Bounded Query Rewriting Using Views

Citation for published version:
Cao, Y, Fan, W, Geerts, F & Lu, P 2018, 'Bounded Query Rewriting Using Views', *ACM Transactions on Database Systems*, vol. 43, no. 1, 6. https://doi.org/10.1145/3183673

Digital Object Identifier (DOI):
10.1145/3183673

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
ACM Transactions on Database Systems

Bounded Query Rewriting Using Views

A query Q in a language $\mathcal{L}$ has a *bounded rewriting* using a set of $\mathcal{L}$-definable views if there exists a query Q' in $\mathcal{L}$ such that given any dataset $\mathcal{D}$, $Q(\mathcal{D})$ can be computed by Q' that accesses only cached views and a small fraction D_Q of $\mathcal{D}$. We consider datasets $\mathcal{D}$ that satisfy a set of access constraints, which are a combination of simple cardinality constraints and associated indices, such that the size $|D_Q|$ of D_Q and the time to identify D_Q are independent of $|\mathcal{D}|$, no matter how big $\mathcal{D}$ is.

This paper studies the problem for deciding whether a query has a bounded rewriting given a set $\mathcal{V}$ of views and a set $\mathcal{A}$ of access constraints. We establish the complexity of the problem for various query languages $\mathcal{L}$, from Σ_3^p-complete for conjunctive queries (CQ), to undecidable for relational algebra (FO). We show that the intractability for CQ is rather robust even for acyclic CQ with fixed $\mathcal{V}$ and $\mathcal{A}$, and characterize when the problem is in PTIME. To make practical use of bounded rewriting, we provide an effective syntax for FO queries that have a bounded rewriting. The syntax characterizes a key subclass of such queries without sacrificing the expressive power, and can be checked in PTIME. Finally, we investigate $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting, when Q in $\mathcal{L}_1$ is allowed to be rewritten into a query Q' in another language $\mathcal{L}_2$. We show that this relaxation does not simplify the analysis of bounded query rewriting using views.

Categories and Subject Descriptors: H.2.1 [**DATABASE MANAGEMENT**]: Logical Design

General Terms: Design, Algorithms, Theory

Additional Key Words and Phrases: Bounded rewriting; big data; complexity

1. INTRODUCTION

To make query answering feasible in big datasets, practitioners have been studying scale independence [Armbrust et al. 2011; Armbrust et al. 2009; Armbrust et al. 2013]. The idea is to compute the answers $Q(\mathcal{D})$ to a query Q in a dataset $\mathcal{D}$ by accessing a bounded amount of data in $\mathcal{D}$, no matter how big the underlying $\mathcal{D}$ is.

This idea was formalized in [Fan et al. 2014; Fan et al. 2015]. As suggested in [Fan et al. 2014], nontrivial queries can be scale independent under a set $\mathcal{A}$ of access constraints, a form of cardinality constraints with associated indices. A query Q is *boundedly evaluable* [Fan et al. 2015] if for all datasets $\mathcal{D}$ that satisfy $\mathcal{A}$, $Q(\mathcal{D})$ can be computed from a fraction D_Q of $\mathcal{D}$, and the time for identifying and fetching D_Q, and hence the size $|D_Q|$ of D_Q are independent of $|\mathcal{D}|$. We identify D_Q by reasoning about the cardinality constraints in $\mathcal{A}$, and fetch D_Q by using the indices of $\mathcal{A}$.

Bounded evaluation has proven useful [Cao et al. 2014; Cao et al. 2015; Cao and Fan 2016]. Experimenting with several real-life datasets, it was shown that under a couple of hundreds of access constraints, 77% of randomly generated conjunctive queries (*a.k.a.* SPC queries) [Cao et al. 2014], 67% of relational algebra queries [Cao and Fan 2016], and 60% of graph pattern queries [Cao et al. 2015] are boundedly evaluable on average. Query plans for boundedly evaluable queries outperform commercial query engines by 3 orders of magnitude, and the gap gets larger on bigger data.

As an example of bounded evaluability, consider a Graph Search query of Facebook [Facebook 2013]: *find me all restaurants in NYC which I have not been to, but in which my friends have dined in May 2015*. A cardinality constraint imposed by Facebook is that a person can have at most 5000 friends [Facebook 2014]. Another one is that one dines at most once per day. Given these and another two similar constraints, the query can be answered by accessing 470000 tuples [Cao and Fan 2016], as opposed to billions of user tuples and trillions of friend tuples in the Facebook dataset [Grujic et al. 2014].

Still, many queries are not boundedly evaluable. Can we do better for such queries? An approach that has proven effective by practitioners is by making use of views [Armbrust et al. 2013]. The idea is to select and materialize a set $\mathcal{V}$ of small views, and answer Q on a dataset $\mathcal{D}$ by using views $\mathcal{V}(\mathcal{D})$ and an additional small fraction of $\mathcal{D}$. That is, we cache $\mathcal{V}(\mathcal{D})$ with fast access, and compute $Q(\mathcal{D})$ by using $\mathcal{V}(\mathcal{D})$ and by restricting costly I/O operations to (possibly big) $\mathcal{D}$. Many real-life queries that are not boundedly evaluable can be efficiently answered by using small views and by accessing a bounded amount of additional data in $\mathcal{D}$ [Armbrust et al. 2013].

