order rewritable, and the claim follows. Observe that *QueryRewrite*(O) does not terminate for one of the following two reasons:

1. The construction of the set *Rew*, which relies on the algorithm *AtomRewrite*, does not terminate.
2. *Rew* is finite and there is $\Phi \in \text{Rew}$ that gives rise to an infinite sequence of resolution steps.

In the former case, it is easy to argue that there exists an atomic cq $R(c_1, \dots, c_n)$ that mentions only constants such that $\text{QueryRewrite}(\Sigma, R(c_1, \dots, c_n))$ does not terminate, i.e., our claim holds with $\hat{Q} = R(c_1, \dots, c_n)$. Let us now discuss the latter case. Clearly, the infinite execution of $\text{QueryRewrite}(O)$ produces an infinite sequence of sentences $s = (\Psi_i)_{i \geq 0}$, where $\Psi_0 = \Phi$ and, for each $i > 0$, Ψ_i is obtained from Ψ_{i-1} via a resolution step that uses an mgu. We can easily argue that there exists an integer $\ell \geq 0$ such that all the sentences $\Psi_\ell, \Psi_{\ell+1}, \dots$ have the exact same negative part (i.e., all those sentences, apart from their positive part, are syntactically the same). This implies that the sequence of cqs $\hat{s} = (\hat{Q}_i)_{i \geq \ell}$, where $\hat{Q}_i$ is obtained from the positive part of Ψ_i after replacing each variable x that occurs in the positive and the negative part of Ψ_i with a new constant c_x , is such that $\hat{Q}_{i+1}$ is obtained from $\hat{Q}_i$ via a resolution step that uses the mgu used to generate Ψ_{i+1} from Ψ_i . It is not difficult to argue that there is an execution of $\text{QueryRewrite}(\Sigma, \hat{Q}_\ell)$ that produces the infinite sequence $\hat{s}$ of cqs, and thus, our claim follows with $\hat{Q}$ being the cq $\hat{Q}_\ell$.

4.3. Correctness of QueryRewrite

Having Lemma 2 in place, to conclude the proof of Proposition 2, it remains to establish the correctness of our algorithm. In other words, we need to establish the following result:

Lemma 3. *Consider a class $C \subseteq \text{FTGD}$ of full tgds such that (C, CQ) is first-order rewritable. For every O from $(C, \text{CQ}^\neg)$, every execution of $\text{QueryRewrite}(O)$ returns a first-order rewriting of O .*

The rest of the section is devoted to discussing the proof of the above lemma. Let C be a class of full tgds such that (C, CQ) is first-order rewritable. Consider an arbitrary omq $O = (\Sigma, Q)$ from $(C, \text{CQ}^\neg)$ and some arbitrary execution of $\text{QueryRewrite}(O)$ that leads to a first-order sentence Q_O . By Lemma 2, Q_O is finite, and thus, well-defined; in fact, Q_O is a finite disjunction of finite first-order sentences in rewrite form. We now need to argue that Q_O is a first-order rewriting of O . Fix an arbitrary database D . By Proposition 1, it suffices to show that the following statements are equivalent:

1. $\text{chase}(D, \Sigma) \models Q$.
2. There exists a disjunct Φ of Q_O such that $D \models \Phi$.

We will refer to the above equivalence as $(\diamond)$. To establish this equivalence, we first need to present some auxiliary lemmas. In what follows, we write $\text{Rew}[\Sigma, Q]$ for the set of sentences Rew computed by the if-then-else statement of $\text{QueryRewrite}(O)$ (i.e., before executing the repeat-until loop). Moreover, a *rewriting Σ -sequence* is a sequence of sentences $(\Phi_i)_{i \in \{0, \dots, m\}}$, for $m \geq 0$, in rewrite form such that Φ_{i+1} is a (σ, τ) -resolvent of Φ_i for some mgu τ of Φ_i with σ , for each integer $i \in \{0, \dots, m-1\}$.

Auxiliary Lemmas. The first auxiliary lemma shows that the rewriting of the negative atoms, performed by the procedure AtomRewrite , is sound.

Lemma 4. *The following statements are equivalent:*

1. $\text{chase}(D, \Sigma) \models Q$.
2. $\text{chase}(D, \Sigma) \models \Phi$, for some sentence $\Phi \in \text{Rew}[\Sigma, Q]$.

