First-Order Rewritability of Rule-Based Ontology Mediated Queries with Negation

Georg Gottlob
University of Calabria, Rende, Italy

Marco Manna
University of Calabria, Rende, Italy

Andreas Pieris
University of Cyprus, Nicosia, Cyprus
University of Edinburgh, UK

Aldo Ricioppo
University of Calabria, Rende, Italy

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 fifteen years 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 $(\mathsf{OL}, \mathsf{QL})$ 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 $(\mathsf{OL}, \mathsf{CQ})$, 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. This research effort led an algorithmic characterization of when a rule-based omq language $(\mathsf{OL}, \mathsf{CQ})$ is first-order rewritable. More precisely, there is a uniform algorithm $\mathsf{Rewrite}$ such that, for every rule-based ontology language OL, the omq language $(\mathsf{OL}, \mathsf{CQ})$ is first-order rewritable iff for every omq O from $(\mathsf{OL}, \mathsf{CQ})$, the algorithm $\mathsf{Rewrite}$ on input O terminates and constructs a first-order rewriting of O. The question that we are interested in is whether the above algorithmic characterization can be extended to rule-based omq languages of the form $(\mathsf{OL}, \mathsf{CQ}^\neg)$, where $\mathsf{CQ}^\neg$ is the language of conjunctive queries with the useful feature of negation. The goal of this work is to initiate effort towards the settlement of the above highly non-trivial question. To this end, we provide a new algorithm, which is a non-trivial extension of the algorithm $\mathsf{Rewrite}$ for positive omqs, and show the following: under the Skolem semantics, a well-established approach for defining the answer to a rule-based omq when the user query can use negation, the proposed algorithm is a first-order rewriter for $(\mathsf{OL}, \mathsf{CQ}^\neg)$, where OL is the language of linear or acyclic tuple-generating dependencies, two central rule-based ontology languages that ensure first-order rewritability for positive omqs. We strongly believe that the new algorithm can serve as a good starting point towards the full settlement of our main question.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases ontology-mediated queries, tuple-generating dependencies, conjunctive queries with negation, first-order rewritability

Digital Object Identifier 10.4230/LIPIcs.ICDT.2026.21

Funding This work was supported by the University of Cyprus, NextGenerationEU through the project FAIR (CUP H23C22000860006), Italian Ministry of University and Research through the project PRODE (CUP H53D23003420006), Italian Ministry of Enterprises and Made in Italy through the projects ASVIN (CUP B29J24000200005) and PreDiCI (CUP B29J24000300005), and Calabrian Region through the project NextGenGuides (CUP J39I24002040005).

1 Introduction

Ontology-mediates Queries. 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) form a prominent rule-based language, originally introduced as a unifying framework for database constraints [1]. Tgds are first-order sentences 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 (the body) and ψ (the head) are conjunctions of relational atoms, and they essentially state that certain tuples in a database imply the presence of some other tuples in the database. The last fifteen years 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. The marriage of ontologies with user queries has been originally proposed in the context of ontology-based data integration in order to facilitate access to data that is heterogeneous and incomplete [14]. The ontology and the user query can be conveniently seen as two components of one composite query, called ontology-mediated query (omq) [6]. In this context, the main problem of interest is to compute the answer to such an omq over a database. In particular, 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 compute the certain answers to O over D, denoted $\mathsf{ans}(O, D)$, 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, i.e., 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, $\mathsf{ans}(O, D)$ coincides with the answer to Q_O over D. It is of course essential that the query Q_O is expressed in a query language that can be handled by standard database systems. The typical target query language that is considered 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. This research effort led to several syntactic classes C of tgds that give rise to first-order rewritable classes of omqs $(\mathsf{C}, \mathsf{CQ})$, i.e., every omq (Σ, Q) of $(\mathsf{C}, \mathsf{CQ})$, 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., [7, 8, 10, 12]. What is even more remarkable is the fact that every first-order rewritable tgd-based class of omqs, no matter how the class of tgds is defined, can be handled via one uniform rewriting algorithm. More precisely, there is a resolution-based algorithm $\mathsf{Rewrite}$ such that, for every class C of tgds, the omq class $(\mathsf{C}, \mathsf{CQ})$ is first-order rewritable iff for every omq O from $(\mathsf{C}, \mathsf{CQ})$, $\mathsf{Rewrite}$ on input O terminates and returns a first-order rewriting of O [12]. This provides an algorithmic characterization of when an omq class $(\mathsf{C}, \mathsf{CQ})$, 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, 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 $(\mathsf{C}, \mathsf{CQ}^{\neg})$, where the user query comes from the class of conjunctive queries

with negation, denoted $\mathsf{CQ}^{\neg}$. The problem of evaluating such non-positive omqs has also attracted attention in the literature; see, e.g., [3, 4, 7, 9, 11]. Having said that, answering non-positive omqs of this kind via first-order rewritability, unlike the case of positive omqs discussed above, is largely unexplored. An interesting question is whether the algorithmic characterization discussed above concerning the first-order rewritability of positive omqs can be extended to non-positive omqs. More precisely, we are asking whether there exists one uniform algorithm, which does not depend on any class of tgds, that acts as a first-order rewriter for every class $(\mathsf{C}, \mathsf{CQ}^{\neg})$ of tgd-based omqs that is first-order rewritable. The goal of this work is to initiate effort towards the settlement of this highly non-trivial question. But first we need to discuss how the answer to such omqs should be defined, and, in fact, whether the standard certain answers semantics is conceptually meaningful in this case.

Consider the database $D = \{R(a), S(b)\}$ and the omq $O = (\Sigma, Q)$, where Σ consists of the tgd $\forall x(R(x) \rightarrow \exists y\, P(x, y))$ and Q is the Boolean query Ans :- $R(x), P(x, y), \neg S(x)$. The answer to O over D is empty (i.e., the query is false) since there exists at least one model of D and Σ that does not entail Q; e.g., the model $I = \{R(a), S(b), P(a, a), S(a)\}$ does not entail Q due to the atom $S(a)$ since the only way to match the positive part of Q to I is by mapping both variables x and y to a, whereas $S(a)$ belongs to I. On the other hand, $S(a)$ is not supported by D and Σ. Indeed, there is no reason to derive the atom $S(a)$ since the instance $I \setminus \{S(a)\}$ is also a model of D and Σ. Hence, the model I should not be a valid reason for the non-entailment of Q by D and Σ. The above example can be generalized and show that for a database D and an omq $O = (\Sigma, Q)$, where Σ is set of tgds and Q a conjunctive query *with at least one negated atom*, there is always a model I of D and Σ such that the answer to Q over I is empty – I is simply the model that collects all the atoms that can be formed using the predicates of the schema and the constants in D and Q.

The above discussion leads to the natural idea that has been adopted in the literature of considering only models without redundant information that may distort the expected answer to the given omq, i.e., considering only *minimal models*. Coming back to the above example, it is easy to see that Q is entailed by every minimal model of D and Σ. Hence, the main problem of interest in this context is as follows: given a database D and an omq (Σ, Q), where Σ is a set of tgds and Q a conjunctive query with negation, compute all the tuples of database elements that are answers to Q derived by every minimal model of D and Σ.

Once we adopt the minimal model semantics, there is one more aspect that we need to carefully consider, which has to do with the existentially quantified variables in tgd-heads, that greatly affects the obtained semantics: either keep those existentially quantified variables, or eliminate them via the standard technique of Skolemization. Both approaches are quite relevant and have been considered in the literature; see, e.g., [2, 9]. In this work, we focus on the latter, which we call Skolem semantics. Let us clarify that for positive tgd-based omqs, considering all the models or only the minimal models, and keeping or eliminating the existentially quantified variables in tgd-heads, lead to the exact same semantics.

Our Contribution. This work makes a first non-trivial step towards an algorithmic characterization of when an omq class $(\mathsf{C}, \mathsf{CQ}^{\neg})$, where C is a class of tgds, is first-order rewritable. In particular, we establish the existence of an algorithm that, no matter which class C of tgds we consider among linear or acyclic tgds – two central classes of tgds that ensure first-order rewritability for positive omqs – is a first-order rewriter for $(\mathsf{C}, \mathsf{CQ}^{\neg})$ under the Skolem semantics. This new algorithm is significantly more sophisticated than the one for positive omqs as it needs to carefully treat the negative part of the user query. We strongly believe that it can serve as a good starting point towards an algorithmic characterization of first-order rewritability for non-positive classes of tgd-based omqs under the Skolem semantics.

Interestingly, the proposed algorithm produces rewritings that fall in a novel fragment of first-order queries, which is strictly less expressive than the full language of first-order queries. We also show that unions of conjunctive queries with equality and negation are not powerful enough for our purposes as one might think, influenced by the fact that unions of conjunctive queries with equality are enough for positive tgd-based omqs.

2 Preliminaries

We proceed to introduce the basic notions about relational instances and databases, relational queries, and tuple-generating dependencies that will be used throughout the paper. For brevity, given an integer $n > 0$, we may write $[n]$ for the set of integers $\{1, \ldots, n\}$.

Basic Vocabularies. Let $\mathbf{C}$, $\mathbf{N}$ and $\mathbf{V}$ be disjoint countably infinite sets of *constants*, *(marked) nulls* and *variables*, respectively. We further consider the countably infinite set $\mathbf{F}$ of *function symbols* (each of them associated with a positive arity), which is disjoint from $\mathbf{C}$, $\mathbf{N}$ and $\mathbf{V}$. We assume that different constants represent different objects and different function symbols represent different functions (*unique name assumption*). *Functional terms* are inductively defined as follows: (i) variables are functional terms, and (ii) $f(t_1, \ldots, t_n)$, where $n > 0$, f is an n-ary function symbol of $\mathbf{F}$, and $t_1, \ldots, t_n$ are functional terms, is a functional term. *Ground functional terms* are defined similarly: (i) constants of $\mathbf{C}$ are ground functional terms, and (ii) $f(t_1, \ldots, t_n)$, where $n > 0$, f is an n-ary function symbol of $\mathbf{F}$, and $t_1, \ldots, t_n$ are ground functional terms, is a ground functional term. We refer to constants, nulls, variables, functional terms, and ground functional terms as *terms*.

Relational Atoms and Homomorphisms. A *schema* $\mathbf{S}$ is a finite set of relation names (or predicates) with associated positive arity. We write R/n to say that R has arity $n > 0$; we may write $\mathsf{ar}(R)$ for the integer 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; note that we may treat a tuple as the set of its elements. For a set A of atoms, we write $\mathsf{terms}(A)$ and $\mathsf{var}(A)$ for the set of terms and variables, respectively, mentioned in the atoms of A.

A central notion is that of homomorphism that allows us to embed a set of atoms into another set of atoms while preserving its structure. Formally, a *homomorphism* from a set of atoms A to a set of atoms B is a function $h : \mathsf{terms}(A) \to \mathsf{terms}(B)$ that is the identity on $\mathbf{C}$ with $R(h(\bar{t})) \in B$ for every $R(\bar{t}) \in A$; for a term $f(t_1, \ldots, t_n)$, $h(f(t_1, \ldots, t_n)) = f(h(t_1), \ldots, h(t_n))$. We write $A \to B$ to denote that there is a homomorphism from A to B.

