Containment for Rule-Based Ontology-Mediated Queries

Pablo Barceló
MI Foundational Research on Data & DCC
University of Chile, Chile
pbarcelo@dcc.uchile.cl

Gerald Berger
Institute for Logic and Computation
TU Wien, Austria
gberger@dbai.tuwien.ac.at

Andreas Pieris
School of Informatics
University of Edinburgh, UK
apieris@inf.ed.ac.uk

ABSTRACT

Many efforts have been dedicated to identifying restrictions on ontologies expressed as tuple-generating dependencies (tgds), a.k.a. existential rules, that lead to the decidability of answering ontology-mediated queries (OMQs). This has given rise to three families of formalisms: guarded, non-recursive, and sticky sets of tgds. We study the containment problem for OMQs expressed in such formalisms, which is a key ingredient for solving static analysis tasks associated with them. Our main contribution is the development of specially tailored techniques for OMQ containment under the classes of tgds stated above. This enables us to obtain sharp complexity bounds for the problems at hand.

ACM Reference Format:
Pablo Barceló, Gerald Berger, and Andreas Pieris. 2018. Containment for Rule-Based Ontology-Mediated Queries. In *PODS'18: 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3196959.3196963

1 INTRODUCTION

Motivation and goals. The novel application of knowledge representation tools for handling incomplete and heterogeneous data is giving rise to a new field, recently coined as *knowledge-enriched data management* [1]. A crucial problem in this field is *ontology-based data access* (OBDA) [32], which refers to the utilization of ontologies (i.e., sets of logical sentences) for providing a unified conceptual view of various data sources. Users can then pose their queries solely in the schema provided by the ontology, abstracting away from the specifics of the individual sources. In OBDA, one interprets the ontology Σ and the user query q, which is typically a *union of conjunctive queries* (UCQ), as two components of one composite query $Q = (\mathbf{S}, \Sigma, q)$, known as *ontology-mediated query* (OMQ); $\mathbf{S}$ is called the *data schema*, indicating that Q will be posed on databases over $\mathbf{S}$ [12]. Therefore, OBDA is often realized as the problem of answering OMQs.

While in this setting *description logics* (DLs) are often used for modeling ontologies, it is widely accepted that for handling arbitrary arity relations in relational databases it is convenient to use *tuple-generating dependencies* (tgds), a.k.a. *existential rules* or *Datalog$^{\pm}$ rules*; cf. [23]. Several aspects of OMQs in which the ontology is a set of tgds and the actual query is a UCQ (simply called OMQs from now on) have been studied in the data management literature; most notably (a) *query evaluation* [2, 15–17], i.e., given an OMQ $Q = (\mathbf{S}, \Sigma, q)$, a database D over $\mathbf{S}$, and a tuple of constants $\bar{c}$, does $\bar{c}$ belong to the evaluation of q over every extension of D that satisfies Σ, or, equivalently, is $\bar{c}$ a *certain answer* for Q over D? and (b) *relative expressiveness* [12, 25, 26]: how does the expressiveness of OMQs compare to the one of other query languages?

This work focuses on another crucial task for OMQs; namely, *containment*: for two OMQs Q_1 and Q_2 with data schema $\mathbf{S}$, does $Q_1(D) \subseteq Q_2(D)$ hold for every (finite) database D over $\mathbf{S}$ (where $Q(D)$ denotes the certain answers for Q over D)? Apart from the traditional applications of containment, such as query optimization or view-based query answering, it has been recently shown that OMQ containment has applications on other important static analysis tasks, namely, distribution over components [9], and UCQ rewritability [10]. Despite its prominence, no work to date has carried out an in-depth investigation of containment for OMQs based on tgds. When considered in its full generality, the OMQ containment problem is undecidable. In order to understand which restrictions on the tgds lead to decidability, we recall the two main reasons that render the general containment problem undecidable:

Undecidability of query evaluation: OMQ evaluation is, in general, undecidable [6], and it can be reduced to OMQ containment. More precisely, OMQ containment is undecidable whenever query evaluation for at least one of the involved languages (i.e., the language of the left-hand or the right-hand side query) is undecidable.

Undecidability of containment for Datalog: decidability of query evaluation does not ensure decidability of query containment. A prime example is Datalog, i.e., the OMQ language based on *full* tgds. Datalog containment is undecidable [34]; thus, OMQ containment is undecidable if the involved languages extend Datalog.

In view of the above observations, we focus on languages that have a decidable query evaluation, and do not extend Datalog. The main classes of tgds, which give rise to OMQ languages with the desirable properties, can be classified into three main families depending on the underlying restrictions: (i) *(frontier-)guarded* tgds [2, 15], which contain inclusion dependencies and linear tgds, (ii) *non-recursive* sets of tgds [22], and (iii) *sticky* sets of tgds [17].

While the decidability of containment for the above OMQ languages can be established via translations into query languages with a decidable containment problem, such translations do not lead to optimal complexity upper bounds (details are given below). Thus,

PODS'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196963

	Arbitrary Arity	Bounded Arity
Linear	PSPACE-c PSPACE-c	Π_2^P-c NP-c
Sticky	coNEXPTIME-c EXPTIME-c	Π_2^P-c NP-c
Non-recursive	in coNEXPTIME^{NP} and P^{NEXP}-hard NEXPTIME-c	in coNEXPTIME^{NP} and P^{NEXP}-hard NEXPTIME-c
Guarded	2EXPTIME-c 2EXPTIME-c	2EXPTIME-c EXPTIME-c
Frontier-guarded	2EXPTIME-c 2EXPTIME-c	2EXPTIME-c 2EXPTIME-c

Table 1: Complexity of OMQ containment – in small fonts, we recall the complexity of OMQ evaluation.

the main goal of this work is to develop specially tailored decision procedures for the containment problem under the OMQ languages in question, and, ideally, obtain precise complexity bounds.

Our contributions. The complexity of OMQ containment for the languages in question is given in Table 1. Using small fonts, we recall the complexity of OMQ evaluation in order to stress that containment is, in general, harder than evaluation. We structure our contributions as follows:

Linear, non-recursive and sticky sets of tgds. The OMQ languages based on linear, non-recursive, and sticky sets of tgds share a useful property: they are *UCQ rewritable* (implicit in [23]), that is, an OMQ can be rewritten into a UCQ. This property immediately yields decidability for their associated containment problems, since UCQ containment is decidable [33]. However, the obtained complexity bounds are not optimal, since the UCQ rewritings are unavoidably very large [23]. To obtain more precise bounds, we reduce containment to query evaluation, an idea that is often applied in query containment; see, e.g., [18, 19, 33].

Consider a UCQ rewritable OMQ language $\mathbb{O}$. If Q_1 and Q_2 belong to $\mathbb{O}$, both with data schema $\mathbf{S}$, then we can establish a *small witness property*, which states that non-containment of Q_1 in Q_2 can be witnessed via a database over $\mathbf{S}$ whose size is bounded by an integer $k \geq 0$, the maximal size of a disjunct in a UCQ rewriting of Q_1. For linear tgds, such an integer k is polynomial, but for non-recursive and sticky sets of tgds it is exponential (implicit in [23]). The above small witness property allows us to devise a simple non-deterministic algorithm, which makes use of query evaluation as a subroutine for checking non-containment of Q_1 in Q_2: guess a database D over $\mathbf{S}$ of size at most k, and then check if there is a certain answer for Q_1 over D that is not a certain answer for Q_2 over D. This leads to an optimal upper bounds for OMQs based on linear and sticky sets of tgds; however, the exact complexity of OMQs based on non-recursive sets of tgds remains open:

- For OMQs based on linear tgds, the containment problem is in PSPACE, and in Π_2^P if the arity is fixed. The PSPACE-hardness is shown by reduction from query evaluation, while the Π_2^P-hardness is implicit in [11].
- For OMQs based on sticky sets of tgds, the problem is in coNEXPTIME, and in Π_2^P if the arity of the schema is fixed. The coNEXPTIME-hardness is shown by exploiting the standard tiling problem for the exponential grid, while the Π_2^P-hardness is inherited from [11].
- Finally, for OMQs based on non-recursive sets of tgds, containment is in coNEXPTIME^{NP} and hard for P^{NEXP}, even for fixed arity. The lower bound is shown by exploiting a recently introduced tiling problem [21].

We conclude that in all these cases OMQ containment is harder than evaluation, with one exception: the OMQs based on linear tgds over schemas of unbounded arity, where both problems are PSPACE-complete. Regarding OMQs based on non-recursive sets of tgds, although our upper bound is not optimal, it is nearly optimal. Indeed, NEXPTIME^{NP}, which forms the Δ_2-level of the exponential hierarchy (EH), and P^{NEXP}, which forms the Δ_2-level of the *strong* EH,[1] are tightly related: if the oracle access in NEXPTIME^{NP} is restricted too much, then it collapses to P^{NEXP} [28].

Guarded tgds. The OMQ language based on guarded tgds is not UCQ rewritable, which forces us to develop different tools to study its containment problem. Let us remark that guarded OMQs can be rewritten as guarded Datalog queries (by exploiting the translations devised in [3, 26]), for which containment is decidable in 2EXPTIME [13]. But, again, the known rewritings are very large [26], and the reduction of containment for guarded OMQs to containment for guarded Datalog does not yield optimal upper bounds.

To obtain optimal bounds for the problem in question, we exploit *two-way alternating parity automata on trees* (2WAPA) [20, 35]. We show that if Q_1 and Q_2 are guarded OMQs such that Q_1 is not contained in Q_2, then this is witnessed over a class of "tree-like" databases that can be represented as the set of trees accepted by a 2WAPA $\mathfrak{A}$. We then build a 2WAPA $\mathfrak{B}$ with exponentially many states that recognizes those trees accepted by $\mathfrak{A}$ that represent witnesses to non-containment of Q_1 in Q_2. Hence, Q_1 is contained in Q_2 iff $\mathfrak{B}$ accepts no tree. Since the emptiness problem for 2WAPA is feasible in exponential time in the number of states [20], we obtain that containment for guarded OMQs is in 2EXPTIME. A matching lower bound, even for fixed arity schemas, follows from [10].

Similar ideas based on 2WAPA have been recently used to show that containment for OMQs based on expressive DLs is in 2EXPTIME [10]. In the DL context, schemas consist only of unary and

[1] The strong EH collapses to its Δ_2-level [28].

binary relations. Our automata construction, however, is different from the one in [10] for two reasons: (a) we need to deal with higher arity relations, and (b) even for unary and binary relations, our OMQ language allows to express properties that are not expressible by the DL-based OMQ languages studied in [10].

Frontier-guarded tgds. Frontier-guarded tgds generalize guarded tgds, and as a matter of fact the techniques we develop for studying OMQ containment for the latter do not extend in a straightforward manner to the former. Instead, we provide a translation from a frontier-guarded OMQ Q into a guarded OMQ Q' such that Q and Q' are equivalent over acyclic databases. This allows to exploit the machinery developed for guarded OMQs, and show that containment for frontier-guarded OMQs is in 2ExpTime. As for guarded OMQs, a matching lower bound is inherited from [10], even for fixed arity schema. Let us stress that the employed translation from frontier-guarded into guarded OMQs does not preserve the query answers over arbitrary databases, but only over acyclic databases. This is not surprising since frontier-guarded OMQs are strictly more expressive than guarded OMQs; see, e.g., [25].