The next lemma is crucial for establishing the completeness of QueryRewrite , i.e., the direction (1) implies (2) of $(\diamond)$. It essentially states that from a prefix s of a chase derivation of D w.r.t. Σ , where its last database entails a sentence Φ in rewrite form, we can extract a sequence of resolution steps starting from Φ , and simulating backwards the chase applications that gave rise to s , that leads to a sentence that is entailed by D . This is shown by induction on the length of the prefix s .

Lemma 5. *Consider a sentence Φ in rewrite form. Let $(D_i)_{i \in \{0, \dots, m\}}$, for $m \geq 0$, be a prefix of a chase derivation of D w.r.t. Σ with $D_i \langle \sigma_i, h_i \rangle D_{i+1}$, for $i \in \{0, \dots, m-1\}$, such that $D_m \models \Phi$. Then, there exists a sequence $(\Psi_i)_{i \in \{0, \dots, m\}}$ of sentences in rewrite form such that:*

1. $\Psi_0 = \Phi$
2. $\Psi_i = \Psi_{i-1}$ or $\Psi_i = \text{res}(\Psi_{i-1}, \sigma_{m-i}, \tau)$ for some mgu τ of Ψ_{i-1} with σ_{m-i} , for $i \in [m]$, and
3. $D_{m-i} \models \Psi_i$, for each $i \in \{0, \dots, m\}$.

The last lemma is crucial for the direction (2) implies (1) of $(\diamond)$, i.e., the soundness of QueryRewrite. It essentially states that given a rewriting Σ -sequence s , if any of its intermediate sentences is entailed by D , then its first sentence is entailed by a database that is obtained by some chase quasi-derivation of D w.r.t. Σ . The notion of chase quasi-derivation is defined in the same way as the notion of chase derivation (see Definition 1) with the crucial difference that triggers may repeat. In other words, a sequence of databases $\kappa = (D_i)_{0 \leq i \leq n}$, with $D = D_0$ and $n \geq 0$, is a *chase quasi-derivation* of D w.r.t. Σ if the conditions (1) and (3) of Definition 1 are satisfied. The *result* of κ , denoted $\text{chase}_\kappa(D, \Sigma)$, is the database D_n . It is easy to see that, for every chase quasi-derivation κ of D w.r.t. Σ , it holds that $\text{chase}_\kappa(D, \Sigma) = \text{chase}(D, \Sigma)$. The reason why we need to consider chase quasi-derivations, instead of chase derivations, in our last lemma is because two rewriting steps in the rewriting Σ -sequence s may correspond to the same chase step. Our technical lemma is shown by induction on the length of s .

Lemma 6. *Consider a rewriting Σ -sequence $(\Psi_i)_{i \in \{0, \dots, m\}}$, for $m \geq 0$, where $\Psi_0 \in \text{Rew}[\Sigma, Q]$. There exists a prefix $(D_i)_{i \in \{0, \dots, m\}}$ of a chase quasi-derivation of D w.r.t. Σ such that, for every $i \in \{0, \dots, m\}$, $D_{m-i} \models \Psi_i$ implies $D_m \models \Psi_0$.*

The Final Proof. We are now ready to establish the equivalence $(\diamond)$, which in turn implies Lemma 3:

(1) $\Rightarrow$ (2). By hypothesis, there exists a chase derivation δ of D w.r.t. Σ such that $\text{chase}_\delta(D, \Sigma) \models Q$. By Lemma 4, there is $\Phi \in \text{Rew}[\Sigma, Q]$ such that $\text{chase}_\delta(D, \Sigma) \models \Phi$. Therefore, there exists a prefix $(D_i)_{i \in \{0, \dots, m\}}$, for $m \geq 0$, of δ such that $D_m \models \Phi$. By Lemma 5, there is a sequence $s = (\Psi_i)_{i \in \{0, \dots, m\}}$, for $m \geq 0$, with the properties stated in the statement of the lemma. Note that there is no guarantee that s is produced by the execution of QueryRewrite(O) under consideration. This is due to the blocking condition that should not be satisfied in order to keep a sentence produced via a resolution step. Clearly, $\Psi_0 \in \text{Rew}[\Sigma, Q]$ since $\Psi_0 = \Phi$. Via an inductive argument on the length of s , we can show that the execution of QueryRewrite(O) under consideration produces a rewriting Σ -sequence $(\Psi'_i)_{i \in \{0, \dots, \ell\}}$, where $\ell \geq 0$, such that $\Psi'_0 = \Psi_0$ and Ψ_m is $\{\Psi'_\ell\}$ -blocked. Since $D \models \Psi_m$, we can show that $D \models \Psi'_\ell$, and the claim follows since Ψ'_ℓ is a disjunct of Q_O .