Example 1.1. Consider a Graph Search query Q_0: *find movies that were released by Universal Studios in 2014, liked by people at NASA, and were rated 5*. The query is defined over a relational schema $\mathcal{R}_0$ consisting of four relations: (a) $\mathsf{person}(\mathsf{pid}, \mathsf{name}, \mathsf{affiliation})$, (b) $\mathsf{movie}(\mathsf{mid}, \mathsf{mname}, \mathsf{studio}, \mathsf{release})$, (c) $\mathsf{rating}(\mathsf{mid}, \mathsf{rank})$ for ranks of movies, and (d) $\mathsf{like}(\mathsf{pid}, \mathsf{id}, \mathsf{type})$, indicating that person pid likes item id of type, including but not limited to movies. Over $\mathcal{R}_0$, Q_0 is written as a conjunctive query:

$$Q_0(\mathsf{mid}) = \exists x_p, x'_p, y_m \ \big(\mathsf{person}(x_p, x'_p, \text{“NASA”}) \wedge \mathsf{movie}(\mathsf{mid}, y_m, \text{“Universal”}, \text{“2014”}) \wedge \mathsf{like}(x_p, \mathsf{mid}, \text{“movie”}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\big).$$

Consider a set $\mathcal{A}_0$ of two access constraints: (a) $\varphi_1 = \mathsf{movie}((\mathsf{studio}, \mathsf{release}) \to \mathsf{mid}, N_0)$, stating that each studio releases at most N_0 movies each year, where N_0 is obtained by aggregating $\mathcal{R}_0$ instances; an index is built on movie relation such that given any (studio, release) value, it returns (at most N_0) corresponding mids; we find that typically $N_0 \leqslant 100$ in practice; and (b) $\varphi_2 = \mathsf{rating}(\mathsf{mid} \to \mathsf{rank}, 1)$, stating that each movie has a unique rating; an index is built on rating to fetch rank as above.

Under $\mathcal{A}_0$, query Q_0 is not boundedly evaluable. Indeed, an instance $\mathcal{D}_0$ of $\mathcal{R}_0$ may have billions of person and like tuples [Grujic et al. 2014], and no constraints in $\mathcal{A}_0$ can help us identify a bounded fraction of these tuples to answer Q_0.

Nonetheless, suppose that we are given a view that collects movies liked by NASA folks, defined as the following conjunctive query:

$$V_1(\mathsf{mid}) = \exists x_p, x'_p, y'_m, z_1, z_2 \ \big(\mathsf{person}(x_p, x'_p, \text{“NASA”}) \wedge \mathsf{movie}(\mathsf{mid}, y'_m, z_1, z_2) \wedge \mathsf{like}(x_p, \mathsf{mid}, \text{“movie”})\big).$$

As will be seen later, Q_0 can be rewritten into a conjunctive query Q_ξ using V_1, such that for all instances $\mathcal{D}_0$ of $\mathcal{R}$ that satisfy $\mathcal{A}_0$, $Q_0(\mathcal{D}_0)$ can be computed by Q_ξ that accesses only $V_1(\mathcal{D}_0)$ and an additional $2N_0$ tuples from $\mathcal{D}_0$, no matter how big $\mathcal{D}_0$ grows. Here $V_1(\mathcal{D}_0)$ is a small set, much smaller than $\mathcal{D}_0$. □

To support scale independence using views, practitioners have developed techniques for selecting views, indexing the views for fast access and for incrementally maintaining the views [Armbrust et al. 2013]. However, there are still fundamental issues that call for a full treatment. How should we characterize scale independence using views? What is the complexity for deciding whether a query is scale independent given a set of views and access constraints? If the complexity of the problem is high, is there any systematic way that helps us make practical use of cached views for querying big data?

Contributions. This paper tackles these questions.

(1) Bounded rewriting. We formalize scale independence using views, referred to as *bounded rewriting* (Section 2). Consider a query language $\mathcal{L}$, a set $\mathcal{V}$ of $\mathcal{L}$-definable views and a database schema $\mathcal{R}$. Informally, under a set $\mathcal{A}$ of access constraints, we say that a query $Q \in \mathcal{L}$ has a *bounded rewriting* Q' in the same $\mathcal{L}$ using $\mathcal{V}$ if for each instance $\mathcal{D}$ of $\mathcal{R}$ that satisfies $\mathcal{A}$, there exists a fraction D_Q of $\mathcal{D}$ such that

— $Q(\mathcal{D}) = Q'(D_Q, \mathcal{V}(\mathcal{D}))$, and

— the time for identifying D_Q and hence the size $|D_Q|$ of D_Q are independent of $|\mathcal{D}|$.

That is, we compute the exact answers $Q(\mathcal{D})$ via Q' by accessing cached $\mathcal{V}(\mathcal{D})$ and a *bounded* fraction D_Q of $\mathcal{D}$. While $\mathcal{V}(\mathcal{D})$ may not be bounded, we can select small views following the methods of [Armbrust et al. 2013], which are cached with fast access. We formalize the notion in terms of query plans in a form of query trees commonly used in database systems [Ramakrishnan and Gehrke 2000], which have a bounded size M determined by our resources such as available processors and time.