Relational Instances and Databases. We make the distinction between instances that mention only constants and nulls and instances that mention only ground functional terms. A *null instance* over a schema $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that mention only constants and nulls. A *functional instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that mention only ground functional terms. A *database* over $\mathbf{S}$ is a non-empty finite set of atoms over $\mathbf{S}$ that mention only constants. Clearly, a database over $\mathbf{S}$ is trivially a null and a functional instance over $\mathbf{S}$. We write $\mathsf{dom}(I)$ for the set of terms occurring in an instance I.

First-order Queries. We assume the reader is familiar with the syntax and semantics of first-order logic. We consider only function-free first-order formulas, i.e., formulas that mention variables and constants of $\mathbf{C}$, but no function symbols. For such a formula Φ, we write $\mathsf{fv}(\Phi)$ and $\mathsf{dom}(\Phi)$ for the set of its free variables and the set of constants occurring in it, respectively. We use the standard symbol $\models$ for the notion of satisfaction of sentences. A

first-order query Q over a schema $\mathbf{S}$ is an expression $\{(x_1, \ldots, x_k) \mid \Phi\}$, for $k \geq 0$, where Φ is a first-order formula over $\mathbf{S}$ and $(x_1, \ldots, x_k)$ is a tuple over $\mathsf{fv}(\Phi)$ such that each variable of $\mathsf{fv}(\Phi)$ occurs in $(x_1, \ldots, x_k)$ at least once, i.e., $\mathsf{fv}(\Phi) = \{x_1, \ldots, x_k\}$. Note that a first-order query is always a finite object, i.e., the first-order formula Φ above is finite. For a tuple of constants $(c_1, \ldots, c_k)$, we write $\Phi(c_1, \ldots, c_k)$ for the sentence obtained from Φ after replacing each occurrence of x_i with c_i, for each $i \in [k]$. Given a database D over $\mathbf{S}$, the *answer to* Q *over* D is the set $Q(D) = \big\{(c_1, \ldots, c_k) \in (\mathsf{dom}(D) \cup \mathsf{dom}(\Phi))^k \mid D \models \Phi(c_1, \ldots, c_k)\big\}$.

Conjunctive Queries with Negation. A restricted class of first-order queries that is crucial for our work is that of conjunctive queries with negation. Formally, a *conjunctive query with negation* (ncq) Q over a schema $\mathbf{S}$ is a first-order query $\{\bar{x} \mid \Phi\}$ over $\mathbf{S}$ with Φ being a formula of the form $\exists \bar{y}\,(R_1(\bar{y}_1) \wedge \cdots \wedge R_n(\bar{y}_n) \wedge \neg P_1(\bar{z}_1) \wedge \cdots \wedge \neg P_m(\bar{z}_m))$ for $n \geq 1$ and $m \geq 0$, where (i) $R_i(\bar{y}_i)$ and $P_j(\bar{z}_j)$ are atoms with $\bar{y}_i$ and $\bar{z}_j$ being tuples of constants and variables mentioned in $\bar{x}$ and $\bar{y}$, for each $i \in [n]$ and $j \in [m]$, and (ii) 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*. Note that if $m = 0$, i.e., there are no negated atoms, then Q is simply a conjunctive query (cq). We write Q^+ for the non-empty set of atoms $\{R_1(\bar{y}_1), \ldots, R_n(\bar{y}_n)\}$ and Q^- for the (possibly empty) set of atoms $\{P_1(\bar{z}_1), \ldots, P_m(\bar{z}_m)\}$. Let $\mathsf{CQ}^\neg$ be the set of all ncqs and CQ the set of all cqs. It is well-known that the semantics of ncqs can be conveniently defined via homomorphisms. Given a null or functional instance I, the *answer to* Q *over* I, denoted $Q(I)$, is the set of all tuples $\bar{c} \in \mathbf{C}^{|\bar{x}|}$ such that (i) there is a homomorphism h from Q^+ to I with $h(\bar{x}) = \bar{c}$, and (ii) for each $i \in [m]$, $P_i(h(\bar{z}_i)) \notin I$. Note that, unlike arbitrary first-order queries, the answer to ncqs is defined over arbitrary instances, and not only over databases, since this is needed for our technical development.

Tuple-generating Dependencies. A *tuple-generating dependency* (tgd) σ over a schema $\mathbf{S}$ is a first-order sentence $\forall \bar{x} \forall \bar{y}\big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})\big)$, 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 $\phi(\bar{x}, \bar{y})$ and $\psi(\bar{x}, \bar{z})$ the *body* and *head* of σ, denoted $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$, respectively. Note that we may treat a conjunction of atoms as the set of its atoms, which means that $\mathsf{body}(\sigma)$ and $\mathsf{head}(\sigma)$ can be treated as sets of atoms. We denote by $\mathsf{fr}(\sigma)$ the so-called *frontier variables* of σ, i.e., the universally quantified variables of σ that appear in $\mathsf{head}(\sigma)$. Let TGD be the family of all finite sets of tgds; in the rest of the paper, a set of tgds is always finite. A class C of tgds is simply a subset of TGD. A null instance I over $\mathbf{S}$ satisfies σ, written $I \models \sigma$, if, whenever $\phi(\bar{x}, \bar{y}) \rightarrow I$ via h, then $\psi(\bar{x}, \bar{z}) \rightarrow I$ via h' that agrees with h on $\bar{x}$. The null instance I satisfies a set Σ of tgds, written $I \models \Sigma$, if $I \models \sigma$ for each $\sigma \in \Sigma$. Given a database D over $\mathbf{S}$, a *model* of D and Σ is a (possibly infinite) null instance $I \supseteq D$ such that $I \models \Sigma$.

From Tuple-generating Dependencies to Logic Programs. We now discuss how a set of tgds can be transformed into a logic program by eliminating the existentially quantified variables via Skolemization. This is needed for defining the semantics of ontology-mediated queries below. The Skolemization of a tgd σ of the form $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$, denoted $\mathsf{Sk}(\sigma)$, is the logic programming rule (or simply LP-rule), $\phi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \mathbf{f}_\sigma(\bar{x}, \bar{y}))$, where $\mathbf{f}_\sigma$ is a vector of function symbols $f_{\sigma,z} \in \mathbf{F}$, one for each variable $z \in \bar{z}$. For example, the Skolemization of the tgd $\sigma = R(x, y) \rightarrow \exists z \exists w\, P(z, y, w)$ is the LP-rule $R(x, y) \rightarrow P(f_{\sigma,z}(x, y), y, f_{\sigma,w}(x, y))$. The Skolemization of a set Σ of tgds, denoted $\mathsf{Sk}(\Sigma)$, is the set of LP-rules $\{\mathsf{Sk}(\sigma) \mid \sigma \in \Sigma\}$. A functional instance I satisfies $\mathsf{Sk}(\sigma)$, where σ is a tgd as above, written $I \models \mathsf{Sk}(\sigma)$, if, whenever $\phi(\bar{x}, \bar{y}) \rightarrow I$ via a homomorphism h, then $\psi(\bar{x}, \mathbf{f}_\sigma(\bar{x}, \bar{y})) \rightarrow I$ via h. The instance I

satisfies $\mathsf{Sk}(\Sigma)$ for a set Σ of tgds, written $I \models \mathsf{Sk}(\Sigma)$, if $I \models \mathsf{Sk}(\sigma)$ for each $\sigma \in \Sigma$. Given a database D, a *model* of D and $\mathsf{Sk}(\Sigma)$ is a (possibly infinite) functional instance $I \supseteq D$ such that $I \models \Sigma$. A model I of D and $\mathsf{Sk}(\Sigma)$ is *minimal* if there is no $J \subset I$ that is a model of D and $\mathsf{Sk}(\Sigma)$. In fact, it is easy to show that there is a unique minimal model of D and $\mathsf{Sk}(\Sigma)$, and thus, we can refer to *the* minimal model of D and $\mathsf{Sk}(\Sigma)$, denoted $\mathsf{mm}(D, \mathsf{Sk}(\Sigma))$.

Ontology-mediated Queries. An *ontology-mediated query* (omq) over a schema $\mathbf{S}$ is a pair $O = (\Sigma, Q)$, where Σ is a set of tgds over $\mathbf{S}$ (*the ontology*) and Q is an ncq over $\mathbf{S}$ (*the user query*). For a class $\mathsf{C} \subseteq \mathsf{TGD}$ of tgds, let $(\mathsf{C}, \mathsf{CQ}^\neg)$ be the class of omqs (Σ, Q), where $\Sigma \in \mathsf{C}$ and $Q \in \mathsf{CQ}^\neg$. The class of omqs $(\mathsf{C}, \mathsf{CQ})$ is defined analogously. One of the standard ways for defining the answer to an omq (Σ, Q) is to first convert Σ into the logic program $\mathsf{Sk}(\Sigma)$ and then evaluate the user query over the unique minimal model of the input database and $\mathsf{Sk}(\Sigma)$. Formally, given a database D and an omq $O = (\Sigma, Q)$, the *answer to O over D*, denoted $\mathsf{ans}_{\mathsf{Sk}}(O, D)$, is defined as $Q(\mathsf{mm}(D, \mathsf{Sk}(\Sigma)))$. At this point, let us stress that this semantics extends the well-established certain answers semantics for positive omqs, where the user query is a cq *without* negation. More precisely, for an omq $O = (\Sigma, Q)$ over $\mathbf{S}$, where Q is a cq without negation, the *certain answers* of O over a database D, denoted $\mathsf{certain}(O, D)$, is defined as $\bigcap_{I \in \mathsf{mods}(D,\Sigma)} Q(I)$, where $\mathsf{mods}(D, \Sigma)$ collects all the models of D and Σ. It is then not difficult to show that $\mathsf{certain}(O, D) = \mathsf{ans}_{\mathsf{Sk}}(O, D)$.