(2) $\Rightarrow$ (1). By hypothesis, $D \models \Phi$ for some disjunct Φ of Q_O . This implies that there exists a rewriting Σ -sequence $(\Psi_i)_{i \in \{0, \dots, m\}}$ such that $\Psi_0 \in \text{Rew}[\Sigma, Q]$, $\Psi_m = \Phi$, and $D \models \Psi_m$. By Lemma 6, there is a prefix $(D_i)_{i \in \{0, \dots, m\}}$ of a chase quasi-derivation of D w.r.t. Σ such that, for $i \in \{0, \dots, m\}$, $D_{m-i} \models \Psi_i$ implies $D_m \models \Psi_0$. Therefore, since $D = D_0$, we get that $D_0 \models \Psi_m$, and thus, $D_m \models \Psi_0$. Interestingly, by using the fact that $\Psi_0 \in \text{Rew}[\Sigma, Q]$, we can establish a monotonicity property for Ψ_0 (which trivially holds for cqs), and show that $\text{chase}_\kappa(D, \Sigma) \models \Psi_0$, where κ is a chase quasi-derivation of D w.r.t. Σ that expands $(D_i)_{i \in \{0, \dots, m\}}$. Therefore, by Lemma 4, we get that $\text{chase}_\kappa(D, \Sigma) \models Q$, and the claim follows since $\text{chase}_\kappa(D, \Sigma) = \text{chase}(D, \Sigma)$.

5. Future Work

The algorithmic characterization established in this work is for uniform first-order rewritability, that is, for omq classes $(C, \text{CQ}^\neg)$, where C is a class of full tgds, that adopt the query language $\text{CQ}^\neg$ as a whole for expressing the user query. There are, of course, cases where $(C, \text{CQ}^\neg)$ is not first-order rewritable, but there is a query language $Q \subsetneq \text{CQ}^\neg$ such that the class of omqs (C, Q) is first-order rewritable. Establishing an algorithmic characterization of first-order rewritable omq classes (C, Q) , where C is a class of full tgds and $Q \subsetneq \text{CQ}^\neg$, is a technically more challenging problem that deserves our attention.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] S. Abiteboul, R. Hull, V. Vianu, *Foundations of Databases*, Addison-Wesley, 1995.
- [2] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, *J. Data Semantics* 10 (2008) 133–173.
- [3] M. Bienvenu, B. ten Cate, C. Lutz, F. Wolter, Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP, *ACM Trans. Database Syst.* 39 (2014) 33:1–33:44.
- [4] A. Cali, G. Gottlob, T. Lukasiewicz, A general Datalog-based framework for tractable query answering over ontologies, *J. Web Sem.* 14 (2012) 57–83.
- [5] A. Cali, G. Gottlob, A. Pieris, Towards more expressive ontology languages: The query answering problem, *Artif. Intell.* 193 (2012) 87–128.
- [6] G. Gottlob, G. Orsi, A. Pieris, Query rewriting and optimization for ontological databases, *ACM Trans. Database Syst.* 39 (2014) 25:1–25:46.
- [7] M. König, M. Leclère, M. Mugnier, M. Thomazo, Sound, complete and minimal ucq-rewriting for existential rules, *Semantic Web* 6 (2015) 451–475.
- [8] M. Alviano, A. Pieris, Default negation for non-guarded existential rules, in: *PODS*, 2015, pp. 79–90.
- [9] J. Baget, F. Garreau, M. Mugnier, S. Rocher, Extending acyclicity notions for existential rules, in: *ECAI*, 2014, pp. 39–44.
- [10] A. Hernich, C. Kupke, T. Lukasiewicz, G. Gottlob, Well-founded semantics for extended datalog and ontological reasoning, in: *PODS*, 2013, pp. 225–236.
- [11] G. Gottlob, A. Hernich, C. Kupke, T. Lukasiewicz, Stable model semantics for guarded existential rules and description logics: Decidability and complexity, *J. ACM* 68 (2021) 35:1–35:87.
- [12] G. Gottlob, M. Manna, A. Pieris, A. Ricioppo, First-order rewritability of rule-based ontology mediated queries with negation, in: *ICDT*, 2026, pp. 21:1–21:20.
- [13] B. Rossman, Homomorphism preservation theorems, *J. ACM* 55 (2008) 15:1–15:53.