Guarded Ontology-Mediated Queries

Chapter 1
Guarded Ontology-Mediated Queries

Pablo Barceló, Gerald Berger, Georg Gottlob, and Andreas Pieris

Abstract We concentrate on ontology-mediated queries (OMQs) expressed using guarded Datalog$^\exists$ and conjunctive queries. Guarded Datalog$^\exists$ is a rule-based knowledge representation formalism inspired by the guarded fragment of first-order logic, while conjunctive queries represent a prominent database query language that lies at the core of relational calculus (i.e., first-order queries). For such guarded OMQs we discuss three main algorithmic tasks: query evaluation, query containment, and first-order rewritability. The first one is the task of computing the answer to an OMQ over an input database. The second one is the task of checking whether the answer to an OMQ is contained in the answer of some other OMQ on every input database. The third one asks whether an OMQ can be equivalently rewritten as a first-order query. For query evaluation, we explain how classical results on the satisfiability problem for the guarded fragment of first-order logic can be applied. For query containment, we discuss how tree automata techniques can be used. Finally, for first-order rewritability, we explain how techniques based on a more sophisticated automata model, known as cost automata, can be exploited.

Pablo Barceló
Department of Computer Science, University of Chile & IMFD Chile
e-mail: pbarcelo@dcc.uchile.cl

Gerald Berger
Institute of Logic and Computation, TU Wien
e-mail: gberger@dbai.tuwien.ac.at

Georg Gottlob
Department of Computer Science, University of Oxford & Institute of Logic and Computation, TU Wien
e-mail: georg.gottlob@cs.ox.ac.uk

Andreas Pieris
School of Informatics, University of Edinburgh
e-mail: apieris@inf.ed.ac.uk

1.1 Introduction

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* [3]. A crucial problem in this field is *ontology-based data access* (OBDA) [31], 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, the ontology O and the user query q, which is typically a *conjunctive query*, can be seen as two components of one composite query $Q = (\mathbf{S}, O, q)$, dubbed *ontology-mediated query* (OMQ); $\mathbf{S}$ is the *data schema*, indicating that Q will be posed on databases over $\mathbf{S}$ [14]. The main tasks that are of special interest in this setting are as follows:

Query Evaluation. Given an OMQ $Q = (\mathbf{S}, O, q)$, a database $\mathfrak{D}$ over the schema $\mathbf{S}$, and a candidate answer $\bar{a}$, which is a tuple of constants from the domain of $\mathfrak{D}$, the question is whether $\bar{a}$ belongs to the evaluation of q over every extension of $\mathfrak{D}$ that satisfies O. In other words, is $\bar{a}$ a *certain answer* for Q over $\mathfrak{D}$? The set of certain answers for Q over $\mathfrak{D}$ is denoted $Q(\mathfrak{D})$.

Query Containment. Given two OMQs Q_1 and Q_2, both with data schema $\mathbf{S}$, the question is whether Q_1 is *contained* in Q_2, i.e., $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$ for every database $\mathfrak{D}$ over $\mathbf{S}$. This is a crucial static analysis task (i.e., no database is involved) with applications in query optimization. Whenever we try to optimize an OMQ Q and get some other one Q' that is easier to evaluate, we have to ensure that Q and Q' are equivalent, i.e., they return the same answer over all databases. This boils down to check that Q is contained in Q' and vice versa.

First-Order Rewritability. Given an OMQ Q with data schema $\mathbf{S}$, the question is whether Q can be rewritten as a first-order query φ_Q that returns the same answer on every input database over the schema $\mathbf{S}$. This is another important static analysis task, which allows us to check whether an OMQ can be executed via standard database management systems (DBMSs), which are highly optimized for evaluating first-order queries. Notice that standard DBMSs are unaware of ontologies, and thus we cannot blindly pass to such systems an OMQ.

1.1.1 Rule-Based Ontology-Mediated Queries

While in the OMQ setting described above *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 *Datalog*$^{\exists}$ *rules*, a.k.a. *tuple-generating dependencies* and *existential rules*, of the form

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

where φ and ψ are conjunctions of atoms. Notice that such rules extend plain Datalog rules with the feature of existential quantification (hence the name Datalog$^{\exists}$), which is crucial for knowledge representation purposes as it allows us to invent new objects that are not already present in the input database.

Unfortunately, the use of Datalog$^{\exists}$ as an ontology language, without posing any additional syntactic restrictions, leads to the undecidability of all the algorithmic tasks for OMQs described above. The undecidability of query evaluation is implicit in [9], where the implication problem for database dependencies is studied. A stronger result of this kind can be found in [16], where it is shown that query evaluation is undecidable even when the ontology and the conjunctive query are fixed. The undecidability of query containment is immediately inherited from the fact that query containment for plain Datalog queries (without existential quantification) is undecidable [35]. Finally, the undecidability of first-order rewritability is again inherited from the fact that the same problem for plain Datalog queries is undecidable. In fact, we know that a Datalog query is first-order rewritable iff it has bounded recursion [1], while the problem of checking whether a Datalog query has bounded recursion is undecidable [24].

The above negative results led to a flurry of activity during the last decade for identifying syntactic restrictions on Datalog$^{\exists}$ rules that make the algorithmic tasks in question, especially query evaluation, decidable. Several decidable fragments have been proposed in the literature based on different syntactic restrictions; see, e.g., [4, 19, 30] – the list is by no means exhaustive.

1.1.2 Guardedness to the Rescue

A prime example of a well-behaved formalism is *guarded* Datalog$^{\exists}$ [16], which has been inspired by the guarded fragment of first-order logic (GFO) introduced by Andréka, Németi, and van Benthem in [2]. A Datalog$^{\exists}$ rule is *guarded* if the left-hand side of the implication has an atom, called a guard, that contains all the universally quantified variables. Guarded Datalog$^{\exists}$ is a member of a broader family of knowledge representation formalisms, known as *Datalog*$^{\pm}$ [18]. Datalog$^{\pm}$ languages extend Datalog with useful features such as existential quantification, equality, negation, etc., and at the same time restrict the syntax (such as guardedness) in order to guarantee decidability of the main tasks. Hence, the symbol '$+$' refers to the additional features, while the symbol '$-$' refers to the syntactic restrictions.

Just as the robustness and the nice algorithmic properties of GFO can be attributed to the *tree model property* [2] (i.e., satisfiable GFO-sentences always have a "tree-like" model, that is, a model of bounded tree-width), the reason why query evaluation for guarded OMQs, i.e., OMQs where the ontology consists of guarded Datalog$^{\exists}$ rules, is decidable, is because we can focus on "tree-like" *universal* models; for more details see [16]. In fact, guarded Datalog$^{\exists}$ (plus guarded denial constraints) is essentially a normal form for the Horn fragment of GFO (cf. [29] for more details on well-behaved fragments of GFO regarding query evaluation). It is

worth mentioning that the tree model property of the guarded fragment renders decidable even the problem of query evaluation for OMQs where the ontology is a GFO-sentence [5], while this fails for other decidable fragments of first-order logic, most notably the two-variable fragment [32]. The goal of this chapter is to discuss the algorithmic tasks introduced above for guarded OMQs, and explain how worst-case optimal complexity results can be established.

Roadmap. After giving some basic preliminaries in Section 1.2, we focus in Section 1.3 on query evaluation, and explain how classical results on the satisfiability problem for GFO can be applied in order to show that the problem in question is in 2ExpTime. This is done by relying on a technique known as *treefication* introduced in [5]. In Section 1.4, we concentrate on query containment, and we discuss how tree automata techniques can be used in order to establish a 2ExpTime upper bound. This section is based on recent results on query containment from [8]. Then, in Section 1.5, we consider first-order rewritability, and explain how techniques based on a more sophisticated automata model, known as cost automata, can be exploited in order to obtain a 2ExpTime upper bound. This section is based on recent results on first-order rewritability from [7]. Finally, in Section 1.6, we discuss the above three algorithmic problems for guarded OMQs under the lenses of finite models, i.e., when the evaluation of an OMQ $Q = (\mathbf{S}, O, q)$ over a database $\mathfrak{D}$ is defined by considering only the *finite* extensions of $\mathfrak{D}$ that satisfy O, denoted $Q_{fin}(\mathfrak{D})$. We present a deep result, which is implicit in [5], that establishes the following: for every guarded OMQ Q, with data schema $\mathbf{S}$, and database $\mathfrak{D}$ over $\mathbf{S}$, $Q(\mathfrak{D}) = Q_{fin}(\mathfrak{D})$. The latter immediately implies that containment and first-order rewritability for guarded OMQs are invariant with respect to whether we consider all the extensions of the database that satisfy the ontology, or only the finite ones.

1.2 Preliminaries

Basics. Let $\mathbf{C}$, $\mathbf{N}$, and $\mathbf{V}$ be disjoint, countably infinite sets of *constants*, *(labeled) nulls*, and *variables*, respectively. We adopt the *unique name assumption*, i.e., different constants represent different values. A *schema* $\mathbf{S}$ is a finite set of *relation symbols*, each having an associated (non-negative) *arity*. The *width* of $\mathbf{S}$, denoted $\mathsf{wd}(\mathbf{S})$, is the maximum arity among all relation symbols of $\mathbf{S}$. We write R/n to indicate that the relation symbol R has arity $n \geq 0$. A *term* is either a constant, a null, or a variable. An *atom* over $\mathbf{S}$ (or simply $\mathbf{S}$-*atom*) is an expression of the form $R(t_1, \ldots, t_n)$, where $R \in \mathbf{S}$ is of arity n and $t_1, \ldots, t_n$ are terms. An $\mathbf{S}$-fact is an $\mathbf{S}$-atom whose arguments consist of constants only.

Databases. An *instance* over a schema $\mathbf{S}$ (or simply $\mathbf{S}$-*instance*) is a (possibly infinite) set of $\mathbf{S}$-atoms that contain constants and nulls as arguments only. A *database* over $\mathbf{S}$ (or simply $\mathbf{S}$-*database*) is a finite set of $\mathbf{S}$-facts, i.e., a finite $\mathbf{S}$-instance that contains constants only. The *active domain* of an instance $\mathfrak{J}$, denoted $\mathsf{adom}(\mathfrak{J})$, consists of all terms occurring in $\mathfrak{J}$.

A *tree decomposition* for an instance $\mathfrak{J}$ is a tuple $\delta = (\mathscr{T}, (X_v)_{v \in T})$, where $\mathscr{T}$ is a rooted tree whose set of nodes is T, and $(X_v)_{v \in T}$ is a collection of subsets of $\mathsf{adom}(\mathfrak{J})$, called *bags*, such that:

1. If $R(t_1, \ldots, t_n) \in \mathfrak{J}$, then there exists a $v \in T$ such that $\{t_1, \ldots, t_n\} \subseteq X_v$.
2. For all $a \in \mathsf{adom}(\mathfrak{J})$, the set $\{v \in T \mid a \in X_v\}$ induces a connected subtree of $\mathscr{T}$.