Querying the Unique Minimal Model and the Chase Procedure. Given a database D and an omq $O = (\Sigma, Q)$, we can use the chase procedure, a useful tool when reasoning with tgds, to construct a model of D and Σ that is isomorphic to the unique minimal model of D and $\mathsf{Sk}(\Sigma)$, which in turn can be used for computing $\mathsf{ans}_{\mathsf{Sk}}(O, D)$. This allows us to directly work with the original set of tgds, instead of its Skolemized version, which is technically more convenient. We assume the reader is familiar with the oblivious version of the chase procedure; see, e.g., [5]. Given a chase derivation δ of a database D w.r.t. a set Σ of tgds, which is essentially a sequence of instances $(I_i)_{i \geq 0}$ where $I_0 = D$ and, for each $i > 0$, I_{i+1} is obtained from I_i by applying a tgd of Σ, we write $\mathsf{chase}_\delta(D, \Sigma)$ for the result of δ, that is, the instance $\bigcup_{i \geq 0} I_i$. It is not difficult to show that there is a bijection μ from the constants and nulls in $\mathsf{chase}_\delta(D, \Sigma)$ to the ground functional terms in $\mathsf{mm}(D, \mathsf{Sk}(\Sigma))$, which is the identity on $\mathbf{C}$, such that $\mu(\mathsf{chase}_\delta(D, \Sigma)) = \mathsf{mm}(D, \mathsf{Sk}(\Sigma))$ and $\mu^{-1}(\mathsf{mm}(D, \mathsf{Sk}(\Sigma))) = \mathsf{chase}_\delta(D, \Sigma)$. Thus, we get the following result that allows us to use the chase procedure for our purposes:

▶ **Proposition 1.** *Consider a database D and an omq (Σ, Q). For every chase derivation δ of D w.r.t. Σ, it holds that $\mathsf{ans}_{\mathsf{Sk}}(O, D) = Q(\mathsf{chase}_\delta(D, \Sigma))$.*

3 Ontology-mediated Queries and First-order Rewritability

Our main objective is to study the central concept 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. The formal definition of first-order rewritability for omqs follows:

▶ **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 $\mathbf{S}$, $\mathsf{ans}_{\mathsf{Sk}}(O, D) = Q_O(D)$. A class $(\mathsf{C}, \mathsf{CQ}^\neg)$ of omqs, where $\mathsf{C} \subseteq \mathsf{TGD}$, is* first-order rewritable *if every omq from $(\mathsf{C}, \mathsf{CQ}^\neg)$ has a first-order rewriting. A* first-order rewriter *for a first-order rewritable class $(\mathsf{C}, \mathsf{CQ}^\neg)$ of omqs is a deterministic algorithm that takes as input an omq $O \in (\mathsf{C}, \mathsf{CQ}^\neg)$ and constructs a first-order rewriting of O. The notions of first-order rewritability and first-order rewriter for a class of omqs $(\mathsf{C}, \mathsf{CQ})$ are defined analogously.* ⌟

The notion of first-order rewritability for positive classes of omqs, i.e., classes of omqs $(\mathsf{C}, \mathsf{CQ})$, where $\mathsf{C} \subseteq \mathsf{TGD}$, has been extensively studied in the database and KR literature. As discussed in Section 1, this research effort led to an algorithmic characterization:

▶ **Theorem 3** ([12]). *There exists a deterministic algorithm* Rewrite *such that, for every class* C *of tgds,* $(\mathsf{C}, \mathsf{CQ})$ *is first-order rewritable iff* Rewrite *is a first-order rewriter for* $(\mathsf{C}, \mathsf{CQ})$.

We are still far from an analogous characterization for non-positive omqs. However, we can establish the existence of an algorithm that, no matter which class C of tgds we consider among linear or acyclic tgds, it is a first-order rewriter for $(\mathsf{C}, \mathsf{CQ}^{\neg})$. The syntactic notions of linearity [7] and acyclicity [13], which we recall below, give rise to central classes of tgds that ensure first-order rewritability for positive omqs:

Linearity. A tgd σ is called *linear* if it has exactly one atom in its body, i.e., $|\mathsf{body}(\sigma)| = 1$. We write Linear for the class of linear tgds, i.e., the family of all finite sets of linear tgds.

Acyclicity. The notion of acyclicity essentially forbids recursive statements. This is typically formalized via the acyclictiy of the predicate graph of a set Σ of tgds. The *predicate graph* of Σ, denoted $\mathsf{pg}(\Sigma)$, is a directed graph (V, E), where V (the set of nodes) consists of all the predicates occurring in Σ, and E (the set of edges) contains an edge (R, P) iff there is a tgd $\sigma \in \Sigma$ that mentions R in its body and P in its head. Essentially, an edge (R, P) encodes the fact that P depends on R. We say that Σ is acyclic if $\mathsf{pg}(\Sigma)$ is acyclic. We write Acyclic for the family of all finite sets of tgds that are acyclic.

The main result of our work, which can be seen as a first non-trivial step towards an algorithmic characterization of first-order rewritability for non-positive omqs, follows:

▶ **Theorem 4.** *There exists a deterministic algorithm* $\mathsf{Rewrite}^{\neg}$ *that is a first-order rewriter for* $(\mathsf{Linear}, \mathsf{CQ}^{\neg})$ *and* $(\mathsf{Acyclic}, \mathsf{CQ}^{\neg})$.

The rest of the paper is devoted to discussing the proof of the above result.

Target Query Language

Before we present the algorithm $\mathsf{Rewrite}^{\neg}$ in Section 4, and show that is indeed a first-order rewriter for $(\mathsf{C}, \mathsf{CQ}^{\neg})$, where $\mathsf{C} \in \{\mathsf{Linear}, \mathsf{Acyclic}\}$, let us conclude this section by discussing its target query language. In the case of positive omq classes $(\mathsf{C}, \mathsf{CQ})$, we know that there is no need to go beyond unions of cqs with equality, which forms a strictly less expressive query language than first-order queries; let $\mathsf{UCQ}^{=}$ be the set all unions of cqs with equality. In other words, for every omq O from $(\mathsf{C}, \mathsf{CQ})$, the algorithm Rewrite, provided by Theorem 3, on input O outputs a first-order rewriting of O that falls in $\mathsf{UCQ}^{=}$. With this in mind, one may be tempted to think that the algorithm $\mathsf{Rewrite}^{\neg}$ should target a query that falls in the language of unions of cqs with equality and negation, denoted $\mathsf{UCQ}^{=,\neg}$. However, we can show that the latter language is not expressive enough for our purposes.

▶ **Proposition 5.** *There exists an omq* $O = (\Sigma, Q)$, *where* $\Sigma \in \mathsf{Linear} \cap \mathsf{Acyclic}$ *and* $Q \in \mathsf{CQ}^{\neg}$, *that admits a first-order rewriting but, for every query* Q *from* $\mathsf{UCQ}^{=,\neg}$, *there exists a database* D *such that* $\mathsf{ans}_{\mathsf{Sk}}(O, D) \neq Q(D)$.

As we shall see, whenever $\mathsf{Rewrite}^{\neg}$ terminates, the obtained query falls in a restricted class of first-order queries, which we call unions of conditional conjunctive queries with equalities, that forms a query language that is strictly less expressive than arbitrary first-order queries. Roughly, a conditional conjunctive query with equality is a cq with equality with additional

conditions that can be expressed via inequalities and the negation of conjunctions of atoms (hence the name conditional cq with equality). For instance, ncqs form a special case of conditional cqs with equality. Formally, a *conditional conjunctive query with equality* ($\text{ccq}^{=}$) Q over a schema $\mathbf{S}$ is a first-order query $\{\bar{u} \mid \Phi\}$ over $\mathbf{S}$ with Φ being a formula of the form

$$\exists \bar{v} \left(\Phi_1(\bar{x}) \ \wedge \ \Phi_2(\bar{y}) \ \wedge \ \neg \left(\bigvee_{i=1}^{n} \exists \bar{z}_i \left(\Phi_3^i(\bar{z}_i, \bar{w}_i) \right) \right) \right)$$

for $\bar{x} = \mathsf{fv}(\Phi) \cup \bar{v}$ and $n \geq 0$ (if $n = 0$, then the negated disjunction is not present), where

- $\Phi_1(\bar{x})$ is a non-empty conjunction of atoms and equalities that mention constants and variables of $\bar{x}$, which means that $\{\bar{u} \mid \exists \bar{v}\, \Phi_1(\bar{x})\}$ is a cq with equality; for brevity, we write $\mathsf{rel}(\Phi_1(\bar{x}))$ for the set of relational atoms and $\mathsf{eq}(\Phi_1(\bar{x}))$ for the set of equalities in $\Phi_1(\bar{x})$,
- $\Phi_2(\bar{y})$ is a (possibly empty) conjunction of inequalities over $\mathbf{C}$ and $\bar{y} \subseteq \bar{x}$, and
- $\Phi_3^i(\bar{z}_i, \bar{w}_i)$, for each $i \in [n]$, is a non-empty conjunction of atoms that mention constants and variables of $\bar{z}_i \cup \bar{w}_i$ with $\bar{x} \cap \bar{z}_i = \emptyset$ and $\bar{w}_i \subseteq \bar{x}$.

Let $\mathsf{CCQ}^{=}$ be the set of all ccqs with equality. The semantics of ccqs with equality can be defined via homomorphisms. Formally, given a null instance I, the *answer to Q over I*, denoted $Q(I)$, is the set of all tuples of constants $\bar{c} \in \mathbf{C}^{|\bar{u}|}$ such that (i) there is a homomorphism h from $\mathsf{rel}(\Phi_1(\bar{x}))$ to I with $h(\bar{u}) = \bar{c}$, (ii) for each $t = t'$ in $\mathsf{eq}(\Phi_1(\bar{x}))$, $h(t) = h(t')$, (iii) for each $t \neq t'$ in $\Phi_2(\bar{y})$, $h(t) \neq h(t')$, and (iv) for each $i \in [n]$, there is no homomorphism from $\Phi_3^i(\bar{z}_i, h(\bar{w}_i))$ to I. A *union of ccqs with equality* ($\text{uccq}^{=}$) Q over $\mathbf{S}$ is a first-order query $\{\bar{u} \mid \Phi\}$ over $\mathbf{S}$ with Φ being a formula of the form $\Psi_1 \vee \cdots \vee \Psi_n$, for $n \geq 1$, and $Q_i = \{\bar{u} \mid \Psi_i\}$ being a $\text{ccq}^{=}$, for every $i \in [n]$. Let $\mathsf{UCCQ}^{=}$ be the set of all uccqs with equality. Given a null instance I, the *answer to Q over I*, denoted $Q(I)$, is $\bigcup_{i \in [n]} Q_i(I)$. Let us stress that uccqs with equality are indeed strictly less expressive than first-order queries. For instance, we can show that the first-order query $\{() \mid \forall x\, P(x)\}$ cannot be equivalently expressed as a $\text{uccq}^{=}$, which implies the following result:

▶ **Proposition 6.** *There exists a first-order query Q such that, for every query Q' from $\mathsf{UCCQ}^{=}$, there exists a database D with $Q(D) \neq Q'(D)$.*

In our analysis, we will also consider unions of ccqs with equality that have infinitely many disjuncts. Formally, *an infinitary union of ccqs with equality* Q over $\mathbf{S}$ is an expression $\{(x_1, \ldots, x_k) \mid \Phi\}$, for $k \geq 0$, where Φ is a formula $\bigvee_{i>0} \Psi_i$, the set of free variables $\mathsf{fv}(\Phi)$ of Φ is $\{x_1, \ldots, x_k\}$, and $Q_i = \{(x_1, \ldots, x_k) \mid \Psi_i\}$ is a $\text{ccq}^{=}$, for every $i > 0$. The answer to Q over a null instance I, denoted $Q(I)$, is $\bigcup_{i>0} Q_i(I)$. Note that, strictly speaking, Q is *not* a first-order query since it is an infinite object, whereas first-order queries are always finite.