(2) Complexity. We study *the bounded rewriting problem* (Section 3), referred to as VBRP($\mathcal{L}$) for a query language $\mathcal{L}$. Given a set $\mathcal{A}$ of access constraints, a query $Q \in \mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views, all defined on the same database schema $\mathcal{R}$, and a bound M, VBRP($\mathcal{L}$) is to decide whether under $\mathcal{A}$, Q has a bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ with a query plan of size no larger than M, referred to as *an M-bounded query plan*.

The need for studying VBRP($\mathcal{L}$) is evident: if Q has a bounded rewriting, then we can find efficient query plans to answer Q on possibly big datasets $\mathcal{D}$. We investigate VBRP($\mathcal{L}$) when $\mathcal{L}$ ranges over conjunctive queries (CQ, *i.e.,* SPC), unions of conjunctive queries (UCQ, *i.e.,* SPCU), positive FO queries ($\exists$FO$^+$, select-project-join-union queries) and first-order logic queries (FO, the full relational algebra). We show that VBRP is Σ_3^p-complete for CQ, UCQ and $\exists$FO$^+$, but it becomes undecidable for FO. In addition, we explore the impact of various parameters ($\mathcal{R}$, M, $\mathcal{A}$ and $\mathcal{V}$) of VBRP on its complexity.

(3) Acyclic conjunctive queries. Worse still, we show that the intractability of VBRP is quite robust (Section 4). It remains intractable for acyclic conjunctive queries (denoted by ACQ), when all parameters M, $\mathcal{R}$, $\mathcal{A}$ and $\mathcal{V}$ are fixed, and even when access constraints in the fixed $\mathcal{A}$ have quite restricted forms. In light of this, we give a characterization for VBRP(ACQ) to be in PTIME, and identify several sub-classes of ACQ and CQ for which VBRP is tractable under fixed M, $\mathcal{R}$, $\mathcal{A}$ and $\mathcal{V}$.

(4) Effective syntax. To cope with the undecidability of VBRP(FO) and the robust intractability of VBRP(CQ), we develop an effective syntax for FO queries that have a bounded rewriting (Section 5). For any $\mathcal{R}, \mathcal{V}, \mathcal{A}$ and M, we show that there exists a class of FO queries, referred to as *queries topped* by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, such that under $\mathcal{A}$,

(a) every FO query that has an M-bounded rewriting using $\mathcal{V}$ is equivalent to a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$;
(b) every query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has an M-bounded rewriting in FO using $\mathcal{V}$ that can be identified in PTIME; and
(c) it takes PTIME in $M, |Q|, |\mathcal{V}|, |\mathcal{R}|, |\mathcal{A}|$ to check whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, using an oracle to check whether views in $\mathcal{V}$ have bounded output (see below).

That is, topped queries make a *key* subclass of FO queries with a bounded rewriting using $\mathcal{V}$, without sacrificing their expressive power, along the same lines as safe-range queries for safe relational calculus (see, *e.g.,* [Abiteboul et al. 1995]). This allows us to reduce VBRP to syntactic checking of topped queries. Given a query Q, we can check syntactically whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in PTIME, by condition (c) above; if so, we can find a bounded rewriting in PTIME as warranted by condition (b); moreover, if Q has a bounded rewriting, then it can be transformed to a topped query by condition (a).

To check topped queries, we need to determine whether some views of $\mathcal{V}$ have *bounded output* $\mathcal{V}(\mathcal{D})$ over all datasets $\mathcal{D}$ that satisfy $\mathcal{A}$, *i.e.,* the size $|\mathcal{V}(\mathcal{D})|$ is bounded by a constant. This is to ensure bounded accesses to $\mathcal{D}$, since a query plan may filter and fetch data from $\mathcal{D}$ by using values from some views in $\mathcal{V}(\mathcal{D})$. This problem is, not surprisingly, undecidable for FO (Section 3). In light of this, we develop an effective syntax for FO queries with bounded output. That is, given $\mathcal{A}$ and $\mathcal{R}$, we identify a class of FO queries, referred to as *size-bounded queries*, such that under $\mathcal{A}$, an FO view (query) over $\mathcal{R}$ has bounded output if and only if it is equivalent to a size-bounded FO query, and it is in PTIME to check whether a query is size-bounded. We use this as a PTIME oracle when checking topped queries (condition (c)) above.

Experimenting with CDR (call detail record) data and queries from one of our industry collaborators, we find that bounded query rewriting using views improves more than 90% of their queries from 25 times to 5 orders of magnitude [Anonymous a].

(4) Rewriting in another language. Finally, we study $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting, when we are allowed to rewrite a query $Q \in \mathcal{L}_1$ into a query Q' in another query language $\mathcal{L}_2$ (Section 6). We reinvestigate the bounded rewriting problem in this setting, denoted by $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$. It is the problem for deciding, given a set $\mathcal{A}$ of access constraints, a query $Q \in \mathcal{L}_1$, a set $\mathcal{V}$ of $\mathcal{L}_1$-definable views, and a bound M, whether under $\mathcal{A}$, Q has a rewriting $Q' \in \mathcal{L}_2$ using $\mathcal{V}$ that has an M-bounded query plan.