The tree decomposition δ is called *guarded* if, for every node $v \in T$, there exists an atom $R(t_1, \ldots, t_n) \in \mathfrak{J}$ such that $X_v \subseteq \{t_1, \ldots, t_n\}$. An instance is *acyclic* if it admits a guarded tree decomposition.

Conjunctive Queries. A *conjunctive query* (CQ) over $\mathbf{S}$ is a first-order formula

$$q(\bar{x}) \coloneqq \exists \bar{y} (\alpha_1(\bar{v}_1) \wedge \cdots \wedge \alpha_m(\bar{v}_m)),$$

where $\bar{x}$ and $\bar{y}$ are sequences of variables, each $\alpha_i(\bar{v}_i)$ is an $\mathbf{S}$-atom that mentions only variables from $\bar{v}_i$, or an *equality atom* of the form $v_i^1 = v_i^2$, with $v_i^1, v_i^2 \in \bar{v}_i$, and $\bar{v}_i \subseteq \bar{x} \cup \bar{y}$, for each $i \in \{1, \ldots, m\}$.[1] The variables $\bar{x}$ are the *answer variables* of $q(\bar{x})$. If $\bar{x}$ is empty, then q is a *Boolean* conjunctive query (BCQ). We shall denote by $\mathsf{var}(q)$ the variables that occur in $q(\bar{x})$.

The evaluation of CQs over instances is defined in terms of homomorphisms. A *homomorphism* from q to an $\mathbf{S}$-instance $\mathfrak{J}$ is a mapping $h \colon \mathsf{var}(q) \to \mathsf{adom}(\mathfrak{J})$ such that, for each $i \in \{1, \ldots, m\}$, (i) $\alpha_i(h(\bar{v}_i)) \in \mathfrak{J}$ if $\alpha_i(\bar{v}_i)$ is an $\mathbf{S}$-atom, and (ii) $h(v_i^1) = h(v_i^2)$ if $\alpha(\bar{v}_i)$ is the equality atom $v_i^1 = v_i^2$. We write $\mathfrak{J} \models q(\bar{a})$ to indicate that there is a homomorphism h from q to $\mathfrak{J}$ such that $h(\bar{x}) = \bar{a}$; in case q is Boolean, we write $\mathfrak{J} \models q$. We also write $q(\mathfrak{J})$ for the set of tuples $\bar{a} \in \mathsf{adom}(\mathfrak{J})^{|\bar{x}|}$ such that $\mathfrak{J} \models q(\bar{a})$. Let CQ (resp., BCQ) be the class of all CQs (resp., BCQs).

Tuple-Generating Dependencies. A *tuple-generating dependency* (TGD)[2] is a first-order sentence of the form

$$\tau : \forall \bar{x} \forall \bar{y} \, (\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z} \, \psi(\bar{x}, \bar{z})),$$

where φ and ψ are conjunctions of atoms that mention only variables, called the *body* and *head* of τ, respectively. For brevity, we shall omit the preceding universal quantifiers and use comma instead of $\wedge$ for joining atoms. We assume that each variable of $\bar{x}$ is mentioned in at least one atom of ψ.

The TGD τ is logically equivalent to the sentence

$$\forall \bar{x} (q_\varphi(\bar{x}) \rightarrow q_\psi(\bar{x})),$$

where $q_\varphi(\bar{x})$ and $q_\psi(\bar{x})$ are the CQs $\exists \bar{y} \, \varphi(\bar{x}, \bar{y})$ and $\exists \bar{z} \, \psi(\bar{x}, \bar{z})$, respectively. An instance $\mathfrak{J}$ *satisfies* τ if $q_\varphi(\mathfrak{J}) \subseteq q_\psi(\mathfrak{J})$, while $\mathfrak{J}$ satisfies a set of TGDs O, denoted $\mathfrak{J} \models O$, if $\mathfrak{J} \models \tau'$ for every $\tau' \in O$. Let TGD be the class of finite sets of TGDs.

---

[1] For technical clarity, we assume that CQs do not mention constants from $\mathbf{C}$. However, all the results that we discuss can be extended to CQs with constants.

[2] For brevity, in the rest of the chapter we adopt the acronym TGD instead of the term Datalog$^\exists$.

A prominent class, which is of special interest for this chapter, is the class of *guarded* TGDs. A TGD τ is guarded if it has an atom in its body, called a *guard*, that contains all the body variables. Let G be the class of finite sets of guarded TGDs.

Ontology-Mediated Queries. An *ontology-mediated query* (OMQ) is a triple of the form $Q = (\mathbf{S}, O, q(\bar{x}))$, where $\mathbf{S}$ is a schema (the *data schema* of Q), O is a finite set of TGDs (the *ontology*), and $q(\bar{x})$ is a CQ over $\mathbf{S} \cup \mathsf{sig}(O)$, with $\mathsf{sig}(O)$ denoting the set of all relation symbols in O. We include the data schema $\mathbf{S}$ in the specification of Q in order to emphasize that Q is evaluated over $\mathbf{S}$-databases, even though O and $q(\bar{x})$ may use additional relation symbols. In fact, O can introduce relations that are not present in $\mathbf{S}$, which in turn allows us to enrich the schema of $q(\bar{x})$.

The semantics of Q is given in terms of certain answers. Let $\mathfrak{D}$ be an $\mathbf{S}$-database. The *certain answers* to $q(\bar{x})$ w.r.t. $\mathfrak{D}$ and O is the set of all tuples $\bar{a}$ of constants such that $(\mathfrak{D}, O) \models q(\bar{a})$, that is, for every instance $\mathfrak{J} \supseteq \mathfrak{D}$ that satisfies O it holds that $\mathfrak{J} \models q(\bar{a})$. We write $\mathfrak{D} \models Q(\bar{a})$ to indicate that $\bar{a}$ is a certain answer to $q(\bar{x})$ w.r.t. $\mathfrak{D}$ and O; in case q is Boolean, we write $\mathfrak{D} \models Q$. Moreover, we write $Q(\mathfrak{D})$ for the set of all $\bar{a}$ such that $\mathfrak{D} \models Q(\bar{a})$. Hence, Q can be seen as a function that maps $\mathbf{S}$-databases to sets of tuples over $\mathbf{C}$.

We write (O, Q) for the class of OMQs $(\mathbf{S}, O, q(\bar{x}))$, where O falls in the class $\mathsf{O} \subseteq \mathsf{TGD}$, and $q(\bar{x})$ in the class $\mathsf{Q} \subseteq \mathsf{CQ}$. We call the pair (O, Q) an *ontology-mediated query language*. In this chapter, we mainly deal with the OMQ language $(\mathsf{G}, \mathsf{CQ})$, which collects all the OMQs where the ontology consists of guarded TGDs.

Example 1. Let $Q = (\mathbf{S}, O, q(x)) \in (\mathsf{G}, \mathsf{CQ})$, where $\mathbf{S} = \{P/1, F/2\}$, and

$$O \coloneqq \{P(x) \rightarrow \exists y\,(F(y,x) \wedge P(y))\} \quad q(x) \coloneqq \exists y \exists z\,(P(x) \wedge F(y,x) \wedge F(x,z)).$$

Consider the $\mathbf{S}$-database $\mathfrak{D} \coloneqq \{P(a), F(a,b), P(b)\}$. It is easy to verify that $Q(\mathfrak{D}) = \{\langle a \rangle\}$. Intuitively, the ontology O states that every person has a father who is himself a person. The query $q(x)$ asks for all persons x that have a father, and that are fathers themselves. Notice that the database $\mathfrak{D}$ does not explicitly store the fact that a has a father. This is implicit information expressed by the ontology O. □

1.3 Query Evaluation

As already discussed in Section 1.1, one of the most important tasks for an ontology-mediated query language is query evaluation, which is defined as follows:

PROBLEM :	$\mathsf{Eval}(\mathsf{O}, \mathsf{Q})$
INPUT :	OMQ $Q = (\mathbf{S}, O, q(\bar{x})) \in (\mathsf{O}, \mathsf{Q})$, $\mathbf{S}$-database $\mathfrak{D}$, $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{\lvert\bar{x}\rvert}$.
QUESTION :	Is it the case that $\bar{a} \in Q(\mathfrak{D})$?

It is well-known that $\mathsf{Eval}(\mathsf{TGD}, \mathsf{CQ})$ is undecidable; implicit in [9]. However, if we focus on $(\mathsf{G}, \mathsf{CQ})$, then the problem becomes decidable. In fact:

Theorem 1. $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$ *is* 2EXPTIME*-complete.*

The above result has been explicitly established in [16]: the upper bound is shown via a sophisticated alternating algorithm that uses exponential space, while the lower bound is shown by simulating the behavior of an alternating exponential space Turing machine. An alternative way to establish the 2EXPTIME upper bound of Theorem 1 is by a reduction to the satisfiability problem of the guarded fragment of first-order logic, which in turn exploits a technique known as treeification [5]. In what follows, we discuss the latter approach that relies on the fact that satisfiability for the guarded fragment of first-order logic is feasible in double exponential time, as shown in [27]. The goal is to reduce $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$ to the problem of deciding whether a guarded sentence is unsatisfiable.

Recall that the guarded fragment of first-order logic, introduced by Andréka, Németi and van Benthem in [2], is a collection of first-order formulas with some syntactic restrictions on quantification patterns, which is analogous to the relativized nature of modal logic. We write $\mathsf{GF}[\mathbf{T}]$, where $\mathbf{T}$ is a schema, for the smallest set of formulas that (i) contains all $\mathbf{T}$-atoms (without constants and nulls) and equalities among variables; (ii) is closed under $\neg$, $\wedge$, $\vee$, $\rightarrow$; and (iii) if α is an atom containing all the variables of $(\bar{x}\cup\bar{y})$, and $\varphi \in \mathsf{GF}[\mathbf{T}]$ with free variables in $(\bar{x}\cup\bar{y})$, then $\forall\bar{x}(\alpha \rightarrow \varphi)$ and $\exists\bar{x}(\alpha \wedge \varphi)$ belong to $\mathsf{GF}[\mathbf{T}]$ as well; α is the *guard* of the quantifier. It has been shown in [2] that satisfiability for guarded sentences is decidable, while Grädel proved in [27] that it is actually 2EXPTIME-complete.