4 A Uniform First-order Rewriter

We now establish the existence of the algorithm $\mathsf{Rewrite}^{\neg}$ claimed in Theorem 4. To this end, we are going to show the following, which is the main technical result of our work:

▶ **Proposition 7.** *There exists a nondeterministic algorithm $\mathsf{NRewrite}^{\neg}$ such that:*

1. *For every $\mathsf{C} \subseteq \mathsf{TGD}$ with $(\mathsf{C}, \mathsf{CQ})$ being first-order rewritable and every omq $O \in (\mathsf{C}, \mathsf{CQ}^{\neg})$, the following holds: every execution of $\mathsf{NRewrite}^{\neg}(O)$ leads to an infinitary $\text{uccq}^{=}$ Q_O such that, for every database D, $\mathsf{ans}_{\mathsf{Sk}}(O, D) = Q_O(D)$.*
2. *For every $\mathsf{C} \in \{\mathsf{Linear}, \mathsf{Acyclic}\}$ and every omq $O \in (\mathsf{C}, \mathsf{CQ}^{\neg})$, it holds that every execution of $\mathsf{NRewrite}^{\neg}(O)$ terminates.*

As we shall discuss later, the assumption in item (1) that $(\mathsf{C}, \mathsf{CQ})$ is first-order rewritable is crucial for guaranteeing that a possibly non-terminating execution of $\mathsf{NRewrite}^\neg$ on input O converges into an infinitary uccq$^=$ Q_O such that, for every database D, $\mathsf{ans}_{\mathsf{Sk}}(O, D) = Q_O(D)$. Since the properties stated in items (1) and (2) of Proposition 7 hold for every execution of $\mathsf{NRewrite}^\neg$, we get that every branch of the computation tree of $\mathsf{NRewrite}^\neg$ corresponds to the computation of a deterministic algorithm that is a first-order rewriter for $(\mathsf{C}, \mathsf{CQ}^\neg)$, where $\mathsf{C} \in \{\mathsf{Linear}, \mathsf{Acyclic}\}$, as needed. The rest of the section is devoted to discussing the proof of Proposition 7. We start by presenting the nondeterministic algorithm $\mathsf{NRewrite}^\neg$.

4.1 The Nondeterministic Algorithm NRewrite$^\neg$

As for positive omqs, the high-level idea of the rewriting algorithm is to start from the positive part of the given ncq and mimic backwards the chase steps that lead to its image in the result of the chase via resolution steps that use the tgds as rewriting rules until we reach the input database; the latter is implemented using the notion of chunk unifier that has been used for rewriting positive omqs. This rewriting process should lead to a union of ccqs with equality that can be directly evaluated over the input database. Unlike the positive case, however, we need to carefully treat the negative part of the query, which makes our task significantly more challenging compared to the positive case. We start by recalling the notion of chunk unifier from [5], which is the building block of our algorithm, explain how it operates on ccqs with equality, and then discuss the rewriting algorithm itself.

Chunk Unifiers

Let A and B be non-empty sets of atoms with constants and variables. The sets A and B *unify* if there is a function θ, which is the identity over constants, 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 function θ'. It is known that if two sets of atoms unify, then there is a unique mgu (up to variable renaming). Now, consider a set of atoms A with constants and variables and a set $U \subseteq \mathsf{var}(A)$. Consider also a tgd σ that does not share variables with A. A *(most general) chunk unifier of A with σ that respects U* is a triple (A_1, A_2, θ), where $\emptyset \subset A_1 \subseteq A$, $\emptyset \subset A_2 \subseteq \mathsf{head}(\sigma)$, and θ is a (most general) unifier for A_1 and A_2 such that, for each existential variable x of σ occurring in A_2, (i) $\theta(x) \notin \mathbf{C}$, and (ii) for every variable y with $x \neq y$, $\theta(x) = \theta(y)$ implies $y \in \mathsf{var}(A_1) \setminus U$.

We now proceed to define the notion of chunk unifier for formulas that define ccqs with equality. Recall that a ccq$^=$ is a first-order query $\{\bar{t} \mid \Phi\}$ with Φ being a formula of the form

$$\exists \bar{v} \left(\Phi_1(\bar{x}) \wedge \Phi_2(\bar{y}) \wedge \neg \left(\bigvee_{i=1}^{n} \exists \bar{z}_i \left(\Phi_3^i(\bar{z}_i, \bar{w}_i) \right) \right) \right)$$

for $\bar{x} = \mathsf{fv}(\Phi) \cup \bar{v}$ and $n \geq 0$, where (i) $\Phi_1(\bar{x})$ is a non-empty conjunction of atoms and equalities that mention constants and variables of $\bar{x}$, (ii) $\Phi_2(\bar{y})$ is a (possibly empty) conjunction of inequalities over constants and variable of $\bar{y} \subseteq \bar{x}$, and (iii) $\Phi_3^i(\bar{z}_i, \bar{w}_i)$, for each $i \in [n]$, is a non-empty conjunction of atoms that mention constants and variables of $\bar{z}_i \cup \bar{w}_i$ with $\bar{x} \cap \bar{z}_i = \emptyset$ and $\bar{w}_i \subseteq \bar{x}$. In what follows, for convenience, we refer to formulas of the above form as formulas in *rewrite form*. For technical clarity, we will actually focus on certain formulas in rewrite form that we call saturated. Formally, a formula Φ in rewrite form as above is *saturated* if (i) for every pair of distinct variables x, y in $\bar{x}$, either the equality $x = y$ occurs in $\mathsf{eq}(\Phi_1(\bar{x}))$ or the inequality $x \neq y$ belongs to $\Phi_2(\bar{y})$, and (ii) for every variable

$x \in \bar{x}$ and constant $c \in \mathsf{dom}(\Phi)$, either $x = c$ occurs in $\mathsf{eq}(\Phi_1(\bar{x}))$ or $x \neq c$ belongs to $\Phi_2(\bar{y})$. This can be done w.l.o.g. since every ccq$^=$ can be equivalently rewritten as a uccq$^=$ that is defined via a disjunction of formulas in rewrite form that are saturated.

Consider now a formula Φ in rewrite form as above that is saturated. Given a subset A of $\mathsf{rel}(\Phi_1(\bar{x}))$ (recall that a conjunction of atoms may be treated as a set of atoms), a variable $x \in \mathsf{var}(A)$ is called *shared* if it belongs to $\mathsf{fv}(\Phi) \cup \mathsf{var}(\mathsf{rel}(\Phi_1(\bar{x}) \setminus A))$; in simple words, x is shared if its a free variable of Φ or it also occurs somewhere in $\mathsf{rel}(\Phi_1(\bar{x}))$ other than the atoms of A. Consider also a tgd σ. A *(most general) chunk unifier of* Φ *with* σ is a quadruple $\tau = (A_1, A_2, \theta, \theta')$, where (i) (A_1, A_2, θ) is a (most general) chunk unifier of $\mathsf{rel}(\Phi_1(\bar{x}))$ with σ that respects the shared variables of A_1, (ii) θ is the identity on $\mathsf{var}(A_1)$, and (ii) $\theta' : \mathsf{var}(\mathsf{body}(\sigma)) \setminus \mathsf{terms}(A_2) \to \mathsf{var}(\Phi_1(\bar{x}) \setminus A_1) \cup \mathsf{dom}(\Phi) \cup \mathsf{var}(\theta(\mathsf{body}(\sigma)))$. Note that, strictly speaking, θ is not defined on the variables of $\mathsf{var}(\mathsf{body}(\sigma)) \setminus \mathsf{terms}(A_2)$, and thus, we adopt the convention that θ is the identity over those variables.

Resolvents

Intuitively, an mgcu $\tau = (A_1, A_2, \theta, \theta')$ of Φ with σ essentially indicates that the subset A_1 of $\mathsf{rel}(\Phi_1(\bar{x}))$ can be satisfied in the result of the chase by applying the tgd σ. Thus, we can mimic this chase application backwards by resolving A_1 using σ, which is done by simply replacing A_1 in $\Phi_1(\bar{x})$ with $\mathsf{body}(\sigma)$ and properly updating the variables according to $\lambda = \theta \cup \theta'$; note that θ' specifies how to update the variables of $\mathsf{body}(\sigma)$ that are not already in A_2, and thus, $\theta \cup \theta'$ is a well-defined function. It should not be overlooked, however, that such a resolution step may lead to an unsafe formula, i.e., a formula where the negative part (including the inequalities) has a free variable that does not appear in the positive part, that is also not saturated. Hence, after resolving A_1 using σ, we need to apply a polishing step that converts the obtained formula into a safe formula in rewrite form that is saturated. Of course, the polishing should be carefully performed in order to avoid the generation of undesirable formulas that will lead to a uccq$^=$ that either computes wrong answers (it is unsound) or misses to compute correct answers (it is incomplete).

The above intuitive discussion about the resolution and the polishing steps are formalized via the notion of (σ, τ)-resolvent. In particular, the (σ, τ)-*resolvent* of Φ is defined as

$$\exists \bar{u} \left(\underbrace{\bigwedge_{\alpha \in (\Phi_1(\bar{x}) \setminus A_1) \cup \mathsf{body}(\sigma)} \lambda(\alpha)}_{\Psi_1} \wedge \underbrace{\widehat{\Phi_2}(\bar{y}')}_{\Psi_2} \wedge \neg \left(\bigvee_{i=1}^{n} \underbrace{\exists \bar{z}_i \, (\widehat{\Phi_3^i}(\bar{z}_i, \bar{w}_i))}_{\Psi_3^i} \right) \right),$$

where

- $\bar{u} = \mathsf{var}(\Psi_1) \setminus \mathsf{fv}(\Phi)$, i.e., it collects the variables in Ψ_1 other than the free variables of Φ,
- Ψ_2 is obtained from $\Phi_2(\bar{y})$ by (i) removing all the inequalities that mention a variable that does not occur in Ψ_1, and (ii) adding as conjuncts all the inequalities of the form $t \neq t'$, where t and t' are different variables of $\mathsf{var}(\Psi_1) \setminus \mathsf{var}(\Phi_1(\bar{x}))$, or $t \in \mathsf{var}(\Psi_1) \setminus \mathsf{var}(\Phi_1(\bar{x}))$ and $t' \in \mathsf{var}(\Phi_1(\bar{x})) \cap \mathsf{var}(\Psi_1)$, or $t \in \mathsf{var}(\Psi_1) \setminus \mathsf{var}(\Phi_1(\bar{x}))$ and $t' \in \mathsf{dom}(\Phi)$, and
- $\Psi_3^i = \exists \bar{z}_i \, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ if $\bar{w}_i \subseteq \mathsf{var}(\Psi_1)$; otherwise,

$$\Psi_3^i = \bigvee_{\mu \in M_i} \exists \bar{z}_\mu \left(\bigwedge_{\alpha \in \mu(\Phi_3^i(\bar{z}_i, \bar{w}_i)) \setminus \mu(\mathsf{head}(\sigma))} \alpha \right),$$

where $\bar{z}_\mu = \bar{z}_i \cap \mathsf{var}(\mu(\Phi_3^i(\bar{z}_i, \bar{w}_i)) \setminus \mu(\mathsf{head}(\sigma)))$, and M_i is the set of all functions of the form $\mu : \mathsf{terms}(B_1) \cup \mathsf{var}(\mathsf{head}(\sigma)) \to \mathsf{var}(B_1) \cup \mathsf{var}(A_1) \cup \mathsf{var}(\Psi_1) \cup \mathsf{dom}(\Phi)$, where $B_1 \subseteq \Phi_3^i(\bar{z}_i, \bar{w}_i)$ and $B_2 \subseteq \mathsf{head}(\sigma)$, such that the following hold:

- $\mu(B_1) = \mu(B_2)$,
- for each existential variable z of σ, if t is a constant, or a variable of $\mathsf{fr}(\sigma)$, or an existential variable of σ other than z, or a variable in $\Phi_3^i(\bar{z}_i, \bar{w}_i) \setminus B_1$, then $\mu(z) \neq \mu(t)$,
- μ is the identity over $\bar{x}$, and
- $\mu(v) = \lambda(v)$, for every variable $v \in \mathsf{fr}(\sigma)$.

The crucial polishing step is performed when we convert $\Phi_2(\bar{y})$ into Ψ_2 and $\exists \bar{z}_i \, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ into Ψ_3. The conversion of $\Phi_2(\bar{y})$ into Ψ_2 is done in the obvious way: item (i) is ensuring safety by removing inequalities, and item (ii) is performing the saturation of the formula by adding inequalities. We now elaborate on the sophisticated conversion of $\exists \bar{z}_i \, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ into Ψ_3. The two obvious polishing strategies that one might think of are:

Polishing Strategy I. For each $i \in [n]$, simply remove the conjunction of atoms $\exists \bar{z}_i \, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ as a whole if $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ mentions a variable that is neither in $\mathsf{var}(\Psi_1)$ nor in $\bar{z}_i$.

Polishing Strategy II. For each $i \in [n]$, remove only the atoms occurring in $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ that mention a variable that is neither in $\mathsf{var}(\Psi_1)$ nor in $\bar{z}_i$.

Although both strategies ensure safety, it is not difficult to show that they are both problematic. Loosely speaking, the first polishing strategy removes more than necessary from the negative part and this leads to an unsound result, whereas the second strategy removes less than necessary from the negative part, which leads to an incomplete result; this is illustrated via examples in the appendix (Section A). The way that we actually convert $\exists \bar{z}_i \, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ into Ψ_3 in the definition of (σ, τ)-resolvent somehow balances what we need to remove and what we need to keep in order ensure both the soundness and the completeness of the final result. The key idea underlying this conversion can be described as follows. Assume that Ξ is a subformula of $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ such that (i) all the variables in $\Phi_3^i(\bar{z}_i, \bar{w}_i) \setminus \Xi$ occur in $\mathsf{var}(\Psi_1)$ or $\bar{z}_i$, and (ii) Ξ is satisfied by the result of the chase due to some homomorphism h that maps Ξ to the set of atoms H obtained by a applying a trigger of the form $(\sigma, \cdot)$ during the construction of the chase. We then keep the conjunction consisting of the atoms of $\Phi_3^i(\bar{z}_i, \bar{w}_i) \setminus \Xi$ after updating their variables in such a way that the part of $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ that we keep cannot be mapped to H via the same homomorphism h used to map Ξ to H, which means that $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ as a whole is not satisfied by the result of the chase due to h.

The Algorithm NRewrite$^\neg$

We now have the key ingredients to discuss the nondeterministic algorithm $\mathsf{NRewrite}^\neg$, claimed in Proposition 7, which is depicted in Algorithm 1. It takes as input an omq $O = (\Sigma, Q = \{\bar{t}_{out} \mid \Xi\})$ from a class $(\mathsf{C}, \mathsf{CQ}^\neg)$ of omqs with the assumption that $(\mathsf{C}, \mathsf{CQ})$ is first-order rewritable, and rewrites O by proceeding in two phases: (i) the *preparatory phase*, where it converts Ξ into a set of formulas in rewrite form that are saturated by considering all the specialized versions of Ξ and properly rewriting their negative part, and (ii) the *rewriting phase*, where it exhaustively rewrites the formulas produced during the preparatory phase via resolution steps and constructs formulas in rewrite form that are saturated, which are stored in a set *Rew*. The final result is $\{\bar{t}_{out} \mid \bigvee_{\Phi \in Rew} \Phi\}$, which is, in general, an infinitary uccq$^=$. We proceed to discuss the details of the preparatory phase and the rewriting phase.

Algorithm 1 $\mathsf{NRewrite}^{\neg}(\Sigma, Q = \{\bar{t}_{out} \mid \Xi\})$.

Preparatory Phase
$\Pi := \{\pi \mid \pi \text{ is a proper partition of } \mathsf{terms}(Q)\}$
$\mathit{Spec} := \{\Phi_{Q,\pi} \mid \pi \in \Pi \text{ and the query } \{\bar{t}_{out} \mid \Phi_{Q,\pi}\} \text{ is satisfiable}\}$
if $\mathit{Spec} = \emptyset$ **then**
  return $\{() \mid \exists y_1 \cdots \exists y_n \, (P(y_1, \ldots, y_n) \wedge \neg P(y_1, \ldots, y_n))\}$

Rewriting Phase
$\mathit{Rew} := \{(\Phi, \epsilon) \mid \Phi \in \mathit{Spec}\}$
$i := 0$
repeat
  $\mathit{Temp} := \mathit{Rew}$
  forall $(\Phi, s) \in \mathit{Rew}$ **do**
    forall $\sigma \in \Sigma$ **do**
      $i := i + 1$
      forall valid mgcu τ of Φ with σ^i such that (σ^i, τ) is s-active **do**
        $\Phi' := (\sigma^i, \tau)$-resolvent of Φ
        if there is no $(\Psi, \cdot) \in \mathit{Rew}$ such that $\Psi \to \Phi'$ **then**
          $\mathit{Rew} := \mathit{Rew} \cup \{(\Phi', s \cdot (\sigma^i, \tau))\}$
until $\mathit{Temp} = \mathit{Rew}$;
return $\left\{\bar{t}_{out} \mid \bigvee_{(\Phi,\cdot) \in \mathit{Rew}} \Phi\right\}$

Algorithm 2 $\mathsf{AtomRewrite}(\Sigma, R(t_1, \ldots, t_n))$.

$\mathit{Col} := \{\{R(\rho(t_1), \ldots, \rho(t_n))\}\}$, where ρ renames variables to constants
$i := 0$
repeat
  $\mathit{Temp} := \mathit{Col}$
  forall $A \in \mathit{Col}$ **do**
    forall $\sigma \in \Sigma$ **do**
      $i := i - 1$
      forall mgcu $(A_1, \cdot, \theta)$ of A with σ that respects $\mathsf{var}(A_1) \cap \mathsf{var}(A \setminus A_1)$ **do**
        $A' := \theta((A \setminus A_1) \cup \mathsf{body}(\sigma^i))$
        if there is no $B \in \mathit{Col}$ such that $B \to A'$ **then**
          $\mathit{Col} := \mathit{Col} \cup \{A'\}$
until $\mathit{Temp} = \mathit{Rew}$;
return $\bigvee_{A \in \mathit{Col}} \bigwedge_{\alpha \in A} \rho^{-1}(\alpha)$

Preparatory Phase

Let $\preceq_{\text{lex}}$ be some arbitrary lexicographic order over $\mathbf{C} \cup \mathbf{V}$ such that (i) the constants are preceding the variables, and (ii) the variables of $\mathsf{fv}(\Xi)$ are preceding the variables of $\mathbf{V} \setminus \mathsf{fv}(\Xi)$. 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 $\pi = \{T_1, \ldots, T_m\}$ of the set of terms occurring in Q, denoted $\mathsf{terms}(Q)$, 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 $\mathsf{terms}(Q)$ 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 further define the function $\mu_\pi : \mathsf{terms}(Q) \to \mathsf{terms}(Q)$ in such a way that, for each $t \in \mathsf{terms}(Q)$, $t \in T_i$ implies $\mu_\pi(t) = [T_i]$. Now, for a proper partition $\pi = \{T_1, \ldots, T_m\}$ of $\mathsf{terms}(Q)$, the algorithm builds the formula $\Phi_{Q,\pi}$ defined as

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

where $\bar{x}_\pi$ are the variables occurring in $\bigwedge_{\alpha \in Q^+} \mu_\pi(\alpha) \wedge \bigwedge_{i \in [m], x \in T_i} x = [T_i]$, other than the variables of $\mathsf{fv}(\Xi)$, and $\bar{z}_\alpha$ are the variables occurring in $\mathsf{AtomRewrite}(\Sigma, \mu_\pi(\alpha))$ but not in $\bar{x}_\pi$. The procedure $\mathsf{AtomRewrite}$ (see Algorithm 2) with input $(\Sigma, \mu_\pi(\alpha))$ essentially rewrites the atom $\mu_\pi(\alpha)$, via resolution steps that use the tgds of Σ, into a disjunction of conjunctions of atoms. Crucially, the rewriting should not remove any of the variables in $\mu_\pi(\alpha)$ since, by definition, they also occur in the positive part of the formula $\Phi_{Q,\pi}$. Moreover, it should not unify any of those variables with a constant or some other variable since the inequalities in $\Phi_{Q,\pi}$ state that, for $i \neq j$, the variables $[T_i]$ and $[T_j]$ cannot be unified. In other words, $\mathsf{AtomRewrite}$ should treat the variables occurring in $\mu_\pi(\alpha)$ as constants, and thus, the final result would be a disjunction of conjunctions of atoms where each disjunct contains *all* the terms $t_1, \ldots, t_n$. The above intuition is implemented by

1. renaming all the variables in $\mu_\pi(\alpha)$ into new constants via a bijective function ρ (i.e., $\rho(t) = t$ if t is a constant, otherwise $\rho(t)$ is a new constant),
2. exhaustively applying resolution steps (using the basic notion of mgcu of a set of atoms that respects a set of variables) in the expected way starting from $\{\rho(\mu_\pi(\alpha))\}$, and
3. finally, converting the constructed collection of sets of atoms into a disjunction of conjunctions of atoms after replacing each constant $\rho(t)$ with t by applying ρ^{-1}.

Note that during a resolution step the procedure considers the 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 clatter among variables during a resolution step. Moreover, at each resolution step, the produced set of atoms A' is added to the collection *Col* only if there is no set of atoms $B \in \mathit{Col}$ such that B can be homomorphically mapped to A, i.e., A' is added to *Col* if there is no $B \in \mathit{Col}$ that is more general than A' since in this case A' will be redundant. The following lemma, which relies on the assumption that the input omq O belongs to a class of omqs $(\mathsf{C}, \mathsf{CQ}^\neg)$ such that $(\mathsf{C}, \mathsf{CQ})$ is first-order rewritable, is crucial in our development:

▶ **Lemma 8.** *For every proper partition π of $\mathsf{terms}(Q)$, $\{\bar{t}_{out} \mid \Phi_{Q,\pi}\}$ is a ccq$^=$.*

Now, the formula $\Phi_{Q,\pi}$ (in fact, the pair $(\Phi_{Q,\pi}, \epsilon)$ – this will be clarified in the discussion of the rewriting phase) is kept in the set *Spec* if the query $Q_\pi = \{\bar{t}_{out} \mid \Phi_{Q,\pi}\}$ is *satisfiable*, i.e., if there is a database D such that $Q_\pi(D) \neq \emptyset$. By Lemma 8, Q_π is a ccq$^=$. Hence, to be able to perform the above satisfiability check, we need a procedure that allows us to decide whether a query from $\mathsf{CCQ}^=$ is satisfiable. We can show the following result:

▶ **Lemma 9.** *Deciding whether a query from $\mathsf{CCQ}^=$ is satisfiable is coNP-complete.*

Note that the satisfiability procedure for ccqs with equality is also used in the rewriting phase of the algorithm. Now, if none of the queries Q_π, where π is a proper partition of $\mathsf{terms}(Q)$, is satisfiable (i.e., $\mathit{Spec} = \emptyset$), then we can show that $\mathsf{ans}_{\mathsf{Sk}}(O, D) = \emptyset$ for every database D. Hence, it suffices to return a simple unsatisfiable ncq over the underlying schema. In case *Spec* is not empty, then the algorithm proceeds with the rewriting phase in order to rewrite via resolution the formulas occurring in the pairs of the set *Spec*.

Rewriting Phase

In this phase, the algorithm exhaustively rewrites the positive part of each formula of *Spec* via resolution steps. During this process, it is important to keep track of the sequence of resolution steps that lead to a certain formula Φ. This is because, for ensuring the soundness of the algorithm, we need to guarantee that in the sequence of resolution steps that lead to a formula Φ, we use only active resolution steps, i.e., resolution steps that mimic different chase steps. To achieve this, we initially flag each formula Φ of *Spec* with the empty sequence of resolution steps, denoted ϵ, which will be updated accordingly during the rewriting process, and add the pair (Φ, ϵ) to *Rew*. Then, for each pair $(\Phi, s) \in \mathit{Rew}$ and for each tgd σ, the algorithm considers all the *valid* mgcus τ of Φ with σ^i such that (σ^i, τ) is *s-active*, and constructs the (σ^i, τ)-resolvent of Φ, denoted Φ'. Now, providing that Φ' is not *blocked* by some $\Psi \in \mathit{Rew}$, the pair $(\Phi', s \cdot (\sigma^i, \tau))$ is added to *Rew*, where $s \cdot (\sigma^i, \tau)$ is the sequence of resolution steps that lead to Φ', that is, the sequence of resolution steps that led to Φ plus the resolution step applied on Φ using the tgd σ^i and the mgcu τ of Φ with σ^i. We proceed to formalize the three auxiliary notions used above, namely validity, activeness, and blocking. Note that, as in AtomRewrite, at each step we consider the tgd σ^i obtained from σ by renaming each variable x in σ to x^i in order to avoid undesirable clatter among variables during a resolution step. Crucially, there is no clutter among the variables introduced during the rewriting phase and those introduced by AtomRewrite, since in the rewriting phase we use positive indices to rename variables, whereas in AtomRewrite we use negative indices.

Validity. The reason why we cannot use all the mgcus of Φ with σ^i, but only the so-called valid ones, is because invalid mgcus lead to unsound resolution steps. By applying an invalid mgcu, we may produce a formula that gives rise to a satisfiable ccq$^=$ from a formula that defines an unsatisfiable ccq$^=$, which is problematic; this is shown in the appendix (Section B). Validity ensures that the resolution of a formula that defines an unsatisfiable ccq$^=$ will produce a formula that defines an unsatisfiable ccq$^=$. An mgcu $\tau = (\cdot, \cdot, \theta, \theta')$ of

$$\Phi \;=\; \exists \bar{v} \left(\Phi_1(\bar{x}) \wedge \Phi_2(\bar{y}) \wedge \neg \left(\bigvee_{i=1}^{n} \exists \bar{z}_i \left(\Phi_3^i(\bar{z}_i, \bar{w}_i) \right) \right) \right),$$

produced either during the preparatory phase or the rewriting phase of the algorithm, with σ^i is *valid* if the ccq$^=$ $\{\bar{t}_{out} \mid \exists \bar{u}\, \Psi\}$, where $\bar{u} = \mathsf{fv}(\Psi) \setminus \mathsf{fv}(\Phi)$ and Ψ is the formula

$$\bigwedge_{\alpha \in \mathsf{head}(\sigma^i)} \lambda(\alpha) \;\wedge\; \Phi_1(\bar{x}) \;\wedge\; \Phi_2(\bar{y}) \;\wedge\; \neg \left(\bigvee_{i=1}^{n} \exists \bar{z}_i \left(\Phi_3^i(\bar{z}_i, \bar{w}_i) \right) \right),$$

for $\lambda = \theta \cup \theta'$, is satisfiable. Note that Ψ is in rewrite form, but it might not be saturated. This is not problematic since we do not apply a resolution step on Ψ, but we only check whether it gives rise to a satisfiable ccq$^=$; the latter is done using the algorithm from Lemma 9.

Activeness. We now discuss the notion of activeness. Consider a formula-sequence pair (Φ, s), where $s = ((\sigma_1^{i_1}, \tau_1), \ldots, (\sigma_k^{i_k}, \tau_k))$, for $k > 0$ and $\tau_i = (\cdot, \cdot, \theta_i, \theta_i')$, which is produced during the rewriting phase of the algorithm and stored in *Rew*, and a mgcu $\tau = (\cdot, \cdot, \theta, \theta')$ of Φ with σ^i. As said above, the goal of activeness is to ensure that the resolution step that uses the pair (σ^i, τ) mimics a different chase step than the resolution steps that use a pair occurring in s. This is guaranteed by the following condition: we say that (σ^i, τ) is *s-active* if, for every $j \in [k]$, $\sigma = \sigma_j$ implies $\lambda(\mathsf{body}(\sigma^i)) \neq \lambda_j(\mathsf{body}(\sigma_j^{i_j}))$ for $\lambda = \theta \cup \theta'$ and $\lambda_j = \theta_j \cup \theta_j'$.

Blocking. It is crucial for the termination of the algorithm (see item (2) of Proposition 7) not to add a formula in *Rew* that gives rise to a ccq$^=$ that is redundant in the sense that its answer, no matter the input database, is always contained in the answer of a ccq$^=$ defined by a formula that is already in *Rew*. Having said that, we can show that the containment problem for ccqs with equality is undecidable, which is a fact of independent interest. The latter essentially tells us that we need to isolate a condition over ccqs with equality that is sufficient for containment, can be effectively checked, and it ensures the termination of the algorithm whenever the input omq obeys linearity or acyclicity, despite the fact that some formulas that give rise to redundant queries will be unavoidably added to the set *Rew*. This is the intention underlying blocking. Consider

$$\Phi \;=\; \exists \bar{v} \left(\Phi_1(\bar{x}) \wedge \Phi_2(\bar{y}) \wedge \neg \left(\bigvee_{i=1}^{n} \exists \bar{z}_i \left(\Phi_3^i(\bar{z}_i, \bar{w}_i) \right) \right) \right)$$

and

$$\Psi \;=\; \exists \bar{v}' \left(\Psi_1(\bar{x}') \wedge \Psi_2(\bar{y}') \wedge \neg \left(\bigvee_{i=1}^{m} \exists \bar{z}_i \left(\Psi_3^i(\bar{z}_i', \bar{w}_i') \right) \right) \right)$$

that are produced during the rewriting phase of the algorithm. We say that Ψ *blocks* Φ, denoted $\Psi \to \Phi$, if the following four conditions hold: (i) $\mathsf{rel}(\Psi_1(\bar{x}')) \to \mathsf{rel}(\Phi_1(\bar{x}))$ via a homomorphism h that is the identity over $\mathsf{fv}(\Psi)$, (ii) for each equality $t = t'$ in $\mathsf{eq}(\Psi_1(\bar{y}'))$, where either $t, t' \in \mathsf{fv}(\Psi)$ or $t \in \mathsf{fv}(\Psi)$ and $t' \in \mathsf{dom}(\Psi)$, the equality $t = t'$ occurs in $\mathsf{eq}(\Phi_1(\bar{y}))$, (iii) for each inequality $t \neq t'$ in $\Psi_2(\bar{y}')$, the inequality $h(t) \neq h(t')$ occurs in $\Phi_2(\bar{y})$, and (iv) for each $i \in [m]$, there exists $j \in [n]$ such that $\Psi_3^i(\bar{z}_i', h(\bar{w}_i')) = \Phi_3^j(\bar{z}_j, \bar{w}_j)$. This completes the description of $\mathsf{NRewrite}^\neg$, which is indeed nondeterministic since, in both of its phases, there is no fixed order in which we apply the resolution steps.

4.2 Correctness and Termination of $\mathsf{NRewrite}^\neg$

Correctness – Item (1) of Proposition 7

We first argue that every execution of the algorithm leads to an infinitary uccq$^=$. Let $\mathit{exec}(O)$ be an arbitrary execution of $\mathsf{NRewrite}^\neg(O)$. We proceed by considering the two cases where the set *Spec* computed during the preparatory phase of the algorithm is empty or not. If $\mathit{Spec} = \emptyset$, then $\mathit{exec}(O)$ terminates and returns an unsatisfiable ncq, which is trivially an infinitary uccq$^=$. Assume now that $\mathit{Spec} \neq \emptyset$. We proceed to argue that it consists of finitely many formulas of finite size. By Lemma 8, we get that, for every partition π of $\mathsf{terms}(Q)$, $\{\bar{t}_{out} \mid \Phi_{Q,\pi}\}$ is a ccq$^=$. This implies that $\Phi_{Q,\pi}$ is a formula in rewrite form of *finite* size. Moreover, since $\mathsf{terms}(Q)$ is finite, it is clear that there are finitely many different proper partitions of it. Consequently, *Spec* consists of finitely many formulas (one for each proper partition of $\mathsf{terms}(Q)$) in rewrite form, each of finite size. With this fact in mind, to conclude

that $\mathit{exec}(O)$ leads to an infinitary uccq$^{=}$ it suffices to observe that, for every pair of the form $(\Phi, \cdot)$ occurring in the set Rew, and every valid mgcu τ of Φ with σ^i, for some $i > 0$, the (σ^i, τ)-resolvent Φ' of Φ is, by construction, a formula in rewrite form of finite size with $\mathsf{fv}(\Phi') = \bar{t}_{out}$. Thus, for every pair $(\Phi, \cdot)$ that occurs in the (possibly infinite) set Rew computed by the rewriting phase of the algorithm, it holds that Φ is a formula in rewrite form of finite size with $\mathsf{fv}(\Phi') = \bar{t}_{out}$. This in turn implies that $\left\{\bar{t}_{out} \mid \bigvee_{(\Phi,\cdot)\in \mathit{Rew}} \Phi\right\}$, which we denote by Q_O, is indeed an infinitary uccq$^{=}$, as needed.

We finally need to argue that, for every database D, $\mathsf{ans}_{\mathsf{Sk}}(O, D) = Q_O(D)$; recall that $O = (\Sigma, Q)$. To this end, we establish the following lemma, which, together with Proposition 1, implies the claim. Given a formula Φ in rewrite form with $\mathsf{fv}(\Phi) = \bar{t}_{out}$, let $Q_\Phi = \{\bar{t}_{out} \mid \Phi\}$.

▶ **Lemma 10.** *Fix an arbitrary tuple $\bar{c} \in \mathsf{dom}(D)^{|\bar{t}_{out}|}$. There exists a chase derivation δ of D w.r.t. Σ such that $\bar{c} \in Q(\mathsf{chase}_\delta(D, \Sigma))$ iff there exists $(\Phi, \cdot) \in \mathit{Rew}$ such that $\bar{c} \in Q_\Phi(D)$.*

Further details about the proof of Lemma 10 are given in the appendix (Section C).

Termination – Item (2) of Proposition 7

Fix an arbitrary execution $\mathit{exec}(O)$ of $\mathsf{NRewrite}^{\neg}(O)$. By linearity and acyclicity, we get a database-independent bound on the number of atoms that can appear in the positive part of a formula produced by $\mathit{exec}(O)$. This in turn allows us to show that the formulas produced by $\mathit{exec}(O)$ share finitely many positive parts (up to renaming of the non-free variables). Note, however, that this is not enough to conclude that the output of $\mathit{exec}(O)$ is finite since we may have several different formulas with the same positive part but different negative part. Interestingly, we can argue that the way the polishing of formulas is performed (see the discussion above about resolvents), ensures that only finitely many different negative parts (up to renaming of the non-free variables) can be produced, and thus, $\mathit{exec}(O)$ is finite.

5 Future Work

Although we are still far from an algorithmic characterization of first-order rewritability for non-positive classes of tgd-based omqs, where the user query is a cq with negation, we believe that the algorithm $\mathsf{NRewrite}^{\neg}$ introduced in this work is a good basis towards such a characterization. In particular, since item (1) of Proposition 7 holds for every class C of tgds such that $(\mathsf{C}, \mathsf{CQ}^{\neg})$ is first-order rewritable, the next obvious step is to try to generalize item (2) of Proposition 7 to every class C of tgds such that $(\mathsf{C}, \mathsf{CQ}^{\neg})$ is first-order rewritable. This highly non-trivial task is our next target, which will establish the desired characterization.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Mario Alviano, Michael Morak, and Andreas Pieris. Stable model semantics for tuple-generating dependencies revisited. In *PODS*, pages 377–388, 2017. doi:10.1145/3034786.3034794.

3 Mario Alviano and Andreas Pieris. Default negation for non-guarded existential rules. In *PODS*, pages 79–90, 2015. doi:10.1145/2745754.2745758.

4 Jean-François Baget, Fabien Garreau, Marie-Laure Mugnier, and Swan Rocher. Extending acyclicity notions for existential rules. In *ECAI*, pages 39–44, 2014. doi:10.3233/978-1-61499-419-0-39.

5 Gerald Berger, Georg Gottlob, Andreas Pieris, and Emanuel Sallinger. The space-efficient core of vadalog. *ACM Trans. Database Syst.*, 47(1):1:1–1:46, 2022. doi:10.1145/3488720.

6 Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. Ontology-based data access: A study through disjunctive datalog, CSP, and MMSNP. *ACM Trans. Database Syst.*, 39(4):33:1–33:44, 2014. doi:10.1145/2661643.

7 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.*, 14:57–83, 2012. doi:10.1016/J.WEBSEM.2012.03.001.

8 Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012. doi:10.1016/J.ARTINT.2012.08.002.

9 Georg Gottlob, André Hernich, Clemens Kupke, and Thomas Lukasiewicz. Stable model semantics for guarded existential rules and description logics: Decidability and complexity. *J. ACM*, 68(5):35:1–35:87, 2021. doi:10.1145/3447508.

10 Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Query rewriting and optimization for ontological databases. *ACM Trans. Database Syst.*, 39(3):25:1–25:46, 2014. doi:10.1145/2638546.

11 André Hernich, Clemens Kupke, Thomas Lukasiewicz, and Georg Gottlob. Well-founded semantics for extended datalog and ontological reasoning. In *PODS*, pages 225–236, 2013. doi:10.1145/2463664.2465229.

12 Mélanie König, Michel Leclère, Marie-Laure Mugnier, and Michaël Thomazo. Sound, complete and minimal ucq-rewriting for existential rules. *Semantic Web*, 6(5):451–475, 2015. doi:10.3233/SW-140153.

13 Thomas Lukasiewicz, Enrico Malizia, Maria Vanina Martinez, Cristian Molinaro, Andreas Pieris, and Gerardo I. Simari. Inconsistency-tolerant query answering for existential rules. *Artif. Intell.*, 307:103685, 2022. doi:10.1016/J.ARTINT.2022.103685.

14 Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008. doi:10.1007/978-3-540-77688-8_5.

A Resolvents and Polishing Strategies

As discussed in the main body of the paper, the two immediate polishing strategies for the conversion of $\exists \bar{z}_i\, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ into a safe formula in rewrite form are:

Polishing Strategy I. For each $i \in [n]$, simply remove the conjunction of atoms $\exists \bar{z}_i\, \Phi_3^i(\bar{z}_i, \bar{w}_i)$ as a whole if $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ mentions a variable that is neither in $\mathsf{var}(\Psi_1)$ nor in $\bar{z}_i$.

Polishing Strategy II. For each $i \in [n]$, remove only the atoms occurring in $\Phi_3^i(\bar{z}_i, \bar{w}_i)$ that mention a variable that is neither in $\mathsf{var}(\Psi_1)$ nor in $\bar{z}_i$.

It is clear that both strategies will ensure safety. It is not difficult, however, to illustrate that both strategies are problematic. Let us start with the first polishing strategy.

▶ **Example 11.** Consider the sentence in rewrite form (that is saturated) and the tgd

$$\Phi \;=\; \exists x\, (P(x) \wedge \neg\, (\exists z\, (T(x) \wedge U(z)))) \qquad \text{and} \qquad \sigma \;=\; R(x') \;\rightarrow\; \exists y'\, (P(y'), T(y')),$$

respectively. The only available mgcu of Φ with σ is

$$\tau \;=\; (\{P(x)\}, \{P(y')\}, \{x \mapsto x, y' \mapsto x\}, \{x' \mapsto x'\})\,.$$

Hence, by resolving $\{P(x)\}$ using σ we get the sentence

$$\Psi \;=\; \exists x'\, (R(x') \wedge \neg\, (\exists z\, (T(x) \wedge U(z))))\,,$$

which needs polishing since the free variable x occurs in the negative part but not in the positive part. According to the first strategy, we remove the negative part as a whole and get

$$\widehat{\Psi} \;=\; \exists x'\, R(x').$$

It is easy to see that the obtained sentence is unsound in the following sense: there is a database D such that the result of the chase of D w.r.t. $\{\sigma\}$ does not satisfy the original sentence Φ, but D satisfies the obtained sentence $\widehat{\Psi}$; in particular, for $D = \{R(a), U(a)\}$. ⌟

Loosely speaking, the first polishing strategy removes more than necessary from the negative part and this leads to an unsound result. Let us now discuss the second strategy.

▶ **Example 12.** Consider the sentence in rewrite form (that is saturated) and the tgd

$$\Phi \;=\; \exists x\, (P(x) \wedge \neg\, (\exists z\, (T(x) \wedge U(z)))) \qquad \text{and} \qquad \sigma \;=\; R(x') \;\rightarrow\; \exists y'\, P(y'),$$

respectively. The only available mgcu of Φ with σ is

$$\tau \;=\; (\{P(x)\}, \{P(y')\}, \{x \mapsto x, y' \mapsto x\}, \{x' \mapsto x'\})\,.$$

Hence, by resolving $\{P(x)\}$ using σ we get the sentence

$$\Psi \;=\; \exists x'\, (R(x') \wedge \neg\, (\exists z\, (T(x) \wedge U(z))))\,,$$

which needs polishing since the free variable x occurs in the negative part but not in the positive part. According to the second strategy, we remove $T(x)$ from the negative part, which is the only atom that mentions x, and get the sentence

$$\widehat{\Psi} \;=\; \exists x'\, (R(x') \wedge \neg\, (\exists z\, U(z)))\,.$$

It is easy to see that the obtained sentence is incomplete in the following sense: there is a database D such that the result of the chase of D w.r.t. $\{\sigma\}$ satisfies the original sentence Φ, but D does not satisfy the obtained sentence $\widehat{\Psi}$; in particular, for $D = \{R(a), U(a)\}$. ⌟

Roughly speaking, the second strategy removes less than necessary from the negative part, which leads to an incomplete result. Coming back to Examples 11 and 12, our correct polishing strategy will give the sentences

$$\exists x'\, (R(x') \wedge \neg\, (\exists z\, U(z))) \qquad \text{and} \qquad \exists x'\, R(x'),$$

respectively, which both correspond to sound and complete rewriting steps.

B Validity

As discussed in the main body of the paper, the reason why we cannot use all the mgcus of Φ with σ^i, but only valid ones, is because invalid mgcus lead to unsound resolution steps. More precisely, by applying an invalid mgcu, we may produce a formula that gives rise to a satisfiable ccq$^=$ from a formula that defines an unsatisfiable ccq$^=$, which is problematic.

▶ **Example 13.** Consider the sentence in rewrite form (that is saturated) and the tgd

$$\Phi \;=\; \exists x\, (P(x) \wedge \neg T(x)) \qquad \text{and} \qquad \sigma \;=\; R(y) \;\rightarrow\; \exists z\, (P(z), T(z))\,,$$

respectively. It is clear that, for every null instance I that (i) contains an atom $R(c)$, (ii) $I \models \sigma$, and (iii) $I \models P(x) \rightarrow T(x)$, it holds that $I \not\models \Phi$, i.e., the query $\{() \mid \Phi\}$ is unsatisfiable w.r.t. instances with the properties (i) - (iii). Now, observe that the only available mgcu of Φ

with σ is $\tau = (\{P(x)\}, \{P(z)\}, \{x \mapsto x, z \mapsto x\}, \{y \mapsto y\})$ and the (σ, τ)-resolvent of Φ is the sentence $\Psi = \exists y\, R(y)$, and thus, the query $\{() \mid \Psi\}$ has a non-empty answer over every null instance that contains an atom $R(c)$. Hence, given the database $D = \{R(c)\}$, the answer to the query $\{() \mid \Phi\}$ over the result of the chase of D w.r.t. $\{\sigma\}$, which is an instance that enjoys the properties (i) - (iii), is empty, whereas the answer to $\{() \mid \Psi\}$ over D is non-empty. Summing up, the mgcu τ is not valid since the sentence $\exists x\,(P(x) \wedge T(x) \wedge \neg T(x))$, which is essentially the conjunction of the head of σ (after updating its variables according to the functions of τ) with Φ, gives rise to an unsatisfiable query. ⌟

C Proof of Lemma 10

The proof of Lemma 10 relies on four useful auxiliary technical lemmas. We proceed to state those lemmas and then we proceed with the final proof.

Four Auxiliary Lemmas

Recall that $O = (\Sigma, Q)$. Moreover, given a formula Φ in rewrite form with $\mathsf{fv}(\Phi) = \bar{t}_{out}$, we write Q_Φ for the query $\{\bar{t}_{out} \mid \Phi\}$. The first auxiliary lemma shows that the rewriting of the negative atoms, performed by the procedure AtomRewrite, is sound.

▶ **Lemma 14.** *Let δ be a chase derivation of D w.r.t Σ. The following are equivalent:*
1. $\bar{c} \in Q(\mathsf{chase}_\delta(D, \Sigma))$.
2. *There exists $\Psi \in Spec$ such that $\bar{c} \in Q_\Psi(\mathsf{chase}_\delta(D, \Sigma))$.*

The next lemma is crucial for establishing the completeness of NRewrite$^\neg$, i.e., the direction ($\Rightarrow$) of Lemma 10. It states that from a prefix p of a chase derivation of D w.r.t. Σ, where its last instance I is such that $\bar{c} \in Q_\Phi(I)$ with Φ being a formula in rewrite form that is saturated and $\mathsf{fv}(\Phi) = \bar{t}_{out}$, we can extract a sequence of resolution steps starting from Φ, and mimicking backwards the chase applications that gave rise to p, that leads to a formula Φ' in rewrite form that is saturated with $\mathsf{fv}(\Phi') = \bar{t}_{out}$ such that $\bar{c} \in Q_{\Phi'}(D)$. This is shown by induction on the length of the prefix p.

▶ **Lemma 15.** *Consider a formula Φ in rewrite form that is saturated with $\mathsf{fv}(\Phi) = \bar{t}_{out}$. Let $(I_i)_{i \in \{0,\ldots,m\}}$, for $m \geq 0$, be a prefix of a chase derivation of D w.r.t. Σ with $I_i \langle \sigma_i, h_i \rangle I_{i+1}$, for $i \in \{0, \ldots, m-1\}$, such that $\bar{c} \in Q_\Phi(I_m)$. There is a sequence $(\Psi_i)_{i \in \{0,\ldots,m\}}$ of formulas in rewrite form that are saturated with $\mathsf{fv}(\Psi_i) = \bar{t}_{out}$, for each $i \in \{0, \ldots, m\}$, such that:*
1. $\Psi_0 = \Phi$
2. $\Psi_i = \Psi_{i-1}$ *or* Ψ_i *is a* (σ^i_{m-i}, τ)*-resolvent of* Ψ_{i-1} *for some valid mgcu* τ *of* Ψ_{i-1} *with* σ^i_{m-i}*, for each* $i \in [m]$*, and*
3. $\bar{c} \in Q_{\Psi_i}(I_{m-i})$*, for each* $i \in \{0, \ldots, m\}$.

The next lemma is also needed for establishing the completeness of NRewrite$^\neg$, i.e., the direction ($\Rightarrow$) of Lemma 10. Intuitively, it states that if a formula Φ in rewrite form is blocked by some formula of *Rew* produced by the algorithm, then all the rewritten formulas obtained via resolution steps from Φ will be blocked by some other formula of *Rew*.

▶ **Lemma 16.** *Consider a set Θ of formulas in rewrite form, and a formula Φ in rewrite form such that Φ is Θ-blocked. Let $\tau = (S, S', \theta, \theta')$ be a chunk unifier of Φ with a tgd $\sigma \in \Sigma$ such that the (σ, τ)-resolvent Ψ of Φ is satisfiable. One of the following holds:*
1. *Ψ is Θ-blocked, or*
2. *Ψ is $\{\Psi'\}$-blocked, where Ψ' is the (σ, τ')-resolvent of some $\Phi' \in \Theta$, where $\tau' = (\cdot, \cdot, \theta_1, \theta'_1)$ is a chunk unifier of Φ' with σ such that $\lambda(\mathsf{body}(\sigma)) = \lambda_1(\mathsf{body}(\sigma))$, where $\lambda = \theta \cup \theta'$ and $\lambda_1 = \theta_1 \cup \theta'_1$.*

The last lemma is crucial for the direction ($\Leftarrow$) of Lemma 10, that is, the soundness of $\mathsf{NRewrite}^{\neg}$. A *rewriting* Σ*-sequence* is a sequence of pairs $((\Phi_i, s_i))_{i\in\{0,\ldots,m\}}$, for $m \geq 0$, where $\Phi_0 \in \mathit{Spec}$, $s_0 = \epsilon$, and, for each $i \in \{0,\ldots,m-1\}$, Φ_{i+1} is a (σ^{i+1}, τ)-resolvent of Φ_i, where $\sigma \in \Sigma$, for some valid mgcu τ of Φ_i with σ^{i+1} such that (σ^{i+1}, τ) is s_i-active, and $s_{i+1} = s_i \cdot (\sigma^{i+1}, \tau)$. The lemma in question essentially states that, for a rewriting Σ-sequence $r = ((\Phi_i, s_i))_{i\in\{0,\ldots,m\}}$, $\bar{c} \in Q_{\Phi_m}(D)$ implies $\bar{c} \in Q_{\Phi_0}(I)$, where I is an null instance obtained by some chase derivation of D w.r.t. Σ. This is shown by induction on the length of r.

▶ **Lemma 17.** *Consider a rewriting Σ-sequence $((\Psi_i, s_i))_{i\in\{0,\ldots,m\}}$. There exists a prefix of a chase derivation $(I_i)_{i\in\{0,\ldots,m\}}$ of D w.r.t. Σ such that $\bar{c} \in Q_{\Psi_m}(I_0)$ implies $\bar{c} \in Q_{\Psi_0}(I_m)$.*

The Final Proof

We are now ready to establish Lemma 10.

($\Rightarrow$) By hypothesis, there exists a chase derivation δ of D w.r.t. Σ such that $\bar{c} \in Q(\mathsf{chase}_\delta(D, \Sigma))$. By Lemma 14, there exists $\Phi \in \mathit{Spec}$ such that $\bar{c} \in Q_\Phi(\mathsf{chase}_\delta(D, \Sigma))$. Therefore, there exists a prefix $(I_i)_{i\in\{0,\ldots,m\}}$, for $m \geq 0$, of δ such that $I_m \models \Phi$. By Lemma 15, there is a sequence $s = (\Psi_i)_{i\in\{0,\ldots,m\}}$, for $m \geq 0$, with the properties stated in the statement of the lemma. Note that there is no guarantee that all the elements in s are produced by the execution of $\mathsf{NRewrite}^{\neg}(O)$ under consideration. This is due on the one hand to the activeness condition of the triggers considered and on the other to the blocking condition that must not be satisfied to maintain a formula produced through a resolution phase. As regards the activeness condition of the triggers, it is easy to prove that the use of inactive triggers cannot be put in one-to-one correspondence with a prefix p of a chase derivation, in fact the same trigger cannot be used more than once by definition. For what concerns the blocking condition, by induction on the length of s, we show that Ψ_m is Θ-blocked, where Θ is the set that collects all the disjuncts of Q_O:

Base Step. If s is of length 1, then $s = \Psi_0$. Therefore, Ψ_0 is $\{\Phi\}$-blocked, which in turn implies that Ψ_0 is Θ-blocked since $\Phi \in \Theta$, and the claim follows.

Inductive Step. Suppose now that s is of length $m+1$, i.e., $s = \Psi_0, \ldots, \Psi_m$. By induction hypothesis, Ψ_{m-1} is Θ-blocked. If $\Psi_m = \Psi_{m-1}$, then we immediately get that Ψ_m is Θ-blocked. Assume now that Ψ_m is the (σ, τ)-resolvent of Ψ_{m-1}, where τ is a valid mgcu of Ψ_{m-1} with σ. By Lemma 16, we get that Ψ_m is Θ-blocked or Ψ_m is $\{\Psi'\}$-blocked, where Ψ' is the (σ, τ')-resolvent of some $\Phi' \in \Theta$. Clearly, Ψ' is Θ-blocked, and thus, Ψ_m is Θ-blocked.

Since $\bar{c} \in Q_{\Psi_m}(D)$ and Φ_m is Θ-blocked, $\bar{c} \in Q_\Phi(D)$ for some disjunct Φ of Q_O.

($\Leftarrow$) By hypothesis, there exists $(\tilde{\Phi}, \cdot) \in \mathit{Rew}$ such that $\bar{c} \in Q_{\tilde{\Phi}}(D)$. This implies that there is a rewriting Σ-sequence $((\Psi_i, s_i))_{i\in\{0,\ldots,m\}}$, with $\Psi_m = \tilde{\Phi}$. By Lemma 17, there exists a prefix of a chase derivation $(I_i)_{i\in\{0,\ldots,m\}}$ of D w.r.t. Σ such that, for $i \in \{0,\ldots,m\}$, $\bar{c} \in Q_{\Psi_i}(I_{m-i})$ implies $\bar{c} \in Q_{\Psi_0}(I_m)$. Therefore, since $D = I_0$, we get that $\bar{c} \in Q_{\Psi_m}(I_0)$, and thus, $\bar{c} \in Q_{\Psi_0}(I_m)$. Since $\Psi_0 \in \mathit{Spec}$, we can establish the following monotonicity property: for a chase derivation $\delta = (J_i)_{i\geq 0}$ such that $I_i = J_i$ for each $i \in \{0,\ldots,m\}$, it holds that $\bar{c} \in Q_{\Psi_0}(\mathsf{chase}_\delta(D, \Sigma))$. Thus, there is a chase derivation δ of D w.r.t. Σ such that $\bar{c} \in Q_{\Psi_0}(\mathsf{chase}_\delta(D, \Sigma))$. Since $\Psi_0 \in \mathit{Spec}$, and $\bar{c} \in Q_{\Psi_0}(\mathsf{chase}_\delta(D, \Sigma))$, Lemma 14 implies that $\bar{c} \in Q(\mathsf{chase}_\delta(D, \Sigma))$, and the claim follows.