One might be tempted to think that this relaxation would make our lives easier. However, we show that VBRP^+ remains Σ_3^p-hard for CQ-to-$\mathcal{L}_2$ when $\mathcal{L}_2$ ranges over UCQ, $\exists\mathsf{FO}^+$and FO; similarly when $\mathcal{L}_1$ is UCQ or $\exists\mathsf{FO}^+$.

This work is an effort to give a formal treatment of scale independence with views, an approach that has been put in action by practitioners. The complexity bounds reveal the inherent difficulty of the problem. The effective syntax, however, suggests a promising direction for making use of bounded rewriting. Various techniques are used in the proofs, including characterizations, algorithms and a wide range of reductions.

2. BOUNDED QUERY REWRITING

In this section we formalize bounded query plans and bounded query rewriting using views under access constraints. We start with a review of basic notions.

Database schema. A relational (database) schema $\mathcal{R}$ consists of a collection of relation schemas $(R_1, \ldots, R_n)$, where each R_i has a fixed set of attributes. We assume a countably infinite domain $\mathbf{U}$ of data values, on which instances $\mathcal{D}$ of $\mathcal{R}$ are defined. We use $|\mathcal{D}|$ to denote the size of $\mathcal{D}$, measured as the total *number of tuples* in $\mathcal{D}$. Instances of a single relation schema R are denoted by D.

Access schema. Following [Fan et al. 2015], we define an *access schema* $\mathcal{A}$ over a database schema $\mathcal{R}$ as a set of *access constraints* $\varphi = R(X \to Y, N)$, where R is a relation schema in $\mathcal{R}$, X and Y are sets of attributes of R, and N is a natural number.

For an instance D of R and an X-value $\bar{a}$ in D, we denote by $D_{R:Y}(X = \bar{a})$ the set $\{t[Y] \mid t \in D,\ t[X] = \bar{a}\}$, and write it as $D_Y(X = \bar{a})$ when R is clear in the context.

An instance D of R *satisfies* access constraint φ if

— for any X-value $\bar{a}$ in D, $|D_{R:Y}(X = \bar{a})| \leqslant N$; and
— there exists a function (referred to as an *index*) that given an X-value $\bar{a}$, returns $D_{R:XY}(X = \bar{a})$ (*i.e.,* $\{t[XY] \mid t \in D,\ t[X] = \bar{a}\}$) from D in $O(N)$ time.

Intuitively, an access constraint is a combination of a cardinality constraint and *an index on X for Y* (*i.e.,* the function). It tells us that given any X-value, there exist at most N distinct corresponding Y-values, and these Y values can be efficiently fetched by using the index. For instance, $\mathcal{A}_0$ described in Example 1.1 is an access schema.

Note that functional dependencies (FDs) are a special case $R(X \to Y, 1)$ of access constraints, *i.e.,* when bound $N = 1$, provided that an index is built from X to Y.

An instance $\mathcal{D}$ of $\mathcal{R} = \{R_1, \ldots, R_n\}$ *satisfies* access schema $\mathcal{A}$, denoted by $\mathcal{D} \models \mathcal{A}$, if the instance of R_i in $\mathcal{D}$ satisfies all the access constraints $\varphi = R_i(X \to Y, N)$ in $\mathcal{A}$.

Query classes. We express queries and views in the same language $\mathcal{L}$.

Following [Abiteboul et al. 1995], we consider atomic formulas that are either relation atoms $R(\bar{x})$ for $R \in \mathcal{R}$, or equality atoms $x = y$ or $x = c$, where $\bar{x}$, x and y variables and c is a constant. We consider the following classes $\mathcal{L}$ of queries built up from atomic formulas.

— Queries in first-order logic (FO) are inductively defined as follows: (a) atomic formulas are FO queries, and (b) if Q, Q_1 and Q_2 are FO queries, then so are $Q_1 \wedge Q_2$, $Q_1 \vee Q_2$, $\neg Q$, $\exists \bar{x}\, Q$ and $\forall \bar{x}\, Q$ (see Chapter 5 of [Abiteboul et al. 1995] for details).
— Positive existential FO queries ($\exists$FO$^+$) are FO queries in which negation ($\neg$) and universal quantification ($\forall$) are disallowed.
— Conjunctive queries (CQ) are $\exists$FO$^+$queries in which disjunction ($\vee$) is disallowed. A CQ query can be written as $Q(\bar{x}) = \exists \bar{x}'\, \phi(\bar{x}, \bar{x}')$, where $\phi(\bar{x}, \bar{x}')$ is a conjunction of atomic formulas (see Chapter 4 of [Abiteboul et al. 1995]).
— Unions of conjunctive queries (UCQ) are of the form $Q(\bar{x}) = Q_1(\bar{x}) \cup \cdots \cup Q_k(\bar{x})$, where $Q_i(\bar{x})$ is a CQ for $i \in [1, k]$. It is known that each $\exists$FO$^+$query Q can be written as a UCQ, which may possibly result in exponential increase in size $|Q|$ [Sagiv and Yannakakis 1980].