1.3.1 From $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$ *to Satisfiability for the Guarded Fragment*

As said above, the goal is to reduce $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$ to the problem of deciding whether a guarded sentence is unsatisfiable. Consider an instance of $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$, i.e., an OMQ $Q = (\mathbf{S}, O, q(\bar{x})) \in (\mathsf{G},\mathsf{CQ})$, an $\mathbf{S}$-database $\mathfrak{D}$, and a tuple of constants $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. We define the first-order sentence

$$\varphi_{Q,\mathfrak{D},\bar{a}} \coloneqq \bigwedge_{\alpha\in\mathfrak{D}} \alpha \wedge \bigwedge_{\tau\in O} \tau \wedge \neg q(\bar{a}),$$

where $q(\bar{a})$ is the sentence obtained from $q(\bar{x})$ after instantiating the variables $\bar{x}$ with the constants $\bar{a}$. It is easy to verify that

$$\bar{a} \in Q(\mathfrak{D}) \iff \varphi_{Q,\mathfrak{D},\bar{a}} \text{ is unsatisfiable.}$$

However, $\varphi_{Q,\mathfrak{D},\bar{a}}$ does not directly fall in a well-behaved fragment, and, in particular, in the guarded fragment of first-order logic, mainly due to the unguarded sentence $q(\bar{a})$. Notice that, strictly speaking, also $\bigwedge_{\tau\in O}\tau$ is unguarded despite the fact that O consists of guarded TGDs. Nevertheless, a guarded TGD can be easily converted in polynomial time into an equisatisfiable guarded sentence (see, e.g., [5]) and thus we assume, w.l.o.g., that $\bigwedge_{\tau\in O}\tau$ falls in the guarded fragment. The goal now is to

transform $\varphi_{Q,\mathfrak{D},\bar{a}}$ into an equisatisfiable guarded sentence $\psi_{Q,\mathfrak{D},\bar{a}}$. This is done by exploiting the technique of *treeification* [5]. Let us first recall this technique, and then explain how we use it in order to construct the desired sentence $\psi_{Q,\mathfrak{D},\bar{a}}$.

Treeification

Every BCQ q over a schema $\mathbf{S}$ can be naturally associated with an $\mathbf{S}$-database, known as its *canonical database*, which consists of the atoms of q after converting each variable $x \in \mathsf{var}(q)$ into a constant a_x. We say that q is *acyclic* if its canonical database is acyclic, i.e., it admits a guarded tree decomposition.

Consider now a BCQ q over a schema $\mathbf{S}$, and a schema $\mathbf{T} \supseteq \mathbf{S}$. The $\mathbf{T}$-*treeification* of q is defined as the set $\Lambda_q^{\mathbf{T}}$ of all acyclic CQs q' over $\mathbf{T}$ of size at most three times the number of atoms occurring in q, such that q' is *contained* in q, i.e., $\mathfrak{D} \models q'$ implies $\mathfrak{D} \models q$ for every $\mathbf{T}$-database $\mathfrak{D}$. By abuse of notation, we may write $\Lambda_q^{\mathbf{T}}$ for the union (or disjunction) of its BCQs, i.e., the sentence $\bigvee_{q' \in \Lambda_q^{\mathbf{T}}} q'$. The main property of treeification is that it preserves satisfiability over acyclic models.[3] Since every satisfiable guarded sentence admits an acyclic model, we get the next useful result from [5]; we write $\varphi \models \bot$ to denote the fact that φ is unsatisfiable:

Lemma 1. *Consider a sentence $\varphi \in \mathsf{GF}[\mathbf{T}]$, and a BCQ q over $\mathbf{T}$. It holds that*

$$\varphi \wedge \neg q \models \bot \iff \varphi \wedge \neg \Lambda_q^{\mathbf{T}} \models \bot.$$

Before we proceed further, let us clarify that an acyclic BCQ does not directly fall in the guarded fragment. However, it is known that every acyclic BCQ can be equivalently rewritten as a guarded sentence [26]. Therefore, in what follows, we assume, w.l.o.g., that the sentence obtained after the treeification of a BCQ is guarded.

The Final Construction

Recall that we consider an instance of $\mathsf{Eval}(\mathsf{G},\mathsf{CQ})$ consisting of the OMQ $Q = (\mathbf{S}, O, q(\bar{x})) \in (\mathsf{G},\mathsf{CQ})$, the $\mathbf{S}$-database $\mathfrak{D}$, and the tuple of constants $\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|}$. At this point, one maybe tempted to think that to convert the sentence $\varphi_{Q,\mathfrak{D},\bar{a}}$ into an equisatisfiable guarded sentence $\psi_{Q,\mathfrak{D},\bar{a}}$, we simply need to treeify the BCQ $q(\bar{a})$. However, before doing this, we first need to properly eliminate the constants that occur in $\bigwedge_{\alpha \in \mathfrak{D}} \alpha$ and $q(\bar{a})$ in order to guarantee that after applying treeification the result will be an equisatisfiable guarded sentence. To this end, we first convert in polynomial time $\varphi_{Q,\mathfrak{D},\bar{a}}$ into a convenient equisatisfiable sentence $\varphi'_{Q,\mathfrak{D},\bar{a}}$, and then apply the treefication technique.

Assume that $\mathsf{adom}(\mathfrak{D}) = \{b_1,\ldots,b_k\}$. Let $\mathfrak{D}^+ := \mathfrak{D} \cup \{C_b(b)\}_{b \in \{b_1,\ldots,b_k\}}$, where $C_{b_1},\ldots,C_{b_k}$ are fresh unary relation symbols not in $\mathbf{S} \cup \mathsf{sig}(O)$. Let also $\mathfrak{D}^+_{var}$ be the set of atoms obtained from $\mathfrak{D}^+$ by replacing each occurrence of a constant

[3] Here, we see models as sets of atoms, i.e., as instances.

$b \in \mathsf{adom}(\mathfrak{D})$ with the variable x_b. Finally, let q^+ be the BCQ obtained from $q(\bar{a})$ by replacing each occurrence of a constant a with a fresh existentially quantified variable y_a, and adding the atom $C_a(y_a)$. We now define $\varphi'_{Q,\mathfrak{D},\bar{a}}$ as the sentence

$$\underbrace{\exists x_{b_1}, \cdots \exists x_{b_k} \left(C(x_{b_1}, \ldots, x_{b_k}) \wedge \bigwedge_{1 \leq i < j \leq k} x_{b_i} \neq x_{b_j} \wedge \bigwedge_{\alpha \in \mathfrak{D}^+_{var}} \alpha \right) \wedge \bigwedge_{\tau \in O} \tau}_{\Xi} \wedge \neg q^+,$$

where C is a fresh k-ary relation symbol. It should be clear that $\varphi_{Q,\mathfrak{D},\bar{a}}$ and $\varphi'_{Q,\mathfrak{D},\bar{a}}$ are equisatisfiable sentences. It is also clear that Ξ is a guarded sentence. Thus, by Lemma 1, we immediately get that

Proposition 1. *$\varphi_{Q,\mathfrak{D},\bar{a}}$ and $\Xi \wedge \Lambda^{\mathbf{T}}_{q^+}$, where $\mathbf{T} = \mathbf{S} \cup \{C\} \cup \{C_{b_i}\}_{1 \leq i \leq k} \cup \mathsf{sig}(O)$, are equisatisfiable sentences.*

The above result implies that $\bar{a} \in Q(\mathfrak{D})$ iff the guarded sentence $\Xi \wedge \Lambda^{\mathbf{T}}_{q^+}$ is unsatisfiable. Thus, we get a reduction from $\mathsf{Eval}(\mathsf{G}, \mathsf{CQ})$ to the unsatisfiability problem for guarded sentences. However, it should not be overlooked that this reduction takes exponential time due to treefication. In fact, we know from [5] that

$$|\Lambda^{\mathbf{T}}_{q^+}| \;\leq\; |\mathbf{T}|^{O(|q^+|)} \cdot (|q^+| \cdot \mathsf{wd}(\mathbf{T}))^{O(|q^+| \cdot \mathsf{wd}(\mathbf{T}))},$$

where $|q^+|$ is the size of q^+, while $\Lambda^{\mathbf{T}}_{q^+}$ can be constructed in time

$$|q^+| \cdot |\mathbf{T}|^{O(|q^+|)} \cdot (|q^+| \cdot \mathsf{wd}(\mathbf{T}))^{O(|q^+| \cdot \mathsf{wd}(\mathbf{T}))}.$$

Nevertheless, since the reduction provided by Proposition 1 increases the arity of the schema only polynomially, while the algorithm for checking whether a guarded sentence is unsatisfiable given in [27] is double exponential only on the arity of the underlying schema, we conclude that $\mathsf{Eval}(\mathsf{G}, \mathsf{CQ})$ is in 2ExpTime, as needed.

Interestingly, query answering against the entire guarded fragment can be shown to be decidable in 2ExpTime with the same construction as given above (cf. [5]).

Remark. The complexity stated in Theorem 1 refers to the *combined complexity* of query evaluation, i.e., when all the components are part of the input. However, in practice, it is realistic to assume that the OMQ is fixed, and only the database and the tuple of constants are part of the input. In this case we refer to the *data complexity* of $\mathsf{Eval}(\mathsf{G}, \mathsf{CQ})$, which is known to be PTime-complete [17]. Another relevant setting is when the arity of the underlying schema is bounded by an integer. In this case, $\mathsf{Eval}(\mathsf{G}, \mathsf{CQ})$ is ExpTime-complete [16]. The machinery described above, which exploits the guarded fragment of first-order logic, is not well-suited for obtaining optimal results in the above settings, and more refined techniques are needed. We refer the interested reader to the relevant literature for details.

1.4 Query Containment

We now focus on another important algorithmic task for a query language, namely query containment. Consider two OMQs $Q_1 = (\mathbf{S}, O_1, q_1(\bar{x}))$ and $Q_2 = (\mathbf{S}, O_2, q_2(\bar{x}))$. We say that Q_1 is *contained* in Q_2, written $Q_1 \subseteq Q_2$, if $Q_1(\mathfrak{D}) \subseteq Q_2(\mathfrak{D})$ for every $\mathbf{S}$-database $\mathfrak{D}$. The main problem that we study in this section is defined as follows:

PROBLEM :	$\mathsf{Cont}(\mathsf{O},\mathsf{Q})$
INPUT :	Two OMQs $Q_1, Q_2 \in (\mathsf{O},\mathsf{Q})$ with the same data schema.
QUESTION :	Is it the case that $Q_1 \subseteq Q_2$?

It is well-known that $\mathsf{Cont}(\mathsf{TGD},\mathsf{CQ})$ is undecidable. This is immediately inherited from the fact that query containment for Datalog queries is undecidable [35], since a Datalog query can be seen as an OMQ that falls in the language (F,CQ), where F denotes the class of *full* TGDs, i.e., TGDs without existentially quantified variables in the head. However, for (G,CQ) the problem becomes decidable. In fact:

Theorem 2. $\mathsf{Cont}(\mathsf{G},\mathsf{CQ})$ *is* 2ExpTime*-complete. The lower bound holds even if* $\mathbf{S} \cup \mathsf{sig}(O_1) \cup \mathsf{sig}(O_2)$ *consists of unary and binary relation symbols only.*

The lower bound for $\mathsf{Cont}(\mathsf{G},\mathsf{CQ})$ is immediately inherited from [12], where it is shown that the containment problem for OMQs where the ontology is formulated using the description logic *ELI* is 2ExpTime-hard. Since a set of *ELI* axioms can be equivalently rewritten, for query answering purposes, as a set of guarded TGDs, the 2ExpTime lower bound follows. It remains to establish the upper bound. In the rest of the section, we give some details on how this upper bound can be shown.

1.4.1 Atomic Queries

We first focus our attention on the simpler OMQ language $(\mathsf{G},\mathsf{AQ}_0)$, where AQ_0 denotes the class of all CQs that consist of a single atom $R()$, where R is 0-ary relation symbol, and show that query containment is in 2ExpTime. For brevity, we will simply write R instead of $R()$. Having this result in place, we are then going to explain how it can be extended to the language (G,CQ) by exploiting the technique of treefication discussed in the previous section. We proceed to show that:

Theorem 3. $\mathsf{Cont}(\mathsf{G},\mathsf{AQ}_0)$ *is in* 2ExpTime.

To establish the above result, we first show the so-called acyclic witness property, which states that non-containment for $(\mathsf{G},\mathsf{AQ}_0)$ is witnessed via an acyclic database, i.e., a database that admits a guarded tree decomposition, which in turn allows us to devise a decision procedure for $\mathsf{Cont}(\mathsf{G},\mathsf{AQ}_0)$ based on alternating tree automata that runs in 2ExpTime. Summing up, the proof for the 2ExpTime membership of $\mathsf{Cont}(\mathsf{G},\mathsf{AQ}_0)$ proceeds in three main steps:

1. Establish the acyclic witness property.
2. Encode the acyclic witnesses as trees that can be accepted by an alternating tree automaton.
3. Construct an automaton that decides $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$; in fact, we reduce our problem to emptiness for two-way alternating parity automata on finite trees.

Each one of the above three steps is discussed in more details below.

Acyclic Witness Property

We proceed to show that non-containment for $(\mathsf{G}, \mathsf{AQ}_0)$ is witnessed via an acyclic database. We write $Q_1 \not\subseteq Q_2$ to denote the fact that the OMQ Q_1 is not contained in the OMQ Q_2, or, equivalently, there exists an $\mathbf{S}$-database $\mathfrak{D}$, where $\mathbf{S}$ is the data schema of Q_1 and Q_2, such that $Q_1(\mathfrak{D}) \not\subseteq Q_2(\mathfrak{D})$.

Proposition 2. *Suppose that Q_1 and Q_2 are OMQs from $(\mathsf{G}, \mathsf{AQ}_0)$ with data schema $\mathbf{S}$. The following are equivalent:*

1. *$Q_1 \not\subseteq Q_2$.*
2. *There exists an acyclic $\mathbf{S}$-database $\mathfrak{D}$ such that $Q_1(\mathfrak{D}) \not\subseteq Q_2(\mathfrak{D})$.*

Proof. The fact that $(2) \Rightarrow (1)$ is trivial. For the other direction, we need an auxiliary result, which is shown using the notion of guarded unraveling (see, e.g, [2]) and the compactness theorem. Given two databases $\mathfrak{D}_1$ and $\mathfrak{D}_2$, a function $h\colon \mathsf{adom}(\mathfrak{D}_1) \to \mathsf{adom}(\mathfrak{D}_2)$ is a homomorphism from $\mathfrak{D}_1$ to $\mathfrak{D}_2$ if, for each $R(\bar{a}) \in \mathfrak{D}_1$, $R(h(\bar{a})) \in \mathfrak{D}_2$. The existence of such a homomorphism is denoted by $\mathfrak{D}_1 \to \mathfrak{D}_2$.

Lemma 2. *Let $\mathfrak{D}'$ be an $\mathbf{S}$-database, and $Q \in (\mathsf{G}, \mathsf{AQ}_0)$ with data schema $\mathbf{S}$. If $\mathfrak{D}' \models Q$ then there is an acyclic $\mathbf{S}$-database $\mathfrak{D}''$ such that $\mathfrak{D}'' \models Q$ and $\mathfrak{D}'' \to \mathfrak{D}'$.*

Having the above lemma in place, we can now show that $(1) \Rightarrow (2)$. By hypothesis, there exists an $\mathbf{S}$-database $\mathfrak{D}'$ such that $\mathfrak{D}' \models Q_1$ and $\mathfrak{D}' \not\models Q_2$. By Lemma 2, there exists an acyclic $\mathbf{S}$-database $\mathfrak{D}$ such that $\mathfrak{D} \models Q_1$ and $\mathfrak{D} \to \mathfrak{D}'$. It is known that OMQs from $(\mathsf{G}, \mathsf{AQ}_0)$ are closed under homomorphisms [14], which immediately implies that $\mathfrak{D} \not\models Q_2$. Thus, $Q_1(\mathfrak{D}) \not\subseteq Q_2(\mathfrak{D})$, as needed. □

Encoding Acyclic Databases

The next step is to encode acyclic databases as trees that can be accepted by an alternating tree automaton. The key observation here is that acyclic databases are "tree-like", i.e., are of bounded tree-width. The *tree-width* of a database $\mathfrak{D}$ is the minimum width among all the tree decompositions $\delta = (\mathscr{T}, (X_v)_{v \in T})$ for $\mathfrak{D}$, while the width of δ is $\max\{|X_v| \mid v \in T\} - 1$. It is generally known that a database $\mathfrak{D}$ whose tree-width 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., [10]. Since the tree-width of an acyclic $\mathbf{S}$-database is bounded by $\mathsf{wd}(\mathbf{S})-1$, such an encoding can be used for acyclic databases.

Let Γ be an alphabet and $(\mathbb{N}\setminus\{0\})^*$ be the set of finite sequences of positive integers, including the empty sequence. A *Γ-labeled tree* is a partial function $t\colon (\mathbb{N}\setminus\{0\})^* \to \Gamma$, whose domain $\mathsf{dom}(t)$ is closed under prefixes, i.e., $x\cdot i \in \mathsf{dom}(t)$ implies $x \in \mathsf{dom}(t)$, for all $x \in (\mathbb{N}\setminus\{0\})^*$ and $i \in \mathbb{N}\setminus\{0\}$. The elements of $\mathsf{dom}(t)$ identify the *nodes* of t. Given an acyclic $\mathbf{S}$-database $\mathfrak{D}$, and a guarded tree decomposition δ for $\mathfrak{D}$ of width $\mathsf{wd}(\mathbf{S})-1$, it can be shown that $\mathfrak{D}$ and δ can be encoded as a $\Gamma_{\mathbf{S}}$-labeled tree t, where $\Gamma_{\mathbf{S}}$ is an alphabet of double exponential size in $\mathsf{wd}(\mathbf{S})$ and exponential size in $|\mathbf{S}|$, such that each node of δ corresponds to exactly one node of t and vice versa.

Although every acyclic $\mathbf{S}$-database can be encoded as a $\Gamma_{\mathbf{S}}$-labeled tree, the other direction does not hold. In other words, it is not the case that every $\Gamma_{\mathbf{S}}$-labeled tree encodes an acyclic $\mathbf{S}$-database and its corresponding guarded tree decomposition. In view of this fact, we need the additional notion of consistency. A $\Gamma_{\mathbf{S}}$-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 $\Gamma_{\mathbf{S}}$-labeled tree t, we can show that t can be decoded into an acyclic $\mathbf{S}$-database $[\![t]\!]$. From the above discussion and Proposition 2, we obtain the following lemma:

Lemma 3. *Suppose that Q_1 and Q_2 are OMQs from $(\mathsf{G},\mathsf{AQ}_0)$ with data schema $\mathbf{S}$. The following are equivalent:*

1. *$Q_1 \not\subseteq Q_2$.*
2. *There exists a consistent $\Gamma_{\mathbf{S}}$-labeled tree t such that $Q_1([\![t]\!]) \not\subseteq Q_2([\![t]\!])$.*

Constructing Tree Automata

We now proceed with our automata-based procedure. We use two-way alternating parity automata (2ATA) that run on finite labeled trees of unbounded degree. 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 $\mathsf{Cont}(\mathsf{G},\mathsf{AQ}_0)$ to the emptiness problem for 2ATA. As usual, given a 2ATA $\mathscr{A}$, we denote by $L(\mathscr{A})$ the *language* of $\mathscr{A}$, i.e., the set of labeled trees it accepts. The emptiness problem is defined as follows: given a 2ATA $\mathscr{A}$, does $L(\mathscr{A}) = \emptyset$? Thus, given $Q_1, Q_2 \in (\mathsf{G},\mathsf{AQ}_0)$, we need to construct a 2ATA $\mathscr{A}$ such that $Q_1 \subseteq Q_2$ iff $L(\mathscr{A}) = \emptyset$. It is well-known that deciding whether $L(\mathscr{A})$ is empty is feasible in exponential time in the number of states, and in polynomial time in the size of the input alphabet [23]. Therefore, in order to obtain the desired 2ExpTime upper bound, we should construct $\mathscr{A}$ 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. The construction of an automaton for this task is fairly standard in the literature on automata for guarded logics (see, e.g., [10, 11]), and we omit the details.

Lemma 4. *Consider a schema* $\mathbf{S}$*. There exists a 2ATA* $\mathscr{C}_{\mathbf{S}}$ *that accepts a* $\Gamma_{\mathbf{S}}$*-labeled tree* t *iff* t *is consistent. The number of states of* $\mathscr{C}_{\mathbf{S}}$ *is exponential in* $\mathsf{wd}(\mathbf{S})$ *and linear in* $|\mathbf{S}|$*. Moreover,* $\mathscr{C}_{\mathbf{S}}$ *can be constructed in double exponential time in* $\mathsf{wd}(\mathbf{S})$ *and in exponential time in* $|\mathbf{S}|$*.*

Now, the crucial task is, given an OMQ $Q \in (\mathsf{G}, \mathsf{AQ}_0)$, to devise an automaton that accepts labeled trees which correspond to databases that make Q true.

Lemma 5. *Consider an OMQ* $Q = (\mathbf{S}, O, q) \in (\mathsf{G}, \mathsf{AQ}_0)$*. There is a 2ATA* $\mathscr{A}_Q$ *that accepts a consistent* $\Gamma_{\mathbf{S}}$*-labeled tree* t *iff* $[\![t]\!] \models Q$*. The number of states of* $\mathscr{A}_Q$ *is exponential in* $\mathsf{wd}(\mathbf{S})$ *and linear in* $|\mathbf{S} \cup \mathsf{sig}(O)|$*. Moreover,* $\mathscr{A}_Q$ *can be constructed in double exponential time in the size of* Q*.*

Let us give some insights on the construction of $\mathscr{A}_Q$. Assume that $Q = (\mathbf{S}, O, G)$, i.e., the atomic query consists of the 0-ary predicate G. Roughly speaking, given a consistent $\Gamma_{\mathbf{S}}$-labeled tree t as input, $\mathscr{A}_Q$ tries to find derivations that witness the fact that $[\![t]\!] \models Q$. But let us first formalize the notion of derivation. Let $\mathfrak{D}$ be an $\mathbf{S}$-database. A *derivation tree* for $\mathfrak{D}$ and Q is a finite labeled tree $\mathscr{T}$, with η being the node labeling function that assigns facts $R(\bar{a})$ to nodes, where $R \in \mathbf{S} \cup \mathsf{sig}(O)$ and $\bar{a} \subseteq \mathsf{adom}(\mathfrak{D})$, such that the following conditions are satisfied:

1. For the root node v of $\mathscr{T}$ we have that $\eta(v) = G$.
2. For each leaf node v of $\mathscr{T}$ we have that $\eta(v) \in \mathfrak{D}$.
3. For each non-leaf node v of $\mathscr{T}$, with $u_1, \ldots, u_k$ being its children, we have that $\{\eta(u_1), \ldots, \eta(u_k)\}$ is *guarded*, i.e., it has an atom that contains all the terms of $\mathsf{adom}(\{\eta(u_1), \ldots, \eta(u_k)\})$, and $(\{\eta(u_1), \ldots, \eta(u_k)\}, O) \models \eta(v)$.

Intuitively, $\mathscr{T}$ describes how the atom G can be entailed from $\mathfrak{D}$ and O. It is easy to show that $\mathfrak{D} \models Q$ iff there exists a derivation tree for $\mathfrak{D}$ and Q. Moreover, due to the guardedness condition in point (3) above, it is possible to show that whenever there is a derivation tree for $\mathfrak{D}$ and Q, there is one whose branching degree is bounded by a function that is exponential in $\mathsf{wd}(\mathbf{S} \cup \mathsf{sig}(O))$. The automaton $\mathscr{A}_Q$ exploits these facts in order to exhaustively search for derivation trees that witness the fact that $[\![t]\!] \models Q$ on an input tree t. To this end, $\mathscr{A}_Q$ maintains states for the possible labels that may occur in a derivation tree for $[\![t]\!]$ and Q. It turns out that this state set is exponential, as stated in Lemma 5. Starting with the atom G, the automaton guesses labels of children of G that may occur in a candidate derivation tree of $[\![t]\!]$ and Q. Thanks to alternation, it can then proceed in the same way for each child label until it succeeds to build a derivation tree for $[\![t]\!]$ and Q. A detailed analysis of the construction of $\mathscr{A}_Q$ reveals that $\mathscr{A}_Q$ can be constructed in double exponential time in the size of Q, where the second exponent depends only on the maximum arity among all relation symbols present in Q.

Having the above automata in place, we can show that $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$ can be reduced to the emptiness problem for 2ATA. But let us first recall some key results about 2ATA, which are essential for the final construction. It is well-known that languages accepted by 2ATAs are closed under intersection and complement. Given two 2ATAs $\mathscr{A}_1$ and $\mathscr{A}_2$, we write $\mathscr{A}_1 \cap \mathscr{A}_2$ for a 2ATA, which can be constructed in

polynomial time, that accepts the language $L(\mathscr{A}_1) \cap L(\mathscr{A}_2)$. Moreover, for a 2ATA $\mathscr{A}$, we write $\overline{\mathscr{A}}$ for the 2ATA, which is also constructible in polynomial time, that accepts the complement of $L(\mathscr{A})$. We can now show the following result:

Proposition 3. *Consider $Q_1, Q_2 \in (\mathsf{G}, \mathsf{AQ}_0)$. We can construct in double exponential time a 2ATA $\mathscr{A}$ with exponentially many states such that $Q_1 \subseteq Q_2$ iff $L(\mathscr{A}) = \emptyset$.*

Proof. Assume that both Q_1 and Q_2 have data schema $\mathbf{S}$. We define $\mathscr{A}$ as the automaton $\mathscr{C}_{\mathbf{S}} \cap \mathscr{A}_{Q_1} \cap \overline{\mathscr{A}_{Q_2}}$. By Lemmas 3, 4 and 5, it is easy to verify that indeed $\mathscr{A}$ is constructible in double exponential time, while it has exponentially many states, and that $Q_1 \subseteq Q_2$ iff $L(\mathscr{A}) = \emptyset$. The claim follows. □

Recall that for a 2ATA $\mathscr{A}$ deciding whether $L(\mathscr{A})$ is empty is feasible in exponential time in the number of states, and in polynomial time in the size of the input alphabet [23]. Consequently, Proposition 3 immediately implies that $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$ is in 2ExpTime, and Theorem 3 follows.

1.4.2 From Conjunctive Queries to Atomic Queries

Let us now explain how we get the desired 2ExpTime upper bound for $\mathsf{Cont}(\mathsf{G}, \mathsf{CQ})$ by exploiting the fact that $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$ is in 2ExpTime. We first observe that it suffices to focus on the OMQ language $(\mathsf{G}, \mathsf{BCQ})$, i.e., queries from $(\mathsf{G}, \mathsf{CQ})$ where the CQ is Boolean. This follows from the fact that there is a simple polynomial time reduction from $\mathsf{Cont}(\mathsf{G}, \mathsf{CQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{BCQ})$, which is a straightforward adaptation of the one give in [12] for OMQs based on the description logic *ELI*. Therefore, the goal is to reduce $\mathsf{Cont}(\mathsf{G}, \mathsf{BCQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$, and then apply Theorem 3. This reduction relies on the treefication technique discussed in Section 1.3, and is inspired by a translation of guarded negation fixed-point sentences to guarded fixed-point sentences given in [6].

Consider an OMQ $Q = (\mathbf{S}, O, q) \in (\mathsf{G}, \mathsf{BCQ})$, and let C be a relation symbol not in $\mathbf{S} \cup \mathsf{sig}(O)$ that has arity $\mathsf{wd}(q)$, where $\mathsf{wd}(q)$ denotes the *width* of q, i.e., the number of variables occurring in q. We define the set of TGDs

$$\eta_C^Q \coloneqq \left\{ q' \to G_q \mid q' \in \Lambda_q^{\mathbf{S} \cup \{C\} \cup \mathsf{sig}(O)} \right\},$$

where G_q is a new 0-ary relation symbol not in $\mathbf{S} \cup \{C\} \cup \mathsf{sig}(O)$. Notice that the TGDs in η_C^Q are, in general, not guarded. However, by construction, their bodies are acyclic Boolean CQs, and this allows us to rewrite each TGD τ into linearly many guarded TGDs, which we denote by γ_τ. We then define the set of guarded TGDs

$$\gamma_C^Q \coloneqq \bigcup_{\tau \in \eta_C^Q} \gamma_\tau.$$

We finally define the OMQ

$$g_C(Q) \coloneqq \left(\mathbf{S} \cup \{C\}, O \cup \gamma_C^Q, G_q\right) \in (\mathsf{G}, \mathsf{AQ}_0).$$

It can be shown that the translation $g_C(\cdot)$ preserves containment. More precisely:

Lemma 6. *Let $Q_i = (\mathbf{S}, O_i, q_i) \in (\mathsf{G}, \mathsf{BCQ})$, for $i \in \{1,2\}$, and consider a predicate $C \notin \mathbf{S} \cup \mathsf{sig}(O_1) \cup \mathsf{sig}(O_2)$ that has arity $\max_{i\in\{1,2\}}\{\mathsf{wd}(q_i)\}$. It holds that*

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

The above lemma provides the reduction from $\mathsf{Cont}(\mathsf{G}, \mathsf{BCQ})$ to $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$, which allows us to apply the algorithm for $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$ underlying Theorem 3. However, it should not be forgotten that this reduction takes exponential time due to treefication. Nevertheless, since the reduction provided by Lemma 6 increases the arity of the schema only polynomially, while the algorithm for $\mathsf{Cont}(\mathsf{G}, \mathsf{AQ}_0)$ provided by Theorem 3 is double exponential only on the arity of the underlying schema, we conclude that $\mathsf{Cont}(\mathsf{G}, \mathsf{BCQ})$ is in 2ExpTime, as needed.

1.5 First-Order Rewritability

We now focus on another algorithmic task that is relevant for OMQs, that is, deciding whether an OMQ can be equivalently rewritten as a first-order query. A *first-order (FO) query* over a schema $\mathbf{S}$ is a (function-free) FO-formula $\varphi(\bar{x})$, with $\bar{x}$ being its free variables, that uses only relations from $\mathbf{S}$. The *evaluation* of φ over an $\mathbf{S}$-database $\mathfrak{D}$, denoted $\varphi(\mathfrak{D})$, is the set of tuples $\{\bar{a} \in \mathsf{adom}(\mathfrak{D})^{|\bar{x}|} \mid \mathfrak{D} \models \varphi(\bar{a})\}$, where $\models$ denotes the standard notion of satisfaction for first-order logic. An OMQ $Q = (\mathbf{S}, O, q(\bar{x}))$ is *FO-rewritable* if there is a (finite) FO-query $\varphi_Q(\bar{x})$ over $\mathbf{S}$ that is equivalent to Q, i.e., $Q(\mathfrak{D}) = \varphi_Q(\mathfrak{D})$ for every $\mathbf{S}$-database $\mathfrak{D}$. We call $\varphi_Q(\bar{x})$ an *FO-rewriting of Q*. The main problem that we study in this section is the following:

PROBLEM :	$\mathsf{FORew}(\mathsf{O}, \mathsf{Q})$
INPUT :	An OMQ $Q \in (\mathsf{O}, \mathsf{Q})$.
QUESTION :	Is it the case that Q is FO-rewritable?

It is well-known that $\mathsf{FORew}(\mathsf{TGD}, \mathsf{CQ})$ is undecidable. This follows from the fact that deciding whether a Datalog query, and thus an OMQ from $(\mathsf{F}, \mathsf{CQ})$[4], is FO-rewritable is undecidable. Actually, we know that a Datalog query is FO-rewritable iff it is bounded [1], while the problem of deciding whether a Datalog query is bounded is undecidable [24]. What about OMQs that fall in the language $(\mathsf{G}, \mathsf{CQ})$?

The next two examples show that $\mathsf{FORew}(\mathsf{G}, \mathsf{CQ})$ is a non-trivial problem in the sense that there are queries from $(\mathsf{G}, \mathsf{CQ})$ that are *not* FO-rewritable, but at the same time there are FO-rewritable queries from $(\mathsf{G}, \mathsf{CQ})$.

[4] Recall that F denotes the class of full TGDs, i.e., TGDs without existentially quantified variables.

Example 2. Let $Q = (\mathbf{S}, O, q) \in (\mathsf{G}, \mathsf{CQ})$, where $\mathbf{S} = \{T/3, A/1, B/1\}$,

$$O \coloneqq \{T(x,y,z), A(z) \rightarrow R(x,z), \quad T(x,y,z), R(x,z) \rightarrow R(x,y)\}$$

and

$$q \coloneqq \exists x \exists y \exists z \, (T(x,y,z) \wedge R(x,z) \wedge B(y)).$$

Intuitively, an FO-rewriting of Q should check for the existence of a set of atoms $\{T(c, a_i, a_{i-1})\}_{1 \leq i \leq k}$ for some $k \geq 0$. However, since there is no upper bound for k, this cannot be done via a finite first-order query, and thus Q is not FO-rewritable. A proof for this fact is given below. □

By slightly adapting the above example, we get a query that is FO-rewritable.

Example 3. Let $Q' = (\mathbf{S}, O, q') \in (\mathsf{G}, \mathsf{CQ})$ be the query obtain from the OMQ Q given in Example 2 by replacing q with

$$q' \coloneqq \exists x \exists y \exists z \, (T(x,y,z) \wedge R(x,z) \wedge B(y) \wedge A(z)),$$

i.e., by simply adding to the CQ q the atom $A(z)$. The query Q' is indeed FO-rewritable since the FO-query, which is actually a CQ,

$$\exists x \exists y \exists z \, (T(x,y,z) \wedge B(y) \wedge A(z))$$

is an FO-rewriting of Q'. □

Interestingly, we can decide whether a query from $(\mathsf{G}, \mathsf{CQ})$ is FO-rewritable:

Theorem 4. $\mathsf{FORew}(\mathsf{G}, \mathsf{CQ})$ *is* 2ExpTime*-complete. The lower bound holds even if* $\mathbf{S} \cup \mathsf{sig}(O)$ *consists of unary and binary relation symbols only.*

The lower bound for $\mathsf{FORew}(\mathsf{G}, \mathsf{CQ})$ is immediately inherited from [12], where it is shown that deciding FO-rewritability for OMQs where the ontology is formulated using the description logic *ELI* is 2ExpTime-hard, while, as discussed in the previous section, a set of *ELI* axioms can be equivalently rewritten, for query answering purposes, as a set of guarded TGDs. It remains to establish the upper bound. In the rest of the section, we discuss how this can be obtained.

1.5.1 Atomic Queries

As in the case of query containment, we first concentrate our attention on the simpler language $(\mathsf{G}, \mathsf{AQ}_0)$, and show that deciding FO-rewritability is in 2ExpTime. Then, we are going to explain how we can get the desired upper bound for queries from $(\mathsf{G}, \mathsf{CQ})$ by exploiting the decision procedure for $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ and the treeification technique. We proceed to show that:

Theorem 5. $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ *is in* 2ExpTime.

Towards a decision procedure for $\mathsf{FORew}(\mathsf{G},\mathsf{AQ}_0)$, we first semantically characterize the FO-rewritable OMQs from $(\mathsf{G},\mathsf{AQ}_0)$, and then explain how this semantic characterization can be exploited in order to devise a decision procedure based on automata techniques.

Semantic Characterization

We give a characterization of FO-rewritability of OMQs from $(\mathsf{G},\mathsf{AQ}_0)$ in terms of the existence of certain acyclic databases. This characterization is related to, but different from characterizations used for OMQs based on DLs such as *ELI* and *EL* [12, 13]. The DL characterizations essentially state that a unary OMQ Q (i.e., one whose query has a single answer variable) is FO-rewritable iff there is a bound k such that, whenever the root of a tree-shaped database $\mathfrak{D}$ is returned as an answer to Q, then this is already true for the restriction of $\mathfrak{D}$ up to depth k. The proof of the (contrapositive of the) "only if" direction uses a locality argument: if there is no such bound k, then this is witnessed by an infinite sequence of deeper and deeper tree databases that establish non-locality of Q. For guarded TGDs, we would have to replace tree-shaped databases with acyclic databases. However, increasing depth of guarded tree decompositions does not correspond to increasing distance in the Gaifman graph and thus does not establish non-locality. We therefore depart from imposing a bound on the depth, and instead we impose a bound on the number of facts (see Proposition 4). It is also interesting to note that, while it is implicit in [12] that an OMQ based on *ELI* and CQs is FO-rewritable iff it is Gaifman local, there is an OMQ from (G,CQ) that is Gaifman local, but not FO-rewritable. Such an OMQ is the one obtained from the query Q given in Example 2, by removing the existential quantification on the variable x in the CQ q, i.e., converting q into a unary CQ.

Proposition 4. *Consider an OMQ $Q \in (\mathsf{G},\mathsf{AQ}_0)$ with data schema $\mathbf{S}$. The following are equivalent:*

1. *Q is FO-rewritable.*
2. *There exists a $k \geq 0$ such that, for every acyclic $\mathbf{S}$-database $\mathfrak{D}$, if $\mathfrak{D} \models Q$, then there is a $\mathfrak{D}' \subseteq \mathfrak{D}$ with at most k facts such that $\mathfrak{D}' \models Q$.*

Proof. For (1) $\Rightarrow$ (2) we exploit the fact that, if $Q \in (\mathsf{G},\mathsf{AQ}_0)$ is FO-rewritable, then it can be expressed as a union of CQs q_Q, i.e., a disjunction of CQs. This follows from the fact that queries from $(\mathsf{G},\mathsf{AQ}_0)$ are preserved under homomorphisms [14], and Rossman's powerful theorem stating that an FO-query is preserved under homomorphisms over finite instances iff it is equivalent to a union of CQs [34]. It is then easy to show that (2) holds with k being the size of the largest disjunct of q_Q.

For (2) $\Rightarrow$ (1) we explicitly construct an FO-rewriting of Q. Let

$$\Lambda_{Q,k} \coloneqq \{\mathfrak{D}' \subseteq \mathfrak{D} \mid \mathfrak{D} \text{ is an acyclic } \mathbf{S}\text{-database}, |\mathfrak{D}'| \leq k, \mathfrak{D}' \models Q\}.$$

We consider the union of Boolean CQs $\varphi_Q \coloneqq \bigvee_{\mathfrak{D} \in \Lambda_{Q,k}} q_{\mathfrak{D}}$, where $q_{\mathfrak{D}}$ is the Boolean CQ obtained from $\mathfrak{D}$ by replacing each constant $c \in \mathsf{adom}(\mathfrak{D})$ with an existentially

quantified variable x_c. It is easy to see that φ_Q is finite (modulo variable renaming). By exploiting Lemma 2, it is not difficult to show that φ_Q is indeed an FO-rewriting of Q, i.e., $Q(\mathfrak{D}) = \varphi_Q(\mathfrak{D})$, for every $\mathbf{S}$-database $\mathfrak{D}$, and the claim follows. □.

The next example illustrates Proposition 4.

Example 4. Let $Q = (\mathbf{S}, O, G) \in (\mathsf{G}, \mathsf{AQ}_0)$, where $\mathbf{S} = \{T/3, A/1, B/1\}$, and O consists of the TGDs given in Example 2 plus the guarded TGD

$$T(x,y,z), R(x,z), B(y) \rightarrow G.$$

It is easy to verify that, for an arbitrary $k \geq 0$, the acyclic $\mathbf{S}$-database

$$\mathfrak{D}_k = \{A(a_0), T(c,a_1,a_0), \ldots, T(c,a_{k-1},a_{k-2}), B(a_{k-1})\}$$

is such that $\mathfrak{D}_k \models Q$, but for every $\mathfrak{D}' \subset \mathfrak{D}_k$ with at most k facts, $\mathfrak{D}' \not\models Q$. Thus, by Proposition 4, Q is not FO-rewritable. □

At this point, one might expect that Proposition 4 allows us to devise a decision procedure for $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ based on 2ATA. Indeed, although the semantic characterization established in Proposition 4 does not immediately provide a way to devise such a procedure, it can be refined in order to arrive at a criterion that permits an implementation via 2ATA [7]. However, the automata-based decision procedure that emerges from this refined characterization runs in triple exponential time. Roughly, the refined semantic characterization relies on a minimality criterion that is defined on the class of all acyclic databases, and ensures that there are only finitely many "minimal" acyclic databases that satisfy the input OMQ Q. One can then devise a 2ATA $\mathscr{A}_Q$ that checks for this criterion, and thus, Q is FO-rewritable iff the language accepted by $\mathscr{A}_Q$ is finite. The latter is feasible in exponential time in the number of states. Unfortunately, the construction of $\mathscr{A}_Q$ relies on the costly operation of projection, and for this reason $\mathscr{A}_Q$ has double exponentially many states. Therefore, this leads to a decision procedure that runs in triple exponential time, which is not optimal. We refer the interested reader to [7] for more details.

Cost Automata Approach

Since it is not apparent how the use of traditional automata techniques may lead to the desired 2ExpTime upper bound for $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$, we instead are going to exploit the more sophisticated model of *cost automata*. The goal is to provide a refined version of the semantic characterization given in Proposition 4 that relies on a minimality criterion. Since in the cost automata model the operation of minimization is a native citizen (details are given below), we can deal with such a minimality criterion in a more efficient way than standard 2ATA. In what follows, we briefly discuss cost automata, we then revisit the semantic characterization in Proposition 4, and finally explain how the refined characterization of FO-rewritability leads to an optimal decision procedure for $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ based on cost automata.

Cost Automata Models. Cost automata extend traditional automata (on words, trees, etc.) by providing counters that can be manipulated at each transition. Instead of assigning a Boolean value to each input structure (indicating whether the input is accepted or not), these automata assign a value from $\mathbb{N}_\infty := \mathbb{N} \cup \{\infty\}$ to each input. We shall only give a high-level idea of cost automata in the following, and refer the reader to the literature for more details [11, 20, 22].

Here, we focus on cost automata that work on finite trees of unbounded degree, and allow for two-way movements; in fact, the automata that we need are those that extend 2ATA over labeled trees with a *single* counter. The operation of such an automaton $\mathscr{A}$ on each input t will be viewed as a two-player *cost game* $\mathscr{G}(\mathscr{A},t)$ between players Eve and Adam. Recall that the acceptance of an input tree for a conventional 2ATA can be formalized via a two-player game as well, and, in fact, the standard parity game for 2ATA can be seen as a special case of a cost game [11]. However, instead of the parity acceptance condition for 2ATA, plays in the cost game between Eve and Adam will be assigned costs, and the cost automaton specifies via an *objective* whether Eve's goal is to minimize or maximize that cost. In case of a minimizing (resp., maximizing) objective, a strategy ξ of Eve in the cost game $\mathscr{G}(\mathscr{A},t)$ is *n-winning* if any play of Adam consistent with ξ has cost at most n (resp., at least n). $\mathscr{A}$ defines the following function on the domain of all input trees:

$$[\![\mathscr{A}]\!] : t \longmapsto \mathrm{op}\{n \mid \text{Eve has an } n\text{-winning strategy in } \mathscr{G}(\mathscr{A},t)\},$$

where op = inf (resp., op = sup) in case Eve's objective is to minimize (resp., maximize). Therefore, $[\![\mathscr{A}]\!]$ defines a function from the domain of input trees to $\mathbb{N}_\infty$. We call functions of that type *cost functions*. A key property of such functions is *boundedness*. We say that $[\![\mathscr{A}]\!]$ is *bounded* if there exists an $n \in \mathbb{N}$ such that $[\![\mathscr{A}]\!](t) \leq n$ for every input tree t.

We employ cost automata on trees with a single counter, where Eve's objective is to minimize the cost, while satisfying the parity condition. Such an automaton is known in the literature as $\mathsf{dist} \wedge \mathsf{parity}$-*automaton* [11]. To navigate in the tree, it may use the directions $\{0, \updownarrow\}$, where 0 indicates that the automaton should stay in the current node, and $\updownarrow$ means that the automaton may move to an arbitrary neighboring node, including the parent. For this type of automaton, we can decide whether its cost function is bounded [11, 21]. As usual, $|\mathscr{A}|$ denotes the size $\mathscr{A}$. Then:[5]

Theorem 6. *There is a polynomial f such that, for every* $\mathsf{dist} \wedge \mathsf{parity}$-*automaton $\mathscr{A}$ using priorities $\{0,1\}$ for the parity acceptance condition, boundedness of $[\![\mathscr{A}]\!]$ is decidable in time $|\mathscr{A}|^{f(m)}$, where m is the number of states of $\mathscr{A}$.*

The goal is to reduce $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ to the problem of deciding whether a $\mathsf{dist} \wedge \mathsf{parity}$-automaton is bounded. To this end, we first need to revise the semantic characterization of FO-rewritability provided in Proposition 4.

A Revised Semantic Charaterization. Consider an $\mathbf{S}$-database $\mathfrak{D}$, and a query $Q = (\mathbf{S}, O, q) \in (\mathsf{G}, \mathsf{AQ}_0)$. Recall that a derivation tree for $\mathfrak{D}$ and Q is a finite labeled tree

[5] This result from [11] initially relied on an unpublished result. Such a result has been now published in [21].

that describes how the atomic query of Q can be entailed from $\mathfrak{D}$ and O; the formal definition can be found in the previous section. The *height* of a derivation tree $\mathscr{T}$ for $\mathfrak{D}$ and Q is the maximum number of nodes of a branch in $\mathscr{T}$, i.e., the maximum number of nodes that lie on a path that leads from the root node to a leaf node without repeating nodes. Assuming that $\mathfrak{D} \models Q$, we define the cost of $\mathfrak{D}$ w.r.t. Q as

$$\mathsf{cost}(\mathfrak{D}, Q) := \min\{n \mid \mathscr{T} \text{ is a derivation tree for } \mathfrak{D} \text{ and } Q \text{ of height } n\}.$$

The *cost* of Q is defined as

$$\mathsf{cost}(Q) := \sup\{\mathsf{cost}(\mathfrak{D}, Q) \mid Q(\mathfrak{D}) \neq \emptyset, \text{ where } \mathfrak{D} \text{ is an acyclic } \mathbf{S}\text{-database}\}.$$

Therefore, the cost of Q is the least upper bound of the height over all derivation trees for all acyclic $\mathbf{S}$-databases $\mathfrak{D}$ such that $\mathfrak{D} \models Q$. If there is no such a database, then the cost of Q is zero since $\sup \emptyset = 0$. Actually, $\mathsf{cost}(Q) = 0$ indicates that there is no (acyclic) database $\mathfrak{D}$ that satisfies Q, which means that Q is unsatisfiable, and thus it is trivially FO-rewritable.

Having the notion of the cost of an OMQ from $(\mathsf{G}, \mathsf{AQ}_0)$ in place, it should be clear how the semantic characterization in Proposition 4 can be refined:

Proposition 5. *Consider an OMQ $Q \in (\mathsf{G}, \mathsf{AQ}_0)$ with data schema $\mathbf{S}$. The following are equivalent:*

1. *Condition 2 from Proposition 4 is satisfied.*
2. *$\mathsf{cost}(Q)$ is finite.*

Constructing Cost Automata. We briefly describe how we can use cost automata in order to devise an algorithm for $\mathsf{FORew}(\mathsf{G}, \mathsf{AQ}_0)$ that runs in double exponential time. Consider an OMQ $Q = (\mathbf{S}, O, G) \in (\mathsf{G}, \mathsf{AQ}_0)$. The goal is to devise a $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathscr{B}_Q$ such that the cost function $[\![\mathscr{B}_Q]\!]$ is bounded iff $\mathsf{cost}(Q)$ is finite. Therefore, by Proposition 5, to check whether Q is FO-rewritable we simply need to check if $[\![\mathscr{B}_Q]\!]$ is bounded, which, by Theorem 6, can be done in exponential time in the size of $\mathscr{B}_Q$. The input trees to our automata will be over the same alphabet $\Gamma_{\mathbf{S}}$ that is used to encode acyclic $\mathbf{S}$-databases in Section 1.4. Recall that, for a $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathscr{A}$, the cost function $[\![\mathscr{A}]\!]$ is bounded over a certain class $\mathscr{C}$ of trees iff there is an $n \in \mathbb{N}$ such that $[\![\mathscr{A}]\!](t) \leq n$ for every input tree $t \in \mathscr{C}$. Then:

Lemma 7. *There is a $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathscr{H}_Q$ such that $[\![\mathscr{H}_Q]\!]$ is bounded over consistent $\Gamma_{\mathbf{S}}$-labeled trees iff $\mathsf{cost}(Q)$ is finite. The number of states of $\mathscr{H}_Q$ is exponential in $\mathsf{wd}(\mathbf{S})$, and polynomial in $|\mathbf{S} \cup \mathsf{sig}(O)|$. Moreover, $\mathscr{H}_Q$ can be constructed in double exponential time in the size of Q.*

The automaton $\mathscr{H}_Q$ is built in such a way that, on an input tree t, Eve has an n-winning strategy in $\mathscr{G}(\mathscr{H}_Q, t)$ iff there is a derivation tree for $[\![t]\!]$ and Q of height at most n. Thus, Eve tries to construct derivation trees of minimal height. The counter is used to count the height of the derivation tree.

Having this automaton in place, we can now complete the proof of Theorem 5. The desired $\mathsf{dist} \wedge \mathsf{parity}$-automaton $\mathscr{B}_Q$ is defined as $\mathscr{C}'_{\mathbf{S}} \cap \mathscr{H}_Q$, where $\mathscr{C}'_{\mathbf{S}}$ is similar

to the 2ATA $\mathscr{C}_{\mathbf{S}}$ (in Lemma 4) that checks for consistency of $\Gamma_{\mathbf{S}}$-labeled trees. Notice that $\mathscr{C}'_{\mathbf{S}}$ is essentially a $\mathsf{dist} \wedge \mathsf{parity}$-automaton that assigns zero (resp., ∞) to input trees that are consistent (resp., inconsistent), and thus, $\mathscr{C}'_{\mathbf{S}} \cap \mathscr{H}_Q$ is well-defined. Since the intersection of $\mathsf{dist} \wedge \mathsf{parity}$-automata is feasible in polynomial time [11], Lemma 4 and Lemma 7 imply that $\mathscr{B}_Q$ has exponentially many states, and it can be constructed in double exponential time. Lemma 7 implies also that $[\![\mathscr{B}_Q]\!]$ is bounded iff $\mathsf{cost}(Q)$ is finite. It remains to show that the boundedness of $[\![\mathscr{B}_Q]\!]$ can be checked in double exponential time. By Theorem 6, there is a polynomial f such that the latter task can be carried out in time $|\mathscr{B}_Q|^{f(m)}$, where m is the number of states of $\mathscr{B}_Q$, and the claim follows.

1.5.2 From Conjunctive Queries to Atomic Queries

In this final section, we explain how we get the desired 2ExpTime upper bound for $\mathsf{FORew}(\mathsf{G},\mathsf{CQ})$ by exploiting the fact that $\mathsf{FORew}(\mathsf{G},\mathsf{AQ}_0)$ is in 2ExpTime. As for containment, it suffices to focus on the OMQ language $(\mathsf{G},\mathsf{BCQ})$; implicit in [12]. Therefore, the goal is to reduce $\mathsf{FORew}(\mathsf{G},\mathsf{BCQ})$ to $\mathsf{FORew}(\mathsf{G},\mathsf{AQ}_0)$, and then apply Theorem 5. To this end, we follow the same approach as for containment. Recall that for an OMQ $Q = (\mathbf{S},O,q) \in (\mathsf{G},\mathsf{BCQ})$, $g_C(Q)$, where C is a new relation symbol not in $\mathbf{S} \cup \mathsf{sig}(O)$ of arity $\mathsf{wd}(q)$, is an OMQ that falls in $(\mathsf{G},\mathsf{AQ}_0)$, while $g_C(\cdot)$ preserves containment. It turned out that $g_C(\cdot)$ preserves also FO-rewritability.

Lemma 8. *Let $Q = (\mathbf{S},O,q) \in (\mathsf{G},\mathsf{BCQ})$, and consider a predicate $C \notin \mathbf{S} \cup \mathsf{sig}(O)$ that has arity $\mathsf{wd}(q)$. It holds that*

$$Q \text{ is FO-rewritable} \iff g_C(Q) \text{ is FO-rewritable}.$$

Even though the reduction from $\mathsf{FORew}(\mathsf{G},\mathsf{BCQ})$ to $\mathsf{FORew}(\mathsf{G},\mathsf{AQ}_0)$ provided by Lemma 8 is exponential, we can still obtain the desired 2ExpTime upper bound for $\mathsf{FORew}(\mathsf{G},\mathsf{BCQ})$. This is because it increases the arity of the schema only polynomially, while the algorithm for $\mathsf{FORew}(\mathsf{G},\mathsf{AQ}_0)$ underlying Theorem 5 is double exponential only on the arity of the schema.

1.6 Reasoning over Finite Instances

The semantics of query evaluation for OMQs is defined in terms of *all* instances, including finite and infinite ones. In particular, recall that, given an OMQ $Q = (\mathbf{S},O,q(\bar{x}))$, a database $\mathfrak{D}$ over $\mathbf{S}$, and a tuple $\bar{a}$ of constants from $\mathsf{adom}(\mathfrak{D})$, we write $\mathfrak{D} \models Q(\bar{a})$ whenever, for every (finite or infinite) instance $\mathfrak{J} \supseteq \mathfrak{D}$ that satisfies O, it holds that $\mathfrak{J} \models q(\bar{a})$. However, there are applications in which reasoning over finite instances is more appropriate. A prime example of this is the area of data management, as databases are by definition finite objects.

We write $\mathfrak{D} \models_{fin} Q(\bar{a})$ whenever, for every *finite* instance $\mathfrak{J} \supseteq \mathfrak{D}$ that satisfies O, it holds that $\mathfrak{J} \models q(\bar{a})$. It is easy to show that for arbitrary sets of TGDs, entailment under arbitrary instances ($\models$) and entailment under finite instances ($\models_{fin}$) are, in general, different. Interestingly, this is not the case when we focus on guarded TGDs, a property known as *finite controllability*. In particular, a deep result in [5], relying on techniques from [33], established that for OMQs from $(\mathsf{G}, \mathsf{CQ})$ the entailment notions of $\models$ and $\models_{fin}$ coincide. This is formalized below.

Theorem 7. *Consider an OMQ $Q \in (\mathsf{G}, \mathsf{CQ})$ with data schema $\mathbf{S}$, a database $\mathfrak{D}$ over $\mathbf{S}$, and a tuple $\bar{a}$ of constants over $\mathsf{adom}(\mathfrak{D})$. It holds that*

$$\mathfrak{D} \models Q(\bar{a}) \iff \mathfrak{D} \models_{fin} Q(\bar{a}).$$

In very rough terms, the main idea behind the proof of Theorem 7 is to show the following: if there exists a counterexample to the fact that $\mathfrak{D} \models Q(\bar{a})$, i.e., an instance $\mathfrak{J} \supseteq \mathfrak{D}$ that satisfies O but $\mathfrak{J} \not\models q(\bar{a})$, then there is also a finite counterexample to it. Following a reasoning similar to that in Proposition 2, we can assume, w.l.o.g., that $\mathfrak{J}$ is acyclic. The finite counterexample $\mathfrak{J}_{fin}$ is then defined as a "nearly-acyclic covering" of $\mathfrak{J}$ with respect to $\mathfrak{D}$ and Q, which is a finite instance such that:

1. $\mathfrak{J}_{fin} \supseteq \mathfrak{D}$,
2. $\mathfrak{J}_{fin}$ satisfies O, and
3. There is a homomorphism from every set of at most $|q|$ atoms in $\mathfrak{J}_{fin}$ to $\mathfrak{J}$ that is the identity over $\mathsf{adom}(\mathfrak{D})$.

From (3), which is the "near-acyclicity" condition, one obtains that $\bar{a} \notin q(\mathfrak{J}_{fin})$. Towards a contradiction, assume that $\bar{a} \in q(\mathfrak{J}_{fin})$. Then, there is a homomorphism h from q to $\mathfrak{D}$ that maps $\bar{x}$ to $\bar{a}$. By composing h with the homomorphism given by (3) that maps the image of q under h to $\mathfrak{J}$, we obtain a homomorphism from q to $\mathfrak{J}$ that maps $\bar{x}$ to $\bar{a}$. This contradicts the fact $\mathfrak{J} \not\models q(\bar{a})$. Therefore, due to (1) and (2), we conclude that $\mathfrak{J}_{fin}$ is a counterexample to $\mathfrak{D} \models_{fin} Q(\bar{a})$.

As a direct corollary to Theorem 7, we obtain that also the notions of containment and first-order rewritability for OMQs based on guarded TGDs are invariant with respect to whether we consider $\models$ or $\models_{fin}$. Formally, consider two OMQs Q and Q' from $(\mathsf{G}, \mathsf{CQ})$ over the same data schema $\mathbf{S}$. We write $Q \subseteq_{fin} Q'$ if, for every database $\mathfrak{D}$ over $\mathbf{S}$ and tuple $\bar{a}$ of constants over $\mathsf{adom}(\mathfrak{D})$, it is the case that $\mathfrak{D} \models_{fin} Q(\bar{a})$ iff $\mathfrak{D} \models_{fin} Q'(\bar{a})$. Moreover, we say that Q is FO-rewritable *in the finite*, if there exists a first-order query ϕ_Q such that $\mathfrak{D} \models_{fin} Q(\bar{a})$ iff $\bar{a} \in \phi_Q(\mathfrak{D})$, for every database $\mathfrak{D}$ over $\mathbf{S}$ and tuple $\bar{a}$ of constants over $\mathsf{adom}(\mathfrak{D})$. We then have the following:

Corollary 1. *Consider $Q, Q' \in (\mathsf{G}, \mathsf{CQ})$ with the same data schema. It holds that:*

- *$Q \subseteq Q'$ iff $Q \subseteq_{fin} Q'$.*
- *Q is FO-rewritable iff Q is FO-rewritable in the finite.*

1.7 Conclusions

We have discussed in depth the crucial tasks of query evaluation, query containment, and first-order rewritability for guarded ontology-mediated queries. For query evaluation, we explained how classical results on the satisfiability problem for the guarded fragment of first-order logic can be applied. For query containment, we discussed how tree automata techniques can be used, while for first-order rewritability, we explained how techniques based on a more sophisticated automata model, known as cost automata, can be exploited. Finally, we discussed that the above problems are invariant with respect to whether we consider arbitrary or finite models.

There are still several open problems that deserve our attention:

- It is unclear whether the results on query containment and first-order rewritability presented for guarded ontology-mediated queries can be extended to the case where the ontology is an arbitrary set of guarded first-order sentences. Recall that in this generalized setting, query evaluation is decidable in 2ExpTime [5].
- The problem of extending guarded TGDs with additional features has been extensively studied in the literature. For example, guarded TGDs have been extended with default negation (a.k.a. negation as failure) in a series of papers. In [17], negation is interpreted according to the perfect model semantics, in [28] according to the well-founded semantics, while in [25] according to the stable model semantics. The query evaluation problem for OMQs based on the above extensions of guarded TGDs is by now well-understood. However, query containment and first-order rewritability have remained unexplored.
- Another relevant extension is guarded TGDs with disjunction, which has been studied in [15]. Again, query evaluation is well-understood, while query containment and first-order rewritability have remained open problems.

References

1. Miklós Ajtai and Yuri Gurevich. Datalog vs first-order logic. *J. Comput. Syst. Sci.*, 49(3):562–588, 1994.
2. Hajnal Andréka, István Németi, and Johan van Benthem. Modal Languages and Bounded Fragments of Predicate Logic. *J. Philosophical Logic*, 27(3):217–274, 1998.
3. Marcelo Arenas, Richard Hull, Wim Martens, Tova Milo, and Thomas Schwentick. Foundations of Data Management (Dagstuhl perspectives workshop 16151). *Dagstuhl Reports*, 6(4):39–56, 2016.
4. Jean-François Baget, Michel Leclère, Marie-Laure Mugnier, and Eric Salvat. On rules with existential variables: Walking the decidability line. *Artif. Intell.*, 175(9-10):1620–1654, 2011.
5. Vince Bárány, Georg Gottlob, and Martin Otto. Querying the Guarded Fragment. *Logical Methods in Computer Science*, 10(2), 2014.
6. Vince Bárány, Balder ten Cate, and Luc Segoufin. Guarded Negation. *J. ACM*, 62(3):22:1–22:26, 2015.
7. Pablo Barceló, Gerald Berger, Carsten Lutz, and Andreas Pieris. First-Order Rewritability of Frontier-Guarded Ontology-Mediated Queries. In *IJCAI*, pages 1707–1713, 2018.
8. Pablo Barceló, Gerald Berger, and Andreas Pieris. Containment for Rule-Based Ontology-Mediated Queries. In *PODS*, pages 267–279, 2018.

9. Catriel Beeri and Moshe Y. Vardi. The implication problem for data dependencies. In *ICALP*, pages 73–85, 1981.
10. Michael Benedikt, Pierre Bourhis, and Michael Vanden Boom. A step up in expressiveness of decidable fixpoint logics. In *LICS*, pages 817–826, 2016.
11. Michael Benedikt, Balder ten Cate, Thomas Colcombet, and Michael Vanden Boom. The complexity of boundedness for guarded logics. In *LICS*, pages 293–304, 2015.
12. Meghyn Bienvenu, Peter Hansen, Carsten Lutz, and Frank Wolter. First Order-Rewritability and Containment of Conjunctive Queries in Horn Description Logics. In *IJCAI*, pages 965–971, 2016.
13. Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-Order Rewritability of Atomic Queries in Horn Description Logics. In *IJCAI*, pages 754–760, 2013.
14. 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.
15. Pierre Bourhis, Marco Manna, Michael Morak, and Andreas Pieris. Guarded-based disjunctive tuple-generating dependencies. *ACM Trans. Database Syst.*, 41(4):27:1–27:45, 2016.
16. Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. *J. Artif. Intell. Res.*, 48:115–174, 2013.
17. Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.
18. Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.
19. Andrea Calì, Georg Gottlob, and Andreas Pieris. Towards more expressive ontology languages: The query answering problem. *Artif. Intell.*, 193:87–128, 2012.
20. Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP*, pages 139–150, 2009.
21. Thomas Colcombet and Nathanaël Fijalkow. The bridge between regular cost functions and omega-regular languages. In *ICALP*, pages 126:1–126:13, 2016.
22. Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.
23. Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs (preliminary report). In *STOC*, pages 477–490, 1988.
24. Haim Gaifman, Harry G. Mairson, Yehoshua Sagiv, and Moshe Y. Vardi. Undecidable optimization problems for database logic programs. *J. ACM*, 40(3):683–713, 1993.
25. Georg Gottlob, André Hernich, Clemens Kupke, and Thomas Lukasiewicz. Stable model semantics for guarded existential rules and description logics. In *KR*, 2014.
26. Georg Gottlob, Nicola Leone, and Francesco Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.
27. Erich Grädel. On the restraining power of guards. *J. Symb. Log.*, 64(4):1719–1742, 1999.
28. André Hernich, Clemens Kupke, Thomas Lukasiewicz, and Georg Gottlob. Well-founded semantics for extended datalog and ontological reasoning. In *PODS*, pages 225–236, 2013.
29. André Hernich, Carsten Lutz, Fabio Papacchini, and Frank Wolter. Dichotomies in ontology-mediated querying with the guarded fragment. In *PODS*, pages 185–199, 2017.
30. Nicola Leone, Marco Manna, Giorgio Terracina, and Pierfrancesco Veltri. Efficiently computable Datalog$\exists$ programs. In *KR*, 2012.
31. 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.
32. Riccardo Rosati. The limits of querying ontologies. In *ICDT*, pages 164–178, 2007.
33. Riccardo Rosati. On the finite controllability of conjunctive query answering in databases under open-world assumption. *J. Comput. Syst. Sci.*, 77(3):572–594, 2011.
34. Benjamin Rossman. Homomorphism preservation theorems. *J. ACM*, 55(3):15:1–15:53, 2008.
35. Oded Shmueli. Equivalence of DATALOG queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.