Combining languages. The above complexity results refer to the containment problem relative to a certain OMQ language $\mathbb{O}$, i.e., both queries fall in $\mathbb{O}$. However, it is natural to consider the version of the problem where the involved OMQs fall in different languages. Unsurprisingly, if the left-hand side query is expressed in a UCQ rewritable OMQ language (based on linear, non-recursive, or sticky sets of tgds), we can use the algorithm that relies on the small witness property discussed above, which provides optimal upper bounds for almost all the considered cases (the only exception is the containment of sticky in non-recursive OMQs over schemas of unbounded arity). Things are more interesting if the ontology of the left-hand side query is expressed using guarded or frontier-guarded tgds, while the ontology of the right-hand side query is not (frontier-)guarded. By using automata techniques, we show that containment of (frontier-)guarded in non-recursive OMQs is in 3ExpTime, while containment of (frontier-)guarded in sticky OMQs is in 2ExpTime. We establish matching lower bounds, even over schemas of fixed arity, by refining techniques from [19].

Organization. Preliminaries are given in Section 2. In Section 3 we introduce the OMQ containment problem. Containment for UCQ rewritable OMQs is studied in Section 4, for guarded OMQs in Section 5, and for frontier-guarded OMQs in Section 6. We consider the case where the involved queries fall in different languages in Section 7. Finally, we conclude in Section 8.

2 PRELIMINARIES

Databases and conjunctive queries. Let $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ be disjoint countably infinite sets of *constants*, *(labeled) nulls*, and (regular) *variables* (used in queries and dependencies), respectively. A *schema* $\mathbf{S}$ is a finite set of relation symbols (or predicates) with associated arity. We write R/n to denote that R has arity n. A *term* is either a constant, null, or variable. An *atom* over $\mathbf{S}$ is an expression of the form $R(\bar{v})$, where $R \in \mathbf{S}$ is of arity $n \geq 0$ and $\bar{v}$ is an n-tuple of terms. A *fact* is an atom whose arguments consist only of constants. An *instance* over $\mathbf{S}$ is a (possibly infinite) set of atoms over $\mathbf{S}$ that contain constants and nulls, while a *database* over $\mathbf{S}$ is a finite set of facts over $\mathbf{S}$. We may call an instance and a database over $\mathbf{S}$ an $\mathbf{S}$-*instance* and $\mathbf{S}$-*database*, respectively. The *active domain* of an instance I, denoted $dom(I)$, is the set of all terms occurring in I.

A *conjunctive query* (CQ) over $\mathbf{S}$ is a formula of the form:

$$q(\bar{x}) \;:=\; \exists \bar{y}\big(R_1(\bar{v}_1) \wedge \cdots \wedge R_m(\bar{v}_m)\big), \tag{1}$$

where each $R_i(\bar{v}_i)$ $(1 \leq i \leq m)$ is an atom without nulls over $\mathbf{S}$, each variable mentioned in the $\bar{v}_i$'s appears either in $\bar{x}$ or $\bar{y}$, and $\bar{x}$ are the *free variables* of q. If $\bar{x}$ is empty, then q is a *Boolean CQ*. As usual, the evaluation of CQs is defined in terms of homomorphisms. Let I be an instance and $q(\bar{x})$ a CQ of the form (1). A *homomorphism* from q to I is a mapping h, which is the identity on $\mathbf{C}$, from the terms that appear in q to the set of constants and nulls $\mathbf{C} \cup \mathbf{N}$ such that $R_i(h(\bar{v}_i)) \in I$, for each $1 \leq i \leq m$. The *evaluation of* $q(\bar{x})$ *over* I, denoted $q(I)$, is the set of all tuples $h(\bar{x})$ of constants such that h is a homomorphism from q to I. We denote by $\mathbb{CQ}$ the class of conjunctive queries. A *union of conjunctive queries* (UCQ) over $\mathbf{S}$ is a formula of the form $q(\bar{x}) := q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$, where each $q_i(\bar{x})$ is a CQ of the form (1). The *evaluation of* $q(\bar{x})$ *over* I, denoted $q(I)$, is the set of tuples $\bigcup_{1 \leq i \leq n} q_i(I)$. We denote by $\mathbb{UCQ}$ the class of union of conjunctive queries.

Tgds and the chase procedure. A *tuple-generating dependency* (tgd) is a first-order sentence of the form:

$$\forall \bar{x} \forall \bar{y}\big(\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})\big), \tag{2}$$

where ϕ and ψ are conjunctions of atoms without nulls. For brevity, we write this tgd as $\phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ and use comma instead of $\wedge$ for conjoining atoms. Notice that ϕ can be empty, in which case the tgd is called *fact tgd* and is written as $\top \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$. We assume that each variable in $\bar{x}$ is mentioned in some atom of ψ. We call ϕ and ψ the *body* and *head* of the tgd, respectively. An instance I over $\mathbf{S}$ *satisfies* the tgd in (2) if the following holds: whenever there is a homomorphism h from ϕ to I, then h can be extended to a homomorphism h' from ψ to I. We say that an instance I satisfies a set Σ of tgds, denoted $I \models \Sigma$, if I satisfies every tgd in Σ. We denote by $\mathbb{TGD}$ the class of (finite) sets of tgds.

The *chase* is a useful algorithmic tool when reasoning with tgds [15, 22, 29, 31]. We start by defining a single chase step. Let I be an instance over a schema $\mathbf{S}$ and $\tau = \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ a tgd over $\mathbf{S}$. We say that τ is *applicable* w.r.t. I if there exists a tuple $(\bar{a}, \bar{b})$ of terms in I such that $\phi(\bar{a}, \bar{b}) \subseteq I$. In this case, *the result of applying* τ *over* I *with* $(\bar{a}, \bar{b})$ is the instance J that extends I with every atom in $\psi(\bar{a}, \bar{\perp})$, where $\bar{\perp}$ is the tuple obtained by simultaneously replacing each variable $z \in \bar{z}$ with a fresh distinct null not occurring in I. For such a single chase step we write $I \xrightarrow{\tau, (\bar{a}, \bar{b})} J$.

Let us assume now that I is an instance and Σ a finite set of tgds. A *chase sequence for* I *under* Σ is a sequence:

$$I_0 \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \cdots$$

of chase steps such that: (1) $I_0 = I$; (2) for each $i \geq 0$, τ_i is a tgd in Σ; and (3) $\bigcup_{i \geq 0} I_i \models \Sigma$. We call $\bigcup_{i \geq 0} I_i$ the *result* of this chase sequence, which always exists. Although the result of a chase sequence is not unique (up to isomorphism), each such result is equally useful for our purposes, since it can be homomorphically embedded into every other result. Henceforth, we denote by $chase(I, \Sigma)$ the result of an arbitrary chase sequence for I under Σ.

Ontology-mediated queries. An *ontology-mediated query* (OMQ) is a triple $(\mathbf{S}, \Sigma, q)$, where $\mathbf{S}$ is a schema, Σ is a set of tgds (the ontology), and q is a (U)CQ over $\mathbf{S} \cup sch(\Sigma)$ (and possibly other predicates), with $sch(\Sigma)$ the set of predicates occurring in Σ.[2] Notice that the set of tgds can introduce predicates not in $\mathbf{S}$, which allows us to enrich the schema of the UCQ q. Moreover, the tgds can modify the content of a predicate $R \in \mathbf{S}$, or, in other words, R can appear in the head of a tgd of Σ. We have explicitly included $\mathbf{S}$ in the specification of the OMQ to emphasize that it will be evaluated over $\mathbf{S}$-databases, even though Σ and q might use additional relational symbols. We call $\mathbf{S}$ the *data schema*.

The semantics of an OMQ is given in terms of certain answers. The *certain answers* to a UCQ $q(\bar{x})$ w.r.t. a database D and a set Σ of tgds is the set of tuples:

$$cert(q, D, \Sigma) \quad = \bigcap_{D \subseteq I \text{ and } I \models \Sigma} \{\bar{c} \in dom(I)^{|\bar{x}|} \mid \bar{c} \in q(I)\}.$$

Consider an OMQ $Q = (\mathbf{S}, \Sigma, q)$. The *evaluation* of Q over an $\mathbf{S}$-database D, denoted $Q(D)$, is defined as $cert(q, D, \Sigma)$. It is well-known that $cert(q, D, \Sigma) = q(chase(D, \Sigma))$ (see, e.g., [15]), which immediately implies that $Q(D) = q(chase(D, \Sigma))$.

Ontology-mediated query languages. We write $(\mathbb{C}, \mathbb{Q})$ for the OMQ language that consists of all OMQs of the form $(\mathbf{S}, \Sigma, q)$, where Σ falls in the class $\mathbb{C}$ of tgds, i.e., $\mathbb{C} \subseteq \mathbb{TGD}$ (concrete classes of tgds are discussed below), and the query q falls in $\mathbb{Q} \in \{\mathbb{CQ}, \mathbb{UCQ}\}$. A problem that is quite important for our work is *OMQ evaluation*, defined as follows:

PROBLEM :	Eval$(\mathbb{C}, \mathbb{Q})$
INPUT :	An OMQ $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{C}, \mathbb{Q})$, an $\mathbf{S}$-database D, and $\bar{c} \in dom(D)^{\lvert\bar{x}\rvert}$.
QUESTION :	Is $\bar{c} \in Q(D)$?

It is well-known that Eval$(\mathbb{TGD}, \mathbb{CQ})$ is undecidable; implicit in [6]. This has led to a flurry of activity for identifying syntactic restrictions on sets of tgds that make the latter problem decidable. Such a restriction defines a subclass $\mathbb{C}$ of tgds. The known decidable classes of tgds are classified into three main decidability paradigms, which, in turn, give rise to decidable OMQ languages:

Guardedness: A tgd is *guarded* (*frontier-guarded*) if it has a body-atom, called *guard* (*frontier-guard*), that contains all the body-variables (all the body-variables that appear in the head). A guarded tgd is trivially frontier-guarded, but there are frontier-guarded tgds that are not guarded. Although the chase under (frontier-) guarded tgds does not necessarily terminate, the problem of deciding whether a tuple of constants is a certain answer to a UCQ w.r.t. a database and a set of (frontier-)guarded tgds is decidable. This follows from the fact that the result of the chase has *bounded treewidth* (see, e.g., [2, 15]). Let $\mathbb{G}$ (resp., $\mathbb{FG}$) be the class of (finite) sets of guarded (resp., frontier-guarded) tgds. Then:

Proposition 2.1. [2, 15] Eval$(\mathbb{G}, (\mathbb{U})\mathbb{CQ})$ *is* 2ExpTime*-complete, and* ExpTime*-complete for fixed arity. Moreover, the problem* Eval$(\mathbb{FG}, (\mathbb{U})\mathbb{CQ})$ *is complete for* 2ExpTime*, even for fixed arity.*[3]

[2] OMQs can be defined for arbitrary first-order theories, not only tgds, and first-order queries, not only UCQs [12].
[3] Eval$(\mathbb{C}, (\mathbb{U})\mathbb{CQ})$ means Eval$(\mathbb{C}, \mathbb{CQ})$ and Eval$(\mathbb{C}, \mathbb{UCQ})$.

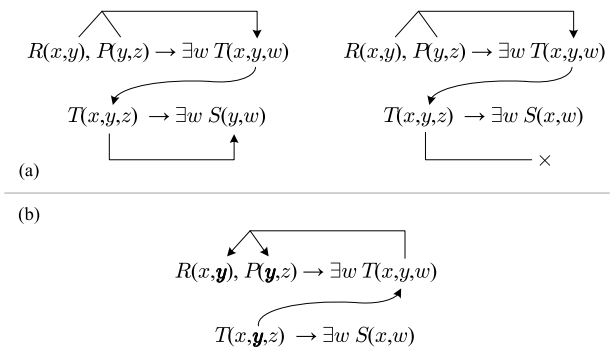

Figure 1: Stickiness and Marking.

An important subclass of guarded tgds is the class of *linear* tgds whose body consists of a single atom. We write $\mathbb{L}$ for the class of (finite) sets of linear tgds. Then:

Proposition 2.2. [16, 29] Eval$(\mathbb{L}, (\mathbb{U})\mathbb{CQ})$ *is* PSpace*-complete, and* NP*-complete for fixed arity.*

Non-recursiveness: A set Σ of tgds is *non-recursive* (a.k.a. *acyclic* [22, 30]), if its predicate graph is acyclic. Recall that the nodes of the predicate graph of Σ are the predicates occurring in Σ, and there is an edge from R to P iff there is a tgd $\sigma \in \Sigma$ such that R occurs in the body of σ and P occurs in the head of σ. Non-recursiveness ensures the termination of the chase, and thus decidability of OMQ evaluation. Let $\mathbb{NR}$ be the class of non-recursive (finite) sets of tgds.

Proposition 2.3. [30] Eval$(\mathbb{NR}, (\mathbb{U})\mathbb{CQ})$ *is* NExpTime*-complete, even for fixed arity.*

Stickiness: This condition ensures neither termination nor bounded treewidth of the chase. Instead, the decidability of OMQ evaluation is obtained by exploiting query rewriting techniques (more details on query rewriting are given in Section 4). The goal of stickiness is to capture joins among variables that are not expressible via guarded tgds, but without forcing the chase to terminate. The key property underlying this condition can be described as follows: during the chase, terms that are associated (via a homomorphism) with variables that appear more than once in the body of a tgd (i.e., join variables) are always propagated (or "stick") to the inferred atoms. This is illustrated in Figure 1(a); the left set of tgds is sticky, while the right set is not. The formal definition is based on an inductive marking procedure that marks the variables that may violate the semantic property of the chase described above [17]. Roughly, during the base step of this procedure, a variable that appears in the body of a tgd τ but not in every head-atom of τ is marked. Then, the marking is inductively propagated from head to body as shown in Figure 1(b). Finally, a finite set of tgds Σ is *sticky* if no tgd in Σ contains two occurrences of a marked variable. Let $\mathbb{S}$ be the class of sticky (finite) sets of tgds. Then:

Proposition 2.4. [17] Eval$(\mathbb{S}, (\mathbb{U})\mathbb{CQ})$ *is* ExpTime*-complete, and* NP*-complete for fixed arity.*

3 OMQ CONTAINMENT: THE BASICS

The goal of this work is to study in depth the problem of checking whether an OMQ Q_1 is *contained* in an OMQ Q_2, both over the same data schema $\mathbf{S}$, or, equivalently, whether $Q_1(D) \subseteq Q_2(D)$ for every (finite) $\mathbf{S}$-database D. In this case we write $Q_1 \subseteq Q_2$; we write $Q_1 \equiv Q_2$ if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$. The *OMQ containment* problem in question is defined as follows; $\mathbb{O}_1$ and $\mathbb{O}_2$ are OMQ languages $(\mathbb{C}, \mathbb{Q})$, where $\mathbb{C}$ is a class of tgds (e.g., linear, non-recursive, sticky, etc.), and $\mathbb{Q} \in \{\mathbb{CQ}, \mathbb{UCQ}\}$:

PROBLEM :	$\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$
INPUT :	Two OMQs $Q_1 \in \mathbb{O}_1$ and $Q_2 \in \mathbb{O}_2$.
QUESTION :	Is $Q_1 \subseteq Q_2$?

Whenever $\mathbb{O}_1 = \mathbb{O}_2 = \mathbb{O}$, we refer to the containment problem by simply writing $\mathsf{Cont}(\mathbb{O})$.

In what follows, we establish some simple but fundamental results, which help to better understand the nature of our problem. We first investigate the relationship between evaluation and containment, which in turn allows us to obtain an initial boundary for the decidability of our problem, i.e., we can obtain a positive result only if the evaluation problem for the involved OMQ languages is decidable (e.g., those introduced in the previous section). We then focus on the OMQ languages introduced in Section 2 and observe that, once we fix the class of tgds, it does not make a difference whether we consider CQs or UCQs. In other words, we show that an OMQ in $(\mathbb{C}, \mathbb{UCQ})$, where $\mathbb{C} \in \{\mathbb{FG}, \mathbb{G}, \mathbb{L}, \mathbb{NR}, \mathbb{S}\}$, can be rewritten as an OMQ in $(\mathbb{C}, \mathbb{CQ})$. This fact simplifies our later complexity analysis since for establishing upper (resp., lower) bounds it suffices to focus on CQs (resp., UCQs).

3.1 Evaluation vs. Containment

OMQ evaluation and OMQ containment are strongly connected. As we explain below, the former can be easily reduced to the latter. But let us first introduce some auxiliary notation. Consider a database D and a tuple $\bar{c} = (c_1, \ldots, c_n) \in \mathit{dom}(D)^n$, where $n \geq 0$. We denote by $q_{D,\bar{c}}(\bar{x})$, where $\bar{x} = (x_{c_1}, \ldots, x_{c_n})$, the CQ obtained from the conjunction of atoms occurring in D after replacing each constant c with the variable x_c. Consider now an OMQ $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C}$ is some class of tgds, an $\mathbf{S}$-database D, and a tuple $\bar{c} \in \mathit{dom}(D)^{|\bar{x}|}$. It is not difficult to show that:

$$\bar{c} \in Q(D) \iff \underbrace{(\mathit{sch}(\Sigma), \varnothing, q_{D,\bar{c}})}_{Q_1} \subseteq \underbrace{(\mathit{sch}(\Sigma), \Sigma, q)}_{Q_2}.$$

Let $\mathbb{O}_\varnothing$ be the OMQ language that consists of all OMQs of the form $(\mathbf{S}, \varnothing, q)$, i.e., the set of tgds is empty, where q is a CQ. It is clear that $Q_1 \in \mathbb{O}_\varnothing$ and $Q_2 \in (\mathbb{C}, \mathbb{CQ})$. Therefore, for every OMQ language $\mathbb{O} = (\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C}$ is a class of tgds, we immediately get that:

Proposition 3.1. $\mathsf{Eval}(\mathbb{O})$ *can be reduced in polynomial time into* $\mathsf{Cont}(\mathbb{O}_\varnothing, \mathbb{O})$.

We now show that the problem of evaluation is also reducible to the complement of containment. Let us say that for technical reasons, which will be made clear in a while, we focus our attention on classes $\mathbb{C}$ of tgds that are *closed under fact tgd extension*, i.e., for every set $\Sigma \in \mathbb{C}$, a set obtained from Σ by adding a (finite) set of fact tgds is still in $\mathbb{C}$. Notice that the classes introduced above enjoy this property. Consider now an OMQ $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C}$ is some class of tgds, an $\mathbf{S}$-database D, and a tuple $\bar{c} \in \mathit{dom}(D)^{|\bar{x}|}$. It is easy to see then that:

$$\bar{c} \in Q(D) \iff \underbrace{(\mathbf{S}, \Sigma_D^\star, q_{\bar{c}}^\star)}_{Q_1} \not\subseteq \underbrace{(\mathbf{S}, \varnothing, \exists x\, P(x))}_{Q_2},$$

where $\Sigma_D^\star$ is obtained from Σ by renaming each predicate R in Σ into $R^\star \notin \mathbf{S}$ and adding the set of fact tgds:

$$\{\top \rightarrow R^\star(c_1, \ldots, c_k) \mid R(c_1, \ldots, c_k) \in D\},$$

$q_{\bar{c}}^\star$ is obtained from $q(\bar{c})$ by renaming each predicate R into $R^\star \notin \mathbf{S}$, and the predicate P does not occur in $\mathbf{S}$. Indeed, the above equivalence holds since $P \notin \mathbf{S}$ implies that $Q_2(D) = \varnothing$, for every $\mathbf{S}$-database D. Since $\mathbb{C}$ is closed under fact tgd extension, $Q_1 \in (\mathbb{C}, \mathbb{CQ})$, while $Q_2 \in \mathbb{O}_\varnothing$. We write $\mathsf{coCont}(\mathbb{O}_1, \mathbb{O}_2)$ for the complement of $\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$. Hence, for every OMQ language $\mathbb{O} = (\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C}$ is a class of tgds (closed under fact tgd extension), it holds that:

Proposition 3.2. $\mathsf{Eval}(\mathbb{O})$ *can be reduced in polynomial time into* $\mathsf{coCont}(\mathbb{O}, \mathbb{O}_\varnothing)$.

By definition, $\mathbb{O}_\varnothing$ is contained in every OMQ language $(\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C}$ is a class of tgds. Therefore, as a corollary of Propositions 3.1 and 3.2, we obtain an initial boundary for the decidability of OMQ containment: we can obtain a positive result only if the evaluation problem for the involved OMQ languages is decidable.

Corollary 3.3. $\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$ *is undecidable if* $\mathsf{Eval}(\mathbb{O}_1)$ *is undecidable or* $\mathsf{Eval}(\mathbb{O}_2)$ *is undecidable.*

Can we prove the converse of Corollary 3.3, i.e., $\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$ is decidable if both $\mathsf{Eval}(\mathbb{O}_1)$ and $\mathsf{Eval}(\mathbb{O}_2)$ are decidable? The answer to this question is negative since containment of Datalog queries is undecidable [34]. Indeed, Datalog queries can be directly encoded in the OMQ language based on the class $\mathbb{F}$ of *full tgds*, i.e., tgds without existentially quantified variables. The next result follows:

Proposition 3.4. [34] $\mathsf{Cont}((\mathbb{F}, \mathbb{CQ}))$ *is undecidable.*

This result, combined with the fact that $\mathsf{Eval}(\mathbb{F})$ is decidable (since the chase under full tgds always terminates), implies that the converse of Corollary 3.3 does not hold. Proposition 3.4 also rules out the OMQ languages that are based on classes of tgds that generalize $\mathbb{F}$; e.g., the weak versions of the ones introduced in Section 2, called *weakly frontier-guarded* [2], *weakly guarded* [15], *weakly acyclic* [22], and *weakly sticky* [17] that guarantee the decidability of OMQ evaluation.[4] The question that comes up concerns the decidability and complexity of containment for the OMQ languages that are based on the non-weak versions of the above classes, i.e., frontier-guarded, guarded, non-recursive, and sticky. This will be the subject of the next three sections.

3.2 From UCQs to CQs

Before we proceed further, let us state the following useful result:

[4]The idea of those classes is the same: relax the condition of the class so that only those positions that receive null values during the chase are taken into account.

Proposition 3.5. *Given an OMQ $Q \in (\mathbb{C}, \mathbb{UCQ})$, where $\mathbb{C} \in \{\mathbb{FG}, \mathbb{G}, \mathbb{L}, \mathbb{NR}, \mathbb{S}\}$, we can construct in polynomial time an OMQ $Q' \in (\mathbb{C}, \mathbb{CQ})$ such that $Q \equiv Q'$.*

The proof of the above result relies on the idea of encoding boolean operations (in our case the 'or' operator) using a set of atoms; this idea has been used in several works (see, e.g., [8, 14, 24]). Proposition 3.5 allows us to focus on OMQs that are based on CQs. Assuming that $\mathbb{C}_1, \mathbb{C}_2 \in \{\mathbb{FG}, \mathbb{G}, \mathbb{L}, \mathbb{NR}, \mathbb{S}\}$ and C is a complexity class that is closed under polynomial time reductions, then:

$$\mathsf{Cont}((\mathbb{C}_1, \mathbb{CQ}), (\mathbb{C}_2, \mathbb{CQ})) \text{ is } C\text{-complete} \iff \mathsf{Cont}((\mathbb{C}_1, \mathbb{UCQ}), (\mathbb{C}_2, \mathbb{UCQ})) \text{ is } C\text{-complete.}$$

3.3 Plan of Attack

We are now ready to proceed with the complexity analysis of containment for the OMQ languages in question. Our plan of attack can be summarized as follows:

- We consider, in Section 4, $\mathsf{Cont}((\mathbb{C}, \mathbb{CQ}))$, for $\mathbb{C} \in \{\mathbb{L}, \mathbb{NR}, \mathbb{S}\}$. These languages enjoy the key property of UCQ rewritability. This property allows us to show the following result: if the containment does not hold, then this is witnessed via a "small" database, which in turn allows us to devise simple guess-and-check algorithms.
- We then proceed, in Section 5, with $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$. This OMQ language does not enjoy UCQ rewritability, and the task of establishing a small witness property that leads to an optimal upper bound turned out to be challenging. However, non-containment is witnessed via a "tree-like" database, which allows us to devise a decision procedure based on two-way alternating parity automata on finite trees.
- $\mathsf{Cont}((\mathbb{FG}, \mathbb{CQ}))$ is studied in Section 6. Recall that $(\mathbb{FG}, \mathbb{CQ})$ is strictly more expressive than $(\mathbb{G}, \mathbb{CQ})$, and thus, we cannot directly apply the machinery for $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$. Nevertheless, after focusing on acyclic databases, frontier-guarded OMQs can be rewritten as guarded OMQs, which essentially provides a reduction from $\mathsf{Cont}((\mathbb{FG}, \mathbb{CQ}))$ to $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$.
- In Section 7, we study the case where the OMQ containment problem involves two different languages. If the left-hand side language is UCQ rewritable, then we can devise a guess-and-check algorithm based on the above small witness property. The interesting case is when the left-hand side language is $(\mathbb{FG}, \mathbb{CQ})$, where we employ tree automata techniques.

4 UCQ REWRITABLE LANGUAGES

We now focus on OMQ languages that enjoy the crucial property of UCQ rewritability. Roughly, an OMQ language $\mathbb{O}$ is UCQ rewritable if every query in $\mathbb{O}$ can be equivalently rewritten as a UCQ. The formal definition follows:

Definition 4.1. (**UCQ Rewritability**) We call an OMQ language $(\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C} \subseteq \mathbb{TGD}$, *UCQ rewritable* if, for each OMQ $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{C}, \mathbb{CQ})$, we can construct a UCQ $q'(\bar{x})$ such that $Q(D) = q'(D)$ for every S-database D. ∎

We proceed to establish our desired small witness property based on UCQ rewritability. By the definition of UCQ rewritability, for each language $\mathbb{O}$ that is UCQ rewritable, there exists a computable function $f_{\mathbb{O}}$ from $\mathbb{O}$ to the natural numbers such that the following holds: for every OMQ $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in \mathbb{O}$, and UCQ rewriting $q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$ of Q, it is the case that $\max_{1 \leq i \leq n}\{|q_i|\} \leq f_{\mathbb{O}}(Q)$, where $|q_i|$ denotes the number of atoms occurring in q_i. Then:

Proposition 4.2. *Consider a UCQ rewritable language $\mathbb{O}$, and two OMQs $Q \in \mathbb{O}$ and $Q' \in (\mathbb{TGD}, \mathbb{CQ})$, both with data schema* S. *If $Q \not\subseteq Q'$, then there exists an* S*-database D, where $|D| \leq f_{\mathbb{O}}(Q)$, such that $Q(D) \not\subseteq Q'(D)$.*

Proof (sketch). We assume that $q(\bar{x}) = \bigvee_{i=1}^{n} q_i(\bar{x})$ is a UCQ rewriting of Q. Since, by hypothesis, $Q \not\subseteq Q'$, we conclude that $q \not\subseteq Q'$, which in turn implies that there exists an $i \in \{1, \ldots, n\}$ such that $q_i \not\subseteq Q'$. It is easy to show that $c(\bar{x}) \notin Q'(D_{q_i})$, where $c(\bar{x})$ is a tuple of constants obtained by replacing each variable x in $\bar{x}$ with the constant $c(x)$, and D_{q_i} is the S-database obtained from q_i after replacing each variable x in q_i with the constant $c(x)$. Since $c(\bar{x}) \in q(D_{q_i})$, we get that $c(\bar{x}) \in Q(D_{q_i})$. Therefore, $Q(D_{q_i}) \not\subseteq Q'(D_{q_i})$, and the claim follows since $|D_{q_i}| \leq f_{\mathbb{O}}(Q)$. □

In Proposition 4.2 we only assume that the left-hand side query falls in a UCQ rewritable language, without any assumption on the language of the right-hand side query. Thus, we immediately get a decision procedure for $\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$ if $\mathbb{O}_1$ is UCQ rewritable and $\mathsf{Eval}(\mathbb{O}_2)$ is decidable. Given $Q_1 = (\mathbf{S}, \Sigma_1, q_1(\bar{x})) \in \mathbb{O}_1$ and $Q_2 = (\mathbf{S}, \Sigma_2, q_2(\bar{x})) \in \mathbb{O}_2$:

(1) Guess an S-database D such that $|D| \leq f_{\mathbb{O}_1}(Q_1)$, and a tuple $\bar{c} \in dom(D)^{|\bar{x}|}$; and
(2) Verify that $\bar{c} \in Q_1(D)$ and $\bar{c} \notin Q_2(D)$.

We immediately get that:

Theorem 4.3. $\mathsf{Cont}(\mathbb{O}_1, \mathbb{O}_2)$ *is decidable if $\mathbb{O}_1$ is UCQ rewritable and* $\mathsf{Eval}(\mathbb{O}_2)$ *is decidable.*

This generic result shows that $\mathsf{Cont}((\mathbb{C}, \mathbb{CQ}))$ is decidable for every class $\mathbb{C} \in \{\mathbb{L}, \mathbb{NR}, \mathbb{S}\}$, but it says nothing about complexity. This will be the subject of the rest of the section.

4.1 Linearity

The problem of computing UCQ rewritings for OMQs in $(\mathbb{L}, \mathbb{CQ})$ has been studied in [23], where a resolution-based procedure, called XRewrite, has been proposed. This rewriting algorithm accepts a query $Q = (\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{L}, \mathbb{CQ})$ and constructs a UCQ rewriting $q'(\bar{x})$ over S by starting from q and exhaustively applying rewriting steps based on resolution. Due to the fact that the set of tgds is linear, i.e., the tgd-bodies consist of single atoms, during the execution of XRewrite, it is not possible to obtain a CQ that has more atoms than the original one. Therefore:

Proposition 4.4. $f_{(\mathbb{L}, \mathbb{CQ})}\big((\mathbf{S}, \Sigma, q)\big) \leq |q|$.

Having the above result in place, it can be shown that the algorithm underlying Theorem 4.3 guesses a polynomially sized witness to non-containment, and then calls a C-oracle for solving query evaluation under linear OMQs, where C is PSpace in general, and NP if the arity is fixed; these complexity classes are obtained from Proposition 2.2. Therefore, $\mathsf{coCont}((\mathbb{L}, \mathbb{CQ}))$ is in PSpace in general, and in Σ_2^P in case of fixed arity. Regarding the lower bounds, Proposition 3.1 allows us to inherit the PSpace-hardness of $\mathsf{Eval}(\mathbb{L}, \mathbb{CQ})$;

this holds even for constant-free tgds. Unfortunately, in the case of fixed arity, we can only obtain NP-hardness, while Proposition 3.2 allows to obtain coNP-hardness. Nevertheless, it is implicit in [11] (see the proof of Theorem 9), where the containment problem for OMQ languages based on description logics is considered, that $\mathsf{Cont}((\mathbb{L}, \mathbb{CQ}))$ is Π_2^P-hard, even for tgds of the form $P(x) \rightarrow R(x)$.

Theorem 4.5. $\mathsf{Cont}((\mathbb{L}, \mathbb{CQ}))$ *is* PSpace*-complete, and* Π_2^P*-complete if the arity of the schema is fixed. The lower bounds hold even for tgds without constants.*

4.2 Non-Recursiveness

Although the OMQ language $(\mathbb{NR}, \mathbb{CQ})$ is not explicitly considered in [23], where the algorithm XRewrite is defined, the same algorithm can deal with $(\mathbb{NR}, \mathbb{CQ})$. By analyzing the UCQ rewritings constructed by XRewrite, whenever the input query falls in $(\mathbb{NR}, \mathbb{CQ})$, we can establish the following result; here, $body(\tau)$ denotes the body of the tgd τ:

Proposition 4.6. *It holds that:*

$$f_{(\mathbb{NR},\mathbb{CQ})}\big((\mathbf{S}, \Sigma, q)\big) \;\leq\; |q| \cdot \left(\max_{\tau \in \Sigma}\{|body(\tau)|\}\right)^{|sch(\Sigma)|}.$$

Proposition 4.6 implies that non-containment for queries that fall in $(\mathbb{NR}, \mathbb{CQ})$ is witnessed via a database of at most exponential size. We show next that this bound is optimal:

Proposition 4.7. *There are sets of* $(\mathbb{NR}, \mathbb{CQ})$ *OMQs*

$$\{Q_1^n = (\mathbf{S}, \Sigma_1^n, q_1)\}_{n>0} \quad \textit{and} \quad \{Q_2^n = (\mathbf{S}, \Sigma_2^n, q_2)\}_{n>0},$$

where $|sch(\Sigma_1^n)| = |sch(\Sigma_2^n)| = n + 2$*, such that for every* $\mathbf{S}$*-database* D*, if* $Q_1^n(D) \not\subseteq Q_2^n(D)$ *then* $|D| \geq 2^{n-1}$*.*

Let us now focus on the complexity of $\mathsf{Cont}((\mathbb{NR}, \mathbb{CQ}))$. By naively combining the algorithm underlying Theorem 4.3 and the exponential bound provided by Proposition 4.6, we get that $\mathsf{coCont}((\mathbb{NR}, \mathbb{CQ}))$ is feasible in non-deterministic exponential time with access to a NExpTime oracle; the oracle is needed for solving $\mathsf{Eval}(\mathbb{NR}, \mathbb{CQ})$. Nevertheless, this rough upper bound can be significantly improved; in fact, it can be decreased to $\text{NExpTime}^{\text{NP}}$, which is nearly optimal (more details are given below), by employing a refined version of the algorithm underlying Theorem 4.3. Recall that $\text{NExpTime}^{\text{NP}}$ forms the second level of the exponential hierarchy, a.k.a. Σ_2^{EXP}, and it collects all the decision problems that can be solved via an alternating exponential time algorithm with two alternations that starts from an existential state, i.e., it can perform a series of existential steps followed by a series of universal steps. The refined version of the algorithm underlying Theorem 4.3 is such an algorithm.

Before giving this algorithm, let us recall a crucial property of non-recursive OMQs. Given a database D, an OMQ $(\mathbf{S}, \Sigma, q(\bar{x})) \in (\mathbb{NR}, \mathbb{CQ})$, and a tuple $\bar{c} \in dom(D)^{|\bar{x}|}$, if $\bar{c} \in Q(D)$ then there exists a finite chase sequence:

$$D = I_0 \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \cdots I_{n-1} \xrightarrow{\tau_{n-1}, \bar{c}_{n-1}} I_{g(D,\Sigma)}$$

for D under Σ, where:

$$g(D, \Sigma) \;=\; |D| \cdot \left(\max_{\tau \in \Sigma}\{|body(\tau)|\}\right)^{|sch(\Sigma)|}$$

such that $\bar{c} \in q(I_{g(D,\Sigma)})$; implicit in [30]. Having this property in place, we can now present our alternating algorithm. Given $Q_1 = (\mathbf{S}, \Sigma_1, q_1(\bar{x}))$ and $Q_2 = (\mathbf{S}, \Sigma_2, q_2(\bar{x}))$:

(1) Guess an $\mathbf{S}$-database D of size at most $f_{(\mathbb{NR},\mathbb{CQ})}(Q_1)$, and a tuple $\bar{c} \in dom(D)^{|\bar{x}|}$.
(2) Guess a chase sequence
$$D \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \cdots I_{n-1} \xrightarrow{\tau_{n-1}, \bar{c}_{n-1}} I_{g(D,\Sigma_1)}$$
for D under Σ_1.
(3) Guess a mapping h, which is the identity on $\mathbf{C}$, from the variables in q_1 to $dom(I_{g(D,\Sigma_1)})$.
(4) If h is a homomorphism from q_1 to $I_{g(D,\Sigma_1)}$ such that $h(\bar{x}) = \bar{c}$, then proceed; otherwise, reject.
(5) Universally select each chase sequence
$$D \xrightarrow{\tau_0, \bar{c}_0} I_1 \xrightarrow{\tau_1, \bar{c}_1} I_2 \cdots I_{n-1} \xrightarrow{\tau_{n-1}, \bar{c}_{n-1}} I_{g(D,\Sigma_2)}$$
for D under Σ_2.
(6) Universally select each mapping h, which is the identity on $\mathbf{C}$, from the variables in q_2 to $dom(I_{g(D,\Sigma_2)})$.
(7) If h is a homomorphism from q_2 to $I_{g(D,\Sigma_2)}$ such that $h(\bar{x}) = \bar{c}$, then reject; otherwise, accept.

The above algorithm is an alternating exponential time algorithm with two alternations that starts from an existential state. Moreover, it accepts iff $Q_1 \not\subseteq Q_2$, and the desired upper bound follows.

It is not known whether our problem is $\text{coNExpTime}^{\text{NP}}$-complete. Nevertheless, we provide a nearly matching lower bound; in fact, P^{NEXP}-hardness. More details on how the above complexity classes are related are discussed below. Let us now explain how P^{NEXP}-hardness is obtained. To this end, we exploit a tiling problem that has been recently introduced in [21]. Roughly speaking, an instance of this tiling problem is a triple (m, T_1, T_2), where m is an integer in unary representation, and T_1, T_2 are standard tiling problems for the $(2^n \times 2^n)$-grid. The question is whether, for every initial condition w of length m, T_1 has no solution with w or T_2 has some solution with w. The initial condition w simply fixes the first m tiles of the first row of the grid. We construct in polynomial time two $(\mathbb{NR}, \mathbb{CQ})$ queries Q_1 and Q_2 such that (m, T_1, T_2) has a solution iff $Q_1 \subseteq Q_2$. The idea is to force every input database to store an initial condition w of length m, and then encode the problem whether T_i has a solution with w into Q_i, for each $i \in \{1, 2\}$. Then:

Theorem 4.8. $\mathsf{Cont}((\mathbb{NR}, \mathbb{CQ}))$ *is in* $\text{coNExpTime}^{\text{NP}}$*, and* P^{NEXP}*-hard. The lower bound holds even if the arity of the schema is fixed and the tgds are without constants.*

$\text{NExpTime}^{\text{NP}}$ vs. P^{NEXP}. It is known that $\text{NExpTime}^{\text{NP}}$ is a delicate class: if we restrict its oracle access too much, it collapses to P^{NEXP} [28]. For example, following the notation of [28], P^{NEXP} coincides with $\text{NExpTime}^{\text{NP}[poly]_{tree}}$, where only polynomially many oracle calls are allowed throughout the computation tree of the Turing machine. Also, P^{NEXP} coincides with $\text{NExpTime}^{\text{NP}[poly]_{path}[exp]_{yes,tree}}$, where only polynomially many oracle calls are allowed on each path of the computation tree, and exponentially many calls with a "yes" answer throughout the computation tree of the Turing machine. The above results support our claim that P^{NEXP} is a nearly matching lower bound for $\mathsf{Cont}((\mathbb{NR}, \mathbb{CQ}))$.

4.3 Stickiness

We now focus on OMQs that fall in $(\mathbb{S}, \mathbb{CQ})$. As shown in [23], given a query $(\mathbf{S}, \Sigma, q)$, there exists an execution of XRewrite that constructs a UCQ rewriting $q_1(\bar{x}) \vee \cdots \vee q_n(\bar{x})$ over $\mathbf{S}$ with the following property: for each $i \in \{1, \ldots, n\}$, if a variable v occurs in q_i in more than one atom, then v already occurs in q. This property has been used in [23] to bound the number of atoms that can appear in a single CQ q_i. We write $T(q)$ for the set of terms (constants and variables) occurring in q; $C(\Sigma)$ for the set of constants occurring in Σ; and $ar(\mathbf{S})$ for the maximum arity over all predicates of $\mathbf{S}$.

Proposition 4.9. *It holds that*

$$f_{(\mathbb{S},\mathbb{CQ})}((\mathbf{S}, \Sigma, q)) \;\leq\; |\mathbf{S}| \cdot (|T(q)| + |C(\Sigma)| + 1)^{ar(\mathbf{S})}.$$

Proposition 4.9 implies that non-containment for $(\mathbb{S}, \mathbb{CQ})$ queries is witnessed via a database of at most exponential size. As for $(\mathbb{NR}, \mathbb{CQ})$ queries, we can show that this bound is optimal; for a set Σ of tgds, we write $\|\Sigma\|$ for the number of symbols occurring in Σ.

Proposition 4.10. *There exists a set of $(\mathbb{S}, \mathbb{CQ})$ OMQs:*

$$\{Q^n = (\{S/n\}, \Sigma^n, q(\bar{x}))\}_{n>0}, \text{ where } \|\Sigma^n\| \in O(n^2),$$

such that for every $Q = (\{S\}, \Sigma', q'(\bar{x})) \in (\mathbb{TGD}, \mathbb{CQ})$ and $\{S\}$-database D, if $Q^n(D) \not\subseteq Q(D)$ then $|D| \geq 2^{n-2}$.

We now study the complexity of $\mathsf{Cont}((\mathbb{S}, \mathbb{CQ}))$. Let us first look at schemas of unbounded arity. Proposition 4.9 implies that the algorithm underlying Theorem 4.3 runs in exponential time assuming access to a C-oracle, where C is a complexity class powerful enough for solving $\mathsf{Eval}(\mathbb{S}, \mathbb{CQ})$ and its complement. But, since $\mathsf{Eval}(\mathbb{S}, \mathbb{CQ})$ is in ExpTime (see Proposition 2.4), both $\mathsf{Eval}(\mathbb{S}, \mathbb{CQ})$ and its complement are in NExpTime, and thus, the oracle call is not really needed. From this discussion, we conclude that $\mathsf{coCont}((\mathbb{C}, \mathbb{CQ}))$ is in NExpTime. A matching lower bound is obtained by a reduction from the standard tiling problem for the $(2^n \times 2^n)$-grid. In fact, the same lower bound has been recently established in [9]; however, our result is stronger as it shows that the problem remains hard even if the right-hand side query is a linear OMQ of a simple form – this is also discussed in Section 7, where containment of queries that fall in different OMQ languages is studied. Regarding schemas of fixed arity, Proposition 4.9 provides a witness for non-containment of polynomial size, which implies that the algorithm underlying Theorem 4.3 runs in polynomial time with access to an NP-oracle. Therefore, $\mathsf{coEval}(\mathbb{S}, \mathbb{CQ})$ is in Σ_2^P, while a matching lower bound is implicit in [11].

Theorem 4.11. $\mathsf{Cont}((\mathbb{S}, \mathbb{CQ}))$ *is* coNExpTime*-compl., even if the set of tgds uses only two constants. In the case of fixed arity, it is Π_2^P-complete, even for constant-free tgds.*

5 GUARDEDNESS

We proceed with the problem of containment for guarded OMQs, and we establish the following result:

Theorem 5.1. $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$ *is* 2ExpTime*-complete. The lower bound holds even if the arity of the schema is fixed, and the tgds are without constants.*

The lower bound is immediately inherited from [10], where it is shown that containment for OMQs based on the description logic $\mathcal{ELI}$ is 2ExpTime-hard. Recall that a set of $\mathcal{ELI}$ axioms can be equivalently rewritten as a constant-free set of guarded tgds using only unary and binary predicates, which implies the lower bound stated in Theorem 5.1. However, we cannot immediately inherit the desired upper bound since the DL-based OMQ languages considered in [10] are either weaker than or incomparable to $(\mathbb{G}, \mathbb{CQ})$. Nevertheless, the technique developed in [10] was extremely useful for our analysis. Actually, our automata-based procedure exploits a combination of ideas from [10, 27]. The rest of this section is devoted to providing a high-level explanation of this procedure.

For clarity, we focus on constant-free tgds and CQs, but all the results can be extended to the general case at the price of more involved definitions and proofs. Moreover, for simplicity, we focus on Boolean CQs. In other words, we study the problem for $(\mathbb{G}, \mathbb{BCQ})$, where $\mathbb{BCQ}$ denotes the class of Boolean CQs. This does not affect the generality of our proof since it is known that $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$ can be reduced in polynomial time to $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$ [10].

A first glimpse. As said, $(\mathbb{G}, \mathbb{CQ})$ is not UCQ rewritable and, therefore, we cannot employ Proposition 4.2 in order to establish a small witness property as in Section 4. We have tried, by following a different route, to establish a small witness property for $(\mathbb{G}, \mathbb{CQ})$, which can then be used for obtaining an optimal upper bound for $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$, but it turned out to be a difficult task. Nevertheless, we can show a tree witness property, which states that non-containment for $(\mathbb{G}, \mathbb{CQ})$ is witnessed via a tree-like database. This allows us to devise a procedure based on alternating tree automata. Summing up, the proof for the 2ExpTime membership of $(\mathbb{G}, \mathbb{CQ})$ proceeds in three steps:

(1) Establish a tree witness property;
(2) Encode the tree-like witnesses as trees that can be accepted by an alternating tree automaton; and
(3) Construct an automaton that decides $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$; in fact, we reduce $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$ into emptiness for two-way alternating parity automata on finite trees.

Each one of the above three steps is discussed in more detail in the following three sections. Let us say that our automata-based approach provides a small witness property for $(\mathbb{G}, \mathbb{CQ})$. We obtain that non-containment is witnessed via a triple-exponentially-sized database; details are given below. However, we do not know whether this is optimal.

5.1 Tree Witness Property

From the above informal discussion, it is clear that tree-like databases are crucial for our analysis. Let us make this notion more precise using guarded tree decompositions. A *tree decomposition* of a database D is a labeled rooted tree $T = (V, E, \lambda)$, where $\lambda : V \to 2^{dom(D)}$, such that: (i) for each atom $R(t_1, \ldots, t_n) \in D$, there exists $v \in V$ such that $\lambda(v) \supseteq \{t_1, \ldots, t_n\}$, and (ii) for every term $t \in dom(D)$, the set $\{v \in V \mid t \in \lambda(v)\}$ induces a connected subtree of T. The tree decomposition T is called $[U]$*-guarded*, where $U \subseteq V$, if, for every node $v \in V \setminus U$, there exists an atom $R(t_1, \ldots, t_n) \in D$ such that $\lambda(v) \subseteq \{t_1, \ldots, t_n\}$. We write $root(T)$ for the root node of T, and $D_T(v)$, where $v \in V$, for the subset of D induced by $\lambda(v)$. We are now ready to formalize the notion of the tree-like database:

Definition 5.2. An S-database D is a C-*tree*, where $C \subseteq D$, if there is a tree decomposition T of D such that:

(1) $D_T(root(T)) = C$ and
(2) T is $[\{root(T)\}]$-guarded. ∎

Roughly, whenever a database D is a C-tree, C is the cyclic part of D, while the rest of D is tree-like. For deciding $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$ it suffices to focus on databases that are C-trees and $|dom(C)|$ depends only on the left-hand side OMQ. Recall that for a schema S we write $ar(\mathbf{S})$ for the maximum arity over all predicates of S. Then:

Proposition 5.3. *Let $Q_i = (\mathbf{S}, \Sigma_i, q_i) \in (\mathbb{G}, \mathbb{BCQ})$, for $i \in \{1, 2\}$. The following are equivalent:*

(1) $Q_1 \subseteq Q_2$.
(2) $Q_1(D) \subseteq Q_2(D)$, for every C-tree S-database D such that $|dom(C)| \leq (ar(\mathbf{S} \cup sch(\Sigma_1)) \cdot |q_1|)$.

The fact that (1) $\Rightarrow$ (2) holds trivially, while (2) $\Rightarrow$ (1) is shown by using a variant of the notion of guarded unravelling and compactness. Let us clarify that the above result does not provide a decision procedure for $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$, since we have to consider infinitely many databases that are C-trees with $|dom(C)| \leq (ar(\mathbf{S} \cup sch(\Sigma_1)) \cdot |q_1|)$.

5.2 Encoding Tree-like Databases

The *treewidth* of a database D is the minimum width among all the tree decompositions $T = (V, E, \lambda)$ of D, while the width of T is defined as $\max_{v \in V}\{|\lambda(v)|\} - 1$. It is generally known that a database D whose treewidth is bounded by an integer k can be encoded into a tree over a finite alphabet of double-exponential size in k that can be accepted by an alternating tree automaton; see, e.g., [7].

Consider an alphabet Γ, and let $\mathbb{N}^*$ be the set of finite sequences of natural numbers, including the empty sequence. A Γ-*labeled tree* is a pair $L = (N, \lambda)$, where $N \subseteq \mathbb{N}^*$ is closed under prefixes, and $\lambda\colon N \to \Gamma$ is the labeling function. The elements of N identify the nodes of L. It can be shown that D and a tree decomposition T of D with width k can be encoded as a Γ-labeled tree L, where Γ is an alphabet of double-exponential size in k, such that each node of T corresponds to exactly one node of L and vice versa.

Consider now a C-tree S-database D, and let T be the tree decomposition that witnesses that D is a C-tree. The width of T is at most $k = (|dom(C)| + ar(\mathbf{S}) - 1)$, and thus, the treewidth of D is bounded by k. Hence, from the above discussion, D and T can be encoded as a Γ-labeled tree, where Γ is of double-exponential size in k. In general, given an S-database D that is a C-tree due to the tree decomposition T, we show that D and T can be encoded as a $\Gamma_{\mathbf{S},l}$-labeled tree, with $|dom(C)| \leq l$ and $|\Gamma_{\mathbf{S},l}|$ being double-exponential in $ar(\mathbf{S})$ and exponential in $|\mathbf{S}|$ and l.

Although every C-tree S-database D can be encoded as a $\Gamma_{\mathbf{S},l}$-labeled tree, the other direction does not hold. In other words, it is not true that every $\Gamma_{\mathbf{S},l}$-labeled tree encodes a C-tree S-database D and its corresponding tree decomposition. In view of this fact, we need the additional notion of consistency. A $\Gamma_{\mathbf{S},l}$-labeled tree is called *consistent* if it satisfies certain syntactic properties – we do not give these properties here since they are not vital in order to understand the high-level idea of the proof. Now, given a consistent $\Gamma_{\mathbf{S},l}$-labeled tree L, we can show that L can be decoded into an S-database $[\![L]\!]$ that is a C-tree with $|dom(C)| \leq l$. From the above discussion and Proposition 5.3, we obtain:

Lemma 5.4. *Let $Q_i = (\mathbf{S}, \Sigma_i, q_i) \in (\mathbb{G}, \mathbb{BCQ})$, for $i \in \{1, 2\}$. The following are equivalent:*

(1) $Q_1 \subseteq Q_2$.
(2) $Q_1([\![L]\!]) \subseteq Q_2([\![L]\!])$, for every consistent $\Gamma_{\mathbf{S},l}$-labeled tree L, where $l = (ar(\mathbf{S} \cup sch(\Sigma_1)) \cdot |q_1|)$.

5.3 Constructing Tree Automata

Having the above result in place, we can now proceed with our automata-based procedure. We use two-way alternating parity automata (2WAPA) that run on finite labeled trees. Two-way alternating automata process the input tree while branching in an alternating fashion to successor states, and thereby moving either down or up the input tree. Our goal is to reduce $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$ to the emptiness problem for 2WAPA. As usual, given a 2WAPA $\mathfrak{A}$, we denote by $\mathcal{L}(\mathfrak{A})$ the *language* of $\mathfrak{A}$, i.e., the set of labeled trees it accepts. The emptiness problem is defined as follows: given a 2WAPA $\mathfrak{A}$, does $\mathcal{L}(\mathfrak{A}) = \varnothing$? Thus, given $Q_1, Q_2 \in (\mathbb{G}, \mathbb{BCQ})$, we need to construct a 2WAPA $\mathfrak{A}$ such that $Q_1 \subseteq Q_2$ iff $\mathcal{L}(\mathfrak{A}) = \varnothing$. Deciding whether $\mathcal{L}(\mathfrak{A})$ is empty is feasible in exponential time in the number of states, and in polynomial time in the size of the input alphabet [20]. Therefore, we should construct $\mathfrak{A}$ in double-exponential time, while the number of states must be at most exponential.

We first need a way to check consistency of labeled trees. It is not difficult to devise an automaton for this task.

Lemma 5.5. *Consider a schema S and an integer $l > 0$. There is a 2WAPA $\mathfrak{C}_{\mathbf{S},l}$ that accepts a $\Gamma_{\mathbf{S},l}$-labeled tree L iff L is consistent. The number of states of $\mathfrak{C}_{\mathbf{S},l}$ is logarithmic in the size of $\Gamma_{\mathbf{S},l}$. Furthermore, $\mathfrak{C}_{\mathbf{S},l}$ can be constructed in polynomial time in the size of $\Gamma_{\mathbf{S},l}$.*

Now, the crucial task is, given an OMQ $Q \in (\mathbb{G}, \mathbb{BCQ})$, to devise an automaton that accepts labeled trees which correspond to databases that make Q true.

Lemma 5.6. *Let $Q = (\mathbf{S}, \Sigma, q) \in (\mathbb{G}, \mathbb{BCQ})$. There is a 2WAPA $\mathfrak{A}_{Q,l}$, where $l > 0$, that accepts a consistent $\Gamma_{\mathbf{S},l}$-labeled tree L iff $Q([\![L]\!]) \neq \varnothing$. $\mathfrak{A}_{Q,l}$ has exponentially many states in $\|Q\|$ and l, and it can be constructed in double-exponential time in $\|Q\|$ and l.*

The intuition underlying $\mathfrak{A}_{Q,l}$ can be described as follows. $\mathfrak{A}_{Q,l}$ tries to identify all the possible ways the CQ q can be mapped to $chase(D, \Sigma)$, for *any* C-tree S-database D such that $|dom(C)| \leq l$. It then arrives at possible ways how the input tree can satisfy Q. These "possible ways" correspond to *squid decompositions*, a notion introduced in [15] that indicates which part of the query is mapped to the cyclic part C of D, and which to the tree-like part of D. The automaton exhaustively checks all squid decompositions by traversing the input tree and, at the same time, explores possible ways how to match the single parts of the squid decomposition at hand. The automaton finally accepts if it finds a squid decomposition that can be mapped to $chase(D, \Sigma)$.

Having the above automata in place, we can proceed with our main technical result, which shows that $\mathsf{Cont}(\mathbb{G}, \mathbb{BCQ})$ can be reduced to the emptiness problem for 2WAPA. But let us first recall some key results about 2WAPA, which are essential for our final construction. It is well-known that languages accepted by 2WAPAs

are closed under intersection and complement. Given two 2WAPAs $\mathfrak{A}_1$ and $\mathfrak{A}_2$, we write $\mathfrak{A}_1 \cap \mathfrak{A}_2$ for a 2WAPA, which can be constructed in polynomial time, that accepts the language $\mathcal{L}(\mathfrak{A}_1) \cap \mathcal{L}(\mathfrak{A}_2)$. Moreover, for a 2WAPA $\mathfrak{A}$, we write $\overline{\mathfrak{A}}$ for a 2WAPA, which is also constructible in polynomial time, that accepts the complement of $\mathcal{L}(\mathfrak{A})$. We can now show the following:

Proposition 5.7. *Consider $Q_1, Q_2 \in (\mathbb{G}, \mathbb{BCQ})$. We can construct in double-exponential time a 2WAPA $\mathfrak{A}$, which has exponentially many states, such that*

$$Q_1 \subseteq Q_2 \iff \mathcal{L}(\mathfrak{A}) = \varnothing.$$

Proof (sketch). Let $Q_i = (\mathbf{S}, \Sigma_i, q_i)$, for $i \in \{1, 2\}$, and $l = (ar(\mathbf{S} \cup sch(\Sigma_1)) \cdot |q_1|)$. Then $\mathfrak{A}$ is defined as

$$(\mathfrak{C}_{\mathbf{S},l} \cap \mathfrak{A}_{Q_1,l}) \cap \overline{\mathfrak{A}_{Q_2,l}}.$$

Since $\Gamma_{\mathbf{S},l}$ has double-exponential size, Lemmas 5.5 and 5.6 imply that $\mathfrak{A}$ can be constructed in double-exponential time, while it has exponentially many states. Lemma 5.4 implies that $Q_1 \subseteq Q_2$ iff $\mathcal{L}(\mathfrak{A}) = \varnothing$, and the claim follows. □

Proposition 5.7 implies that $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$ is in 2ExpTime, and Theorem 5.1 follows. The above proposition provides a small witness property for $\mathsf{Cont}((\mathbb{G}, \mathbb{BCQ}))$. In particular, if $Q_1 \not\subseteq Q_2$, then this is witnessed via a database $[\![L]\!]$, where L is a tree accepted by the automaton $\mathfrak{A}$ in Proposition 5.7. Since $\mathfrak{A}$ has exponentially many states, we can conclude that the trees accepted by $\mathfrak{A}$ have size at most triple-exponential. This is because $\mathfrak{A}$ can be transformed into a non-deterministic tree automaton with double-exponentially many states, which in turn accepts trees of size at most triple-exponential. Therefore, $[\![L]\!]$ is a triple-exponentially-sized database. It is open whether this is an optimal upper bound.

6 FRONTIER-GUARDEDNESS

We proceed to show that Theorem 5.1 can be extended to OMQs based on frontier-guarded tgds:

Theorem 6.1. $\mathsf{Cont}((\mathbb{FG}, \mathbb{CQ}))$ *is complete for* 2ExpTime. *The lower bound holds even if the arity of the schema is fixed, and the tgds are without constants.*

As for $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$, the lower bound is inherited from [10]. The rest of this section is devoted to establish the desired upper bound. As in Section 5, we focus on constant-free tgds and constant-free BCQs, but the result can be extended to the general case. In fact, in order to simplify our analysis even more, let us observe that for containment purposes under OMQs based on frontier-guarded tgds, it suffices to focus on Boolean atomic queries, i.e., BCQs consisting of a single atom; we refer to this class of queries as $\mathbb{BAQ}$. The reason for this is because a BCQ can be seen as frontier-guarded tgd. More precisely, an OMQ $(\mathbf{S}, \Sigma, q) \in (\mathbb{FG}, \mathbb{BCQ})$ can be equivalently rewritten as the OMQ $(\mathbf{S}, \Sigma \cup \{q \rightarrow \mathrm{Ans}\}, \mathrm{Ans}) \in (\mathbb{FG}, \mathbb{BAQ})$, where each variable in $q \rightarrow \mathrm{Ans}$ is interpreted as a universally quantified variable. From the above discussion, it suffices to show that $\mathsf{Cont}((\mathbb{FG}, \mathbb{BAQ}))$ is in 2ExpTime.

Our goal is to provide a reduction from $\mathsf{Cont}((\mathbb{FG}, \mathbb{BAQ}))$ to $\mathsf{Cont}((\mathbb{G}, \mathbb{BAQ}))$, and then apply Theorem 5.1. The main ingredients of our reduction are the following:

(1) A query $Q \in (\mathbb{FG}, \mathbb{BAQ})$ can be rewritten as a query $Q' \in (\mathbb{G}, \mathbb{BAQ})$ in such a way that Q and Q' are equivalent over *acyclic* databases, i.e., databases that have a $[\varnothing]$-guarded tree decomposition.
(2) We observe that for $(\mathbb{G}, \mathbb{BAQ})$ we can characterize satisfiability via acyclic databases. In other words, if there exists a database that satisfies a $(\mathbb{G}, \mathbb{BAQ})$ query Q, then Q is satisfied by an acyclic database.

Let us make the above statements more formal. The translation of $(\mathbb{FG}, \mathbb{BAQ})$ into a $(\mathbb{G}, \mathbb{BAQ})$ relies on the notion of *treeification* (see, e.g., [4, 5]), and is inspired by a construction given in [5] that translates guarded negation fixed point sentences into guarded fixed point sentences. Our goal is to transform a frontier-guarded tgd into a set of guarded tgds by treeifying the body of the former. In fact, the treefication procedure will first transform a tgd-body, which is essentially a CQ, to a set of *strictly acyclic* CQs, i.e., CQs that are acyclic and have an atom that contains its free variables. Then each strictly acyclic query will give rise to linearly many guarded tgds. Let us now recall treefications.

Consider a CQ $q(\bar{x})$ over a schema $\mathbf{S}$. The $\mathbf{T}$-*treefication* of $q(\bar{x})$, where $\mathbf{T} \supseteq \mathbf{S}$, is the set $\Lambda_q^{\mathbf{T}}$ of all strictly acyclic CQs $q'(\bar{x})$ over $\mathbf{T}$ of size at most $3|q|$ such that (i) $q' \subseteq q$, and (ii) is minimal, i.e., by removing an atom would render into a CQ that is not strictly acyclic or $q' \not\subseteq q$. The set $\Lambda_q^{\mathbf{T}}$ can be seen as the UCQ $\Lambda_q^{\mathbf{T}}(\bar{x})$ defined as the disjunction of all CQs contained in $\Lambda_q^{\mathbf{T}}$. Notice that the query $q(\bar{x})$ is in general not equivalent to its $\mathbf{T}$-treeification. However, $q(\bar{x})$ and $\Lambda_q^{\mathbf{T}}(\bar{x})$ are equivalent over acyclic $\mathbf{T}$-databases [4, 5].

We are now ready to explain how a frontier-guarded OMQ is transformed into a guarded OMQ. Consider a frontier-guarded tgd $\tau\colon \phi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z})$ and a schema $\mathbf{T}$. Let $f_C^{\mathbf{T}}(\tau)$, where C is a predicate not in $\mathbf{T}$, be the set of tgds

$$\left\{ q(\bar{x}) \rightarrow \exists \bar{z}\, \psi(\bar{x}, \bar{z}) \mid q(\bar{x}) \in \Lambda^{\mathbf{T} \cup \{C\}}_{\exists \bar{y}\, \phi(\bar{x}, \bar{y})} \right\}.$$

Notice that the tgds in $f_C^{\mathbf{T}}(\tau)$ may not be guarded. However, by construction, their bodies are strictly acyclic CQs, and this allows us to rewrite each tgd in $f_C^{\mathbf{T}}(\tau)$ into linearly many guarded tgds, which we denote by $g_C^{\mathbf{T}}(\tau)$. Given an OMQ $Q = (\mathbf{S}, \Sigma, q) \in (\mathbb{FG}, \mathbb{BAQ})$, let

$$g_C(Q) = \left(\mathbf{S} \cup \{C\}, \bigcup_{\tau \in \Sigma} g_C^{\mathbf{S} \cup sch(\Sigma)}(\tau), q \right) \in (\mathbb{G}, \mathbb{BAQ}),$$

where C is an auxiliary predicate not in $\mathbf{S} \cup sch(\Sigma)$. This completes the translation from frontier-guarded to guarded OMQs. We can show the following crucial lemma, which actually formalizes the first intuitive statement given above. Given a schema $\mathbf{S}$ and a predicate $C/n \notin \mathbf{S}$, for brevity, we write $\mathbf{S}_C$ for $\mathbf{S} \cup \{C\}$. Given an $\mathbf{S}$-database D, let D_C be the $\mathbf{S}_C$-database $D \cup \{C(\bar{t}) \mid \bar{t} \in dom(D)^n\}$. By the *width* of an OMQ Q, written $width(Q)$, we mean the maximum number of variables in the body of a tgd of Q.

Lemma 6.2. *Let $Q = (\mathbf{S}, \Sigma, q) \in (\mathbb{FG}, \mathbb{BAQ})$, and $Q' = g_C(Q)$, where $C \notin \mathbf{S}$ has arity at least $width(Q)$. Then:*

(1) For each acyclic $\mathbf{S}_C$-database D, $Q(D) = Q'(D)$.
(2) For each $\mathbf{S}$-database D, $Q(D) \neq \varnothing \Rightarrow Q'(D_C) \neq \varnothing$.

Let us now formalize the second intuitive statement given above. Actually, the next result is implicit in the proof of Proposition 5.3,

which establishes that non-containment for $(\mathbb{G}, \mathbb{CQ})$ is witnessed via a tree-like database. We write $I \to D$ for the fact that the instance I can be mapped via a homomorphism to the database D.

Lemma 6.3. *Consider an* S*-database D, and an OMQ $Q = (\mathbf{S}, \Sigma, q) \in (\mathbb{G}, \mathbb{BAQ})$. If $Q(D) \neq \varnothing$, then there is a finite acyclic* S*-instance I such that $Q(I) \neq \varnothing$ and $I \to D$.*

Having the above lemmas in place, it is easy to show that $g_C(\cdot)$ provides a reduction from $\mathsf{Cont}((\mathbb{FG}, \mathbb{BAQ}))$ to $\mathsf{Cont}((\mathbb{G}, \mathbb{BAQ}))$, if the arity of C is sufficiently large.

Proposition 6.4. *Let $Q_i = (\mathbf{S}, \Sigma_i, q_i) \in (\mathbb{G}, \mathbb{BAQ})$, for $i \in \{1, 2\}$, and consider a predicate $C \notin (\mathbf{S} \cup sch(\Sigma_1) \cup sch(\Sigma_2))$ that has arity $\max_{i \in \{1,2\}}\{width(Q_i)\}$. Then,*

$$Q_1 \subseteq Q_2 \iff g_C(Q_1) \subseteq g_C(Q_2).$$

Proof (sketch). Let $Q_i' = g_C(Q_i)$, for $i \in \{1, 2\}$. Assume that $Q_1 \not\subseteq Q_2$. This implies that there exists an S-database D such that $Q_1(D) \neq \varnothing$ and $Q_2(D) = \varnothing$. By Lemma 6.2, $Q_1'(D_C) \neq \varnothing$, and thus, by Lemma 6.3, there exists a finite acyclic $\mathbf{S}_C$-instance I such that $Q_1'(I) \neq \varnothing$ and $I \to D_C$. Since $Q_2(D_C) = Q_2(D) = \varnothing$, and Q_2 is closed under homomorphisms, $Q_2(I) = \varnothing$. Consequently, by Lemma 6.2, $Q_2'(I) = \varnothing$, which implies that $Q_1' \not\subseteq Q_2'$. The other direction can be shown analogously. □

The above proposition provides the desired reduction from $\mathsf{Cont}((\mathbb{FG}, \mathbb{BAQ}))$ to $\mathsf{Cont}((\mathbb{G}, \mathbb{BAQ}))$, which allows us to apply the algorithm for $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$, devised in Section 5. However, it should not be overlooked that this reduction takes exponential time due to the treefication procedure. In fact, for a CQ q, $|\Lambda_q^{\mathbf{T}}| \leq |\mathbf{T}|^{O(|q|)}(|q|w)^{O(|q|w)}$, where w is the maximum arity over all predicates of $\mathbf{T}$ [4, 5]. Nevertheless, since the reduction provided by Proposition 6.4 increases the arity of the schema only polynomially, while the algorithm for $\mathsf{Cont}((\mathbb{G}, \mathbb{BAQ}))$ provided by Theorem 5.1 is double-exponential only on the arity of the underlying schema, we obtain that $\mathsf{Cont}((\mathbb{FG}, \mathbb{BAQ}))$ is feasible in double-exponential time, as needed.

We conclude this section by noticing that, as for guarded OMQs, we get a small witness property for $\mathsf{Cont}((\mathbb{FG}, \mathbb{CQ}))$, which states that non-containment is witnessed via a triple-exponentially-sized database. More precisely, $Q_1 \not\subseteq Q_2$ implies $g_C(Q_1) \not\subseteq g_C(Q_2)$, and we can show that the latter non-containment is witnessed via a triple-exponentially-sized acyclic database D. Since, by Lemma 6.2, Q_i and $g_C(Q_i)$, for $i \in \{1, 2\}$, are equivalent over acyclic databases, D is a witness for $Q_1 \not\subseteq Q_2$.

7 COMBINING LANGUAGES

In the previous three sections, we studied the containment problem relative to a language $\mathbb{O}$, i.e., both OMQs fall in $\mathbb{O}$. However, it is natural to consider the version of the problem where the involved OMQs fall in different languages. This is the goal of this section. Our analysis proceeds by considering the two cases where the left-hand side (LHS) query falls in a UCQ rewritable OMQ language, or it is guarded. Notice that the two cases where the LHS query is guarded or frontier-guarded behave in the same way. Thus, for brevity, we only focus on the former case.

7.1 The LHS Query is UCQ Rewritable

As an immediate corollary of Theorem 4.3 we obtain the following result: $\mathsf{Cont}((\mathbb{C}_1, \mathbb{CQ}), (\mathbb{C}_2, \mathbb{CQ}))$, for $\mathbb{C}_1 \neq \mathbb{C}_2$, $\mathbb{C}_1 \in \{\mathbb{L}, \mathbb{NR}, \mathbb{S}\}$ and $\mathbb{C}_2 \in \{\mathbb{L}, \mathbb{NR}, \mathbb{S}, \mathbb{FG}, \mathbb{G}\}$, is decidable. By exploiting the algorithm underlying Theorem 4.3, we establish optimal upper bounds for all the problems at hand with the only exception of $\mathsf{Cont}((\mathbb{S}, \mathbb{CQ}), (\mathbb{NR}, \mathbb{CQ}))$. For the latter, we obtain a coNExpTime$^{\text{NP}}$ upper bound, by providing a similar analysis as for $\mathsf{Cont}((\mathbb{NR}, \mathbb{CQ}))$, while a NExpTime lower bound is inherited from query evaluation by exploiting Proposition 3.1. It is rather tedious to go through all the containment problems in question and explain in details how the exact upper bounds are obtained.[5]

Regarding the matching lower bounds, in most of the cases they are inherited from query evaluation by exploiting Propositions 3.1 and 3.2. There are, however, some exceptions:

- $\mathsf{Cont}((\mathbb{S}, \mathbb{CQ}), (\mathbb{L}, \mathbb{CQ}))$ in the case of unbounded arity, where the problem is coNExpTime-hard, even for sets of tgds that use only two constants. This is shown by a reduction from the standard tiling problem for the exponential grid $2^n \times 2^n$.
- $\mathsf{Cont}((\mathbb{L}, \mathbb{CQ}), (\mathbb{S}, \mathbb{CQ}))$ and $\mathsf{Cont}((\mathbb{S}, \mathbb{CQ}), (\mathbb{L}, \mathbb{CQ}))$ in the case of bounded arity, where both problems are Π_2^P-hard even for constant-free tgds; implicit in [11].

7.2 The LHS Query is Guarded

We proceed with the case where the LHS query is guarded, and we show the following result:

Theorem 7.1. *The problem $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), (\mathbb{C}, \mathbb{CQ}))$ is C-complete, where:*

$$C = \begin{cases} \text{2ExpTime}, & \text{if } \mathbb{C} \in \{\mathbb{L}, \mathbb{S}\}, \\ \text{3ExpTime}, & \text{if } \mathbb{C} = \mathbb{NR}. \end{cases}$$

The lower bounds hold even if the arity of the schema is fixed. Moreover, for $\mathbb{C} = \mathbb{L}$ (resp., $\mathbb{C} \in \{\mathbb{NR}, \mathbb{S}\}$) it holds even for tgds with one constant (resp., without constants).

Upper bounds. The 2ExpTime membership when $\mathbb{C} = \mathbb{L}$ is an immediate corollary of Theorem 5.1. This is not true when $\mathbb{C} \in \{\mathbb{NR}, \mathbb{S}\}$ since the right-hand side query is not guarded. But in this case, since $(\mathbb{NR}, \mathbb{CQ})$ and $(\mathbb{S}, \mathbb{CQ})$ are UCQ rewritable, one can rewrite the right-hand side query as a UCQ, and then apply the machinery developed in Section 5 for solving $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}))$. More precisely, given OMQs $Q_1 \in (\mathbb{G}, \mathbb{CQ})$ and $Q_2 \in (\mathbb{C}, \mathbb{CQ})$, where $\mathbb{C} \in \{\mathbb{NR}, \mathbb{S}\}$, $Q_1 \subseteq Q_2$ iff $Q_1 \subseteq q$, where q is a UCQ rewriting of Q_2. Thus, an immediate decision procedure, which exploits the algorithm XRewrite, is the following:

(1) Let $q = \mathsf{XRewrite}(Q_2)$;
(2) For each $q' \in q$: if $Q_1 \subseteq q'$, then proceed; otherwise, reject; and
(3) Accept.

The above procedure runs in triple-exponential time. The first step is feasible in double-exponential time [23]. Now, for a single CQ $q' \in q$ (which is a guarded OMQ with an empty set of tgds) the

[5] There are twenty-four different cases obtained by considering all the possible pairs $(\mathbb{O}_1, \mathbb{O}_2)$ of OMQ languages, where $\mathbb{O}_1 \neq \mathbb{O}_2$ and $\mathbb{O}_1$ is UCQ rewritable, and the two cases whether the arity of the schema is fixed or not.

check whether $Q_1 \subseteq q'$ can be done by using the machinery developed in Section 5, which reduces our problem to checking whether the language of a 2WAPA $\mathfrak{A}$ is empty. However, it should not be forgotten that q' is of exponential size, and thus, the automaton $\mathfrak{A}$ has double-exponentially many states. This in turn implies that checking whether $\mathcal{L}(\mathfrak{A}) = \varnothing$ is in 3ExpTime, as claimed.

Although the above algorithm establishes an optimal upper bound for non-recursive OMQs, a more refined analysis is needed for sticky OMQs. In fact, we need a more refined complexity analysis for the problem $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), \mathbb{UCQ})$, that is, to decide whether a guarded OMQ is contained in a UCQ. To this end, we provide an automata construction different from the one employed in Section 5, which allows us to establish a refined complexity upper bound for the problem in question. Consider a $(\mathbb{G}, \mathbb{CQ})$ query Q, and a UCQ $q = q_1 \vee \cdots \vee q_n$. As usual, we write $\|Q\|$ and $\|q_i\|$ for the number of symbols that occur in Q and q_i, respectively, and we write $var_{\geq 2}(q_i)$ for the set of variables that appear in more than one atom of q_i. By exploiting our new automata-based procedure, we show that the problem of checking if $Q \subseteq q$ is feasible in double-exponential time in $(\|Q\| + \max_{1 \leq i \leq n}\{|var_{\geq 2}(q_i)|\})$, exponential time in $\max_{1 \leq i \leq n}\{\|q_i\|\}$, and polynomial time in n.

This result allows us to show that the above procedure establishes 2ExpTime-membership when the right-hand side OMQ is sticky. But first we need to recall the following key properties of the UCQ rewriting $q = \mathsf{XRewrite}(Q_2)$, constructed during the first step of the algorithm:

(1) q consists of double-exponentially many CQs,
(2) each CQ of q is of exponential size, and
(3) for each $q' \in q$, $var_{\geq 2}(q')$ is a subset of the variables of the original CQ that appears in Q_2.

By combining these key properties with the complexity analysis performed above, it is now straightforward to show that $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), (\mathbb{S}, \mathbb{CQ}))$ is in 2ExpTime.

Lower Bounds. We establish matching lower bounds by refining techniques from [19], where it is shown that containment of Datalog in UCQ is 2ExpTime-complete, while containment of Datalog in non-recursive Datalog is 3ExpTime-complete; the lower bounds hold for fixed-arity predicates, and constant-free rules. Interestingly, the LHS query can be transformed into a Datalog query such that each rule has a body-atom that contains all the variables, i.e., is guarded. This is achieved by increasing the arity of some predicates in order to have enough positions for all the body-variables. However, for each rule, the number of unguarded variables that we need to guard is constant, and thus, the arity of the schema remains constant. We conclude that $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), (\mathbb{NR}, \mathbb{CQ}))$ is 3ExpTime-hard. Moreover, containment of guarded OMQs in UCQs is 2ExpTime-hard, which in turn allows us to show, by exploiting the construction underlying Proposition 3.5, that $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), (\mathbb{L}, \mathbb{CQ}))$ is 2ExpTime-hard, even if the set of linear tgds uses only one constant, while $\mathsf{Cont}((\mathbb{G}, \mathbb{CQ}), (\mathbb{S}, \mathbb{CQ}))$ is 2ExpTime-hard, even for tgds without constants.

8 CONCLUSIONS

We have concentrated on the fundamental problem of containment for OMQ languages based on the main decidable classes of tgds, and we have developed specially tailored techniques that allow us to obtain a relatively complete picture for the complexity of the problem at hand. Our main conclusion is that for most of the OMQ languages in question, the containment problem is harder (under widely accepted complexity assumptions) than query evaluation.

Acknowledgements. Barceló is funded by the Millennium Institute for Foundational Research on Data and Fondecyt grant 1170109. Berger is funded by the Austrian Science Fund (FWF), project number W1255-N23 and DOC fellowship of the Austrian Academy of Sciences. Pieris is funded by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems - Principles and Architecture".

REFERENCES

[1] Marcelo Arenas, Richard Hull, Wim Martens, Tova Milo, and Thomas Schwentick. 2016. Foundations of Data Management (Dagstuhl Perspectives Workshop 16151). *Dagstuhl Reports* 6, 4 (2016), 39–56.
[2] Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. 2011. On rules with existential variables: Walking the decidability line. *Artif. Intell.* 175, 9-10 (2011), 1620–1654.
[3] Vince Bárány, Michael Benedikt, and Balder ten Cate. 2013. Rewriting Guarded Negation Queries. In *MFCS*. 98–110.
[4] Vince Bárány, Georg Gottlob, and Martin Otto. 2014. Querying the Guarded Fragment. *Logical Methods in Computer Science* 10, 2 (2014).
[5] Vince Bárány, Balder ten Cate, and Luc Segoufin. 2015. Guarded Negation. *J. ACM* 62, 3 (2015), 22:1–22:26.
[6] Catriel Beeri and Moshe Y. Vardi. 1981. The Implication Problem for Data Dependencies. In *ICALP*. 73–85.
[7] Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. 2016. A Step Up in Expressiveness of Decidable Fixpoint Logics. In *LICS*. 817–826.
[8] Michael Benedikt and Georg Gottlob. 2010. The Impact of Virtual Views on Containment. *PVLDB* 3, 1 (2010), 297–308.
[9] Gerald Berger and Andreas Pieris. 2016. Ontology-Mediated Queries Distributing over Components. In *IJCAI*. 943–949.
[10] Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. 2016. First Order-Rewritability and Containment of Conjunctive Queries in Horn Description Logics. In *IJCAI*. 965–971.
[11] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. 2012. Query Containment in Description Logics Reconsidered. In *KR*.
[12] Meghyn Bienvenu, Balder ten Cate, Carsten Lutz, and Frank Wolter. 2013. Ontology-based data access: a study through disjunctive datalog, CSP, and MMSNP. In *PODS*. 213–224.
[13] Pierre Bourhis, Markus Krötzsch, and Sebastian Rudolph. 2015. Reasonable Highly Expressive Query Languages. In *IJCAI*. 2826–2832.
[14] Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. 2016. Guarded-Based Disjunctive Tuple-Generating Dependencies. *ACM Trans. Database Syst.* 41, 4 (2016).
[15] Andrea Calì, Georg Gottlob, and Michael Kifer. 2013. Taming the Infinite Chase: Query Answering under Expressive Relational Constraints. *J. Artif. Intell. Res.* 48 (2013), 115–174.
[16] Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. 2012. A general Datalog-based framework for tractable query answering over ontologies. *J. Web Sem.* 14 (2012), 57–83.
[17] Andrea Calì, Georg Gottlob, and Andreas Pieris. 2012. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.* 193 (2012), 87–128.
[18] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*. 77–90.
[19] Surajit Chaudhuri and Moshe Y. Vardi. 1997. On the Equivalence of Recursive and Nonrecursive Datalog Programs. *J. Comput. Syst. Sci.* 54, 1 (1997), 61–78.
[20] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. 1988. Decidable Optimization Problems for Database Logic Programs (Preliminary Report). In *STOC*. 477–490.
[21] Thomas Eiter, Thomas Lukasiewicz, and Livia Predoiu. 2016. Generalized Consistent Query Answering under Existential Rules. In *KR*. 359–368.
[22] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. 2005. Data exchange: Semantics and query answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.
[23] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2014. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.* 39, 3 (2014), 25:1–25:46.

[24] Georg Gottlob and Christos H. Papadimitriou. 2003. On the complexity of single-rule datalog queries. *Inf. Comput.* 183, 1 (2003), 104–122.
[25] Georg Gottlob, Andreas Pieris, and Mantas Simkus. 2018. The Impact of Active Domain Predicates on Guarded Existential Rules. *Fundam. Inform.* 159, 1-2 (2018), 123–146.
[26] Georg Gottlob, Sebastian Rudolph, and Mantas Simkus. 2014. Expressiveness of guarded existential rule languages. In *PODS*. 27–38.
[27] Erich Grädel and Igor Walukiewicz. 1999. Guarded Fixed Point Logic. In *LICS*. 45–54.
[28] Lane A. Hemachandra. 1989. The Strong Exponential Hierarchy Collapses. *J. Comput. Syst. Sci.* 39, 3 (1989), 299–322.
[29] David S. Johnson and Anthony C. Klug. 1984. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *J. Comput. Syst. Sci.* 28, 1 (1984), 167–189.
[30] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. 2015. From Classical to Consistent Query Answering under Existential Rules. In *AAAI*. 1546–1552.
[31] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. 1979. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.* 4, 4 (1979), 455–469.
[32] Antonella Poggi, Domenico Lembo, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. 2008. Linking Data to Ontologies. *J. Data Semantics* 10 (2008), 133–173.
[33] Yehoshua Sagiv and Mihalis Yannakakis. 1980. Equivalences Among Relational Expressions with the Union and Difference Operators. *J. ACM* 27, 4 (1980), 633–655.
[34] Oded Shmueli. 1993. Equivalence of DATALOG Queries is Undecidable. *J. Log. Program.* 15, 3 (1993), 231–241.
[35] Thomas Wilke. 2001. Alternating tree automata, parity games, and modal μ-calculus. *Bull. Belg. Math. Soc. Simon Stevin* 8, 2 (2001), 359–391.