Bounded query rewriting. To simplify the definition, we present bounded query rewriting in terms of the relational algebra with projection π, selection σ, Cartesian product $\times$, union $\cup$, set difference $\setminus$ and renaming ρ. Consider an access schema $\mathcal{A}$ and a set $\mathcal{V}$ of views, both defined over the same database schema $\mathcal{R}$. We first extend the relational algebra under $\mathcal{A}$ with $\mathcal{V}$, denoted by $\mathsf{RA}_{\mathcal{A},\mathcal{V}}$, as follows:

$$Q ::= c \mid \mathsf{fetch}(X \in Q, R, Y) \mid V \mid \pi_Y(Q) \mid \sigma_C(Q) \mid Q \times Q \mid Q \cup Q \mid Q \setminus Q \mid \rho_{x \to y} Q,$$

where c is a constant, x and y are variables, V is a view in $\mathcal{V}$, $\pi_Y(Q), \sigma_C(Q), Q \times Q, Q \cup Q$, $Q \setminus Q$ and $\rho_{x \to y} Q$ denote projection, selection, Cartesian product, union, set difference and renaming as in the relational algebra, respectively; $\mathsf{fetch}(X \in Q, R, Y)$ requires that $\varphi = R(X \to Y, N)$ is an access constraint in $\mathcal{A}$ and that $Q(\mathcal{D})$ returns a set of X-attributes of R given an instance $\mathcal{D}$ of $\mathcal{R}$; for each $\bar{a}$ in $Q(\mathcal{D})$, it retrieves $D_{R:XY}(X = \bar{a})$ in the instance D of R in $\mathcal{D}$ by using the index associated with φ. Similarly, we also define $\mathcal{L}_{\mathcal{A},\mathcal{V}}$ for fragment $\mathcal{L}$ of $\mathsf{RA}_{\mathcal{A},\mathcal{V}}$ that corresponds to CQ, UCQ or $\exists$FO$^+$.

Intuitively, $\mathsf{RA}_{\mathcal{A},\mathcal{V}}$ revises the relational algebra by replacing direct access to relation R with $\mathsf{fetch}(X \in Q, R, Y)$, *i.e.,* it accesses instances of $\mathcal{R}$ only via the indices of access constraints in $\mathcal{A}$ only. It also allows accesses to cached views of $\mathcal{V}$.

Consider a query Q in a language $\mathcal{L}$. For a natural number M, we say that Q has an *M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under $\mathcal{A}$*, or simply *a bounded rewriting using* $\mathcal{V}$ when M and $\mathcal{A}$ are clear from the context, if there exists a query $Q' \in \mathcal{L}_{\mathcal{A},\mathcal{V}}$ such that (a) all constants in Q' are taken from Q, (b) for all instances $\mathcal{D}$ of $\mathcal{R}$ satisfying $\mathcal{A}$, $Q(\mathcal{D}) = Q'(\mathcal{D})$, and (c) there are at most M constants and operations ($\mathsf{fetch}, V, \pi, \sigma, \times, \cup, \setminus, \rho$) in Q'.

Intuitively, under $\mathcal{A}$, query Q is equivalent to Q', *i.e.,* Q' is a rewriting of Q using $\mathcal{V}$. Moreover, while Q' can retrieve entire cached views, its access to the underlying $\mathcal{D}$ must be via fetch operations only, by using the indices in the access constraints of $\mathcal{A}$. Hence only a bounded amount of data is fetched from $\mathcal{D}$. Here M is a threshold picked by users and is determined by available resources. The less resources we have, the smaller M we can afford. Without the bound M, we find that the query Q' is often of exponential length when experimenting with real-life data, which are not very practical; indeed, it would be EXPSPACE-hard to decide whether there exists a bounded rewriting even for CQ, by reduction from the problem for deciding bounded evaluability for CQ [Fan et al. 2015]. Hence we opt to let users specify M based on their resources.

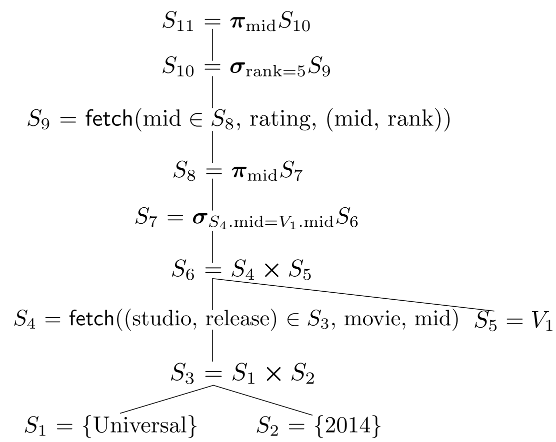

Fig. 1. A query plan ξ_0 using view V_1.

We next give an "operational semantics" of rewritings, by means of query plans.

Query plans. Following [Ramakrishnan and Gehrke 2000], we define a *query plan* using $\mathcal{V}$, denoted by $\xi(\mathcal{V}, \mathcal{R})$, as a tree T_ξ that satisfies the following two conditions.

(1) Each node u of T_ξ is labeled $S_i = \delta_i$, where S_i denotes a relation for partial results, and δ_i is as follows:

(a) $\{c\}$ for a constant c, if u is a leaf of T_ξ;
(b) a view V for $V \in \mathcal{V}$, if u is a leaf of T_ξ;
(c) $\mathsf{fetch}(X \in S_j, R, Y)$ if u has a single child v labeled with $S_j = \delta_j$, and S_j has attributes X; here X and Y are attributes in R and X can possibly be empty;
(d) $\pi_Y(S_j)$, $\sigma_C(S_j)$ or $\rho(S_j)$, if u has a single child v labeled with $S_j = \delta_j$; here Y is a set of attributes in S_j, and C is a condition defined on S_j; or
(e) $S_j \times S_l$, $S_j \cup S_l$ or $S_j \setminus S_l$, if u has two children v and v' labeled with $S_j = \delta_j$ and $S_l = \delta_l$, respectively.

Intuitively, given an instance $\mathcal{D}$ of $\mathcal{R}$, relations S_i's are computed by δ_i, bottom up in T_ξ [Ramakrishnan and Gehrke 2000]. More specifically, δ_i may (a) extract constant values, (b) access cached views $V(\mathcal{D})$, and (c) access $\mathcal{D}$ via a fetch operation, which, for each $\bar{a} \in S_j$, retrieves $D_{R:XY}(X = \bar{a})$ from the instance D of R in $\mathcal{D}$ on which the fetch operator is defined; it may also be a relational operation ((d) and (e) above).

(2) For each instance $\mathcal{D}$ of $\mathcal{R}$, the *result* $\xi(\mathcal{D})$ of applying $\xi(\mathcal{V}, \mathcal{R})$ to $\mathcal{D}$ is the relation S_n at root of T_ξ computed as above.

The *size* of plan ξ is the number of nodes in T_ξ. We use D_ξ to denote the bag of all tuples fetched for computing $\xi(\mathcal{D})$, *i.e.,* the multiset that collects tuples in $D_{R:XY}(X = \bar{a})$ for all $\mathsf{fetch}(X \in S_j, R, Y)$. Intuitively, it measures the amount of I/O operations used to access $\mathcal{D}$. Note that tuples retrieved from the cached views do not incur any I/O.

Example 2.1. A plan $\xi_0(V_1, \mathcal{R}_0)$ using view V_1 is depicted in Fig. 1. Given an instance $\mathcal{D}$ of $\mathcal{R}_0$, (a) it fetches the set S_4 of mids of all movies released by Universal Studios in 2014; (b) filters S_4 with mids in $V_1(\mathcal{D})$ via join, to get a subset S_8 of S_4 of movies liked by NASA folks; (c) fetches rating tuples using the mids of S_8; and (d) finds the set S_{11} of mids. One can verify that $\xi_0(\mathcal{D}) = Q_0(\mathcal{D})$ for Q_0 given in Example 1.1. □

Bounded plans. Consider an access schema $\mathcal{A}$ over $\mathcal{R}$. A query plan $\xi(\mathcal{V}, \mathcal{R})$ is said to *conform to* $\mathcal{A}$ if (a) for each $\mathsf{fetch}(X \in S_j, R, Y)$ operation in ξ, there exists an access constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$, and (b) there exists a constant N_ξ such that for all instances $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$, $|D_\xi| \leqslant N_\xi$.

That is, ξ can access cached views, and fetch D_ξ from $\mathcal{D}$ controlled by access schema $\mathcal{A}$. Plan ξ tells us how to retrieve D_ξ such that $\xi(\mathcal{D})$ is computed by using the data in D_ξ and $\mathcal{V}(\mathcal{D})$ only. Better still, D_ξ is *bounded*: $|D_\xi|$ is decided by Q and constants N in $\mathcal{A}$ only, and is independent of possibly big $|\mathcal{D}|$. The time for identifying and fetching D_ξ is also independent of $|\mathcal{D}|$ (assuming that given an X-value $\bar{a}$, it takes $O(N)$ time to fetch $D_{R:XY}(X = \bar{a})$ from the instance D of R in $\mathcal{D}$, via the index for $R(X \to Y, N)$).

Given a natural number M, we say that $\xi(\mathcal{V}, \mathcal{R})$ is *M-bounded for query Q using $\mathcal{V}$ under $\mathcal{A}$* if (a) ξ conforms to $\mathcal{A}$, (b) the size of ξ is at most M, (c) for all $\mathcal{D} \models \mathcal{A}$, $Q(\mathcal{D}) = \xi(\mathcal{D})$, *i.e.,* Q is equivalent to ξ on all instances $\mathcal{D} \models \mathcal{A}$, and (d) ξ only uses constants from Q. If these hold, then we write $\xi(Q, \mathcal{V}, \mathcal{R})$ to indicate that ξ answers Q.

If $\xi(Q, \mathcal{V}, \mathcal{R})$ is M-bounded under $\mathcal{A}$, then for all datasets $\mathcal{D}$ that satisfy $\mathcal{A}$, we can efficiently answer Q in $\mathcal{D}$ by carrying out ξ and accessing a bounded amount of data from $\mathcal{D}$ in addition to cached views $\mathcal{V}(\mathcal{D})$, as opposed to $Q(\mathcal{D})$ that accesses $\mathcal{D}$ only.

Example 2.2. Plan ξ_0 shown in Fig. 1 is 11-bounded for Q_0 using V_1 under $\mathcal{A}_0$. Indeed, (a) both fetch operations (S_4 and S_9) are controlled by the access constraints of $\mathcal{A}_0$, and (b) for any instance $\mathcal{D} \models \mathcal{A}_0$ of $\mathcal{R}_0$, ξ_0 accesses at most $2N_0$ tuples from $\mathcal{D}$, where N_0 is the constant in φ_1 of $\mathcal{A}_0$, since $|S_4| \leqslant N_0$ by φ_1, and $|S_9| \leqslant N_0$ by $S_8 \subseteq S_4$ and constraint φ_2 on rating in $\mathcal{A}_0$; and (c) eleven operations are conducted in total.

Observe that rating tuples in $\mathcal{D}$ are fetched by using S_8, which is obtained by relational operations on $V_1(\mathcal{D})$ and S_4. While V_1 is not boundedly evaluable under $\mathcal{A}_0$, the amount of data fetched from $\mathcal{D}$ is independent of $|\mathcal{D}|$. □

Bounded query rewriting (revisited). We conclude this section by rephrasing bounded query rewriting in terms of query plans. Consider a query Q in a language $\mathcal{L}$, a set $\mathcal{V}$ of $\mathcal{L}$-definable views, and an access schema $\mathcal{A}$, all defined over the same database schema $\mathcal{R}$. For a bound M, it is readily verified that Q has an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under $\mathcal{A}$ if it has an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$ such that ξ is *a query plan in $\mathcal{L}$, i.e.,* in each label $S_i = \delta_i$ of ξ,

— if $\mathcal{L}$ is CQ, then δ_i is a fetch, π, σ, $\times$ or ρ operation;
— if $\mathcal{L}$ is UCQ, δ_i can be fetch, π, σ, $\times$, ρ or $\cup$, and for any node labeled $\cup$, all its ancestors in the tree T_ξ of ξ are also labeled with $\cup$; that is, $\cup$ is at "the top level" only;
— if $\mathcal{L}$ is $\exists$FO$^+$, then δ_i is fetch, π, σ, $\times$, $\cup$ or ρ; and
— if $\mathcal{L}$ is FO, δ_i can be fetch, π, σ, $\times$, $\cup$, $\setminus$ or ρ.

One can verify that if ξ is a plan in $\mathcal{L}$, then there exists a query Q_ξ in $\mathcal{L}$ such that for all instances $\mathcal{D}$ of $\mathcal{R}$, $\xi(\mathcal{D}) = Q_\xi(\mathcal{D})$, and the size $|Q_\xi|$ of Q_ξ is linear in the size of ξ. Such query Q_ξ is unique up to equivalence. We refer to Q_ξ as the query *expressed by* ξ. Both ξ and Q_ξ may access $\mathcal{V}(\mathcal{D})$, and $\xi(\mathcal{D}) = Q_\xi(\mathcal{D})$ for all $\mathcal{D}$, either $\mathcal{D} \models \mathcal{A}$ or not.

Example 2.3. The CQ Q_0 of Example 1.1 has an 11-bounded rewriting in CQ using V_1 under $\mathcal{A}_0$. Indeed, ξ_0 of Fig. 1 is such a bounded plan, which expresses

$$Q_\xi(\mathsf{mid}) = \exists y_m \left(\mathsf{movie}(\mathsf{mid}, y_m, \text{“Universal”}, \text{“2014”}) \wedge V_1(\mathsf{mid}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\right).$$

It is a rewriting of Q_0 using V_1 in CQ. □

For the converse, if Q is a query in $\mathcal{L}$ using $\mathcal{L}$-definable views $\mathcal{V}$, then syntactic safety conditions on Q are required to ensure that there is a query plan ξ_Q in $\mathcal{L}$ such that $\xi_Q(\mathcal{D}, \mathcal{V}(\mathcal{D})) = Q(\mathcal{D}, \mathcal{V}(\mathcal{D}))$. We refer to Chapter 5 of [Abiteboul et al. 1995] for details on safety. We will come back to this issue in Section 5 when we present a syntactic fragment for bounded rewriting of FO queries using views under access constraints. Notations used in this paper are summarized in Table II in the electronic appendix.

3. DECIDING BOUNDED REWRITING

To make effective use of bounded rewriting, we need to settle *the bounded rewriting problem*, denoted by VBRP($\mathcal{L}$) for a query language $\mathcal{L}$ and stated as follows.

— INPUT: A database schema $\mathcal{R}$, a natural number M (in unary), an access schema $\mathcal{A}$, a query $Q \in \mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views all defined on $\mathcal{R}$.
— QUESTION: Under $\mathcal{A}$, does Q have an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$?

The problem VBRP($\mathcal{L}$) has, however, high complexity and can be even undecidable.

THEOREM 3.1. *Problem* VBRP($\mathcal{L}$) *is*

(1) Σ_3^p*-complete when* $\mathcal{L}$ *is* CQ, UCQ *or* $\exists$FO$^+$*; and*
(2) undecidable when $\mathcal{L}$ *is* FO. □

Below we first reveal the inherent complexity of VBRP($\mathcal{L}$) by studying problems embedded in it, and prove Theorem 3.1 for various $\mathcal{L}$ (Section 3.1). We then investigate the impact of parameters $\mathcal{R}$, $\mathcal{A}$, $\mathcal{V}$ and M on the complexity of VBRP($\mathcal{L}$) (Section 3.2).

3.1. The Bounded Rewriting Problem

To understand where the complexity of VBRP($\mathcal{L}$) arises, consider a problem embedded in it. Given an access schema $\mathcal{A}$, a query Q, a set $\mathcal{V}$ of views, and a query plan ξ of length M, it is to decide whether ξ is a bounded plan for Q using $\mathcal{V}$ under $\mathcal{A}$. This requires that we check the following: (a) Is the query Q_ξ expressed by ξ equivalent to Q under $\mathcal{A}$? (b) Does ξ conform to $\mathcal{A}$? None of these questions is trivial. To simplify the discussion, we focus on CQ for our examples.

$\mathcal{A}$-equivalence. Consider a database schema $\mathcal{R}$ and two queries Q_1 and Q_2 defined over $\mathcal{R}$. Under an access schema $\mathcal{A}$ over $\mathcal{R}$, we say that Q_1 is *$\mathcal{A}$-contained* in Q_2, denoted by $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, if for all instances $\mathcal{D}$ of $\mathcal{R}$ that satisfy $\mathcal{A}$, $Q_1(\mathcal{D}) \subseteq Q_2(\mathcal{D})$. We say that Q_1 and Q_2 are *$\mathcal{A}$-equivalent*, denoted by $Q_1 \equiv_{\mathcal{A}} Q_2$, if $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ and $Q_2 \sqsubseteq_{\mathcal{A}} Q_1$.

This is a notion weaker than the conventional notion of query equivalence $Q_1 \equiv Q_2$. The latter is to decide whether for all instances $\mathcal{D}$ of $\mathcal{R}$, $Q_1(\mathcal{D}) = Q_2(\mathcal{D})$, regardless of whether $\mathcal{D} \models \mathcal{A}$. Indeed, if $Q_1 \equiv Q_2$ then $Q_1 \equiv_{\mathcal{A}} Q_2$, but the converse does not hold. It is known that query equivalence for CQ is NP-complete [Chandra and Merlin 1977]. In contrast, it has been shown that $\mathcal{A}$-equivalence is Π_2^p-complete for CQ [Fan et al. 2015]. We show below that the upper bound remains valid for $\exists$FO$^+$.

LEMMA 3.2 [Fan et al. 2015]: *Given access schema $\mathcal{A}$, it is Π_2^p-complete to decide whether $Q_1 \equiv_{\mathcal{A}} Q_2$ and $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$, for queries Q_1 and Q_2 in* CQ, UCQ *or* $\exists$FO$^+$. □

Proof: Since it has been proven that it is Π_2^p-hard to decide whether $Q_1 \equiv_{\mathcal{A}} Q_2$ and $Q_1 \sqsubseteq_{\mathcal{A}} Q_2$ for CQ in [Fan et al. 2015], we only need to give an Σ_2^p algorithm to check whether $Q_1 \not\equiv_{\mathcal{A}} Q_2$ for $\exists$FO$^+$(similarly for $Q_1 \not\sqsubseteq_{\mathcal{A}} Q_2$). The algorithm works as follows.

(1) guess a disjunction Q_1^1 of Q_1, a disjunction Q_2^1 of Q_2, a valuation ν_1 of the tableau representation $(T_{Q_1^1}, \bar{u})$ of Q_1^1, and a valuation ν_2 of the tableau $(T_{Q_2^1}, \bar{u})$ of Q_2^1;

(2) check whether $\nu_1(T_{Q_1^1}) \not\models \mathcal{A}$ or $\nu_2(T_{Q_2^1}) \not\models \mathcal{A}$; if so, reject the current guess; otherwise, continue;
(3) check for all disjunctions Q_2^2 of Q_2, whether $\nu_1(\bar{u}) \notin Q_2^2(\nu_1(T_{Q_1^1}))$; if so, return true;
(4) check for all disjunctions Q_1^2 of Q_1, whether $\nu_2(\bar{u}) \notin Q_1^2(\nu_2(T_{Q_2^1}))$; if so, return true.

The tableau representation of a CQ $Q(\bar{x})$ is of the form $(T_Q, \bar{u})$, where T_Q is an "instance" of $\mathcal{R}$ obtained by taking all relation atoms in Q and (transitively) equating variables and constants as specified in the equality atoms in Q; the summary $\bar{u}$ of the tableau is obtained from $\bar{x}$ by equating variables and constants as described.

The correctness of the algorithm follows from the semantics of $Q_1 \equiv_{\mathcal{A}} Q_2$. For the complexity of the algorithm, step (2) is in PTIME, which follows from the definition of the access schema. Step (3) is in coNP, since we can check whether there exists a disjunction Q_2^2 of Q_2 such that $\nu_1(\bar{u}) \in Q_2^2(\nu_1(T_{Q_1^1}))$ as follows: guess a disjunction Q_2^2 of Q_2 and a homomorphism h from Q_2^2 to $\nu_1(T_{Q_1^1})$, and check whether $\nu_1(\bar{u}) \in Q_2^2(\nu_1(T_{Q_1^1}))$; if so, return true; otherwise, reject the guess. Similarly, step (4) is also in coNP. Hence the algorithm is in $\mathsf{NP}^{\mathsf{coNP}}$. That is, checking whether $Q_1 \equiv_{\mathcal{A}} Q_2$ is in Π_2^p for $\exists\mathsf{FO}^+$. □

Coming back to VBRP, for a query plan ξ and a query Q, we need to check whether ξ is a query plan for Q, *i.e.,* whether $Q_\xi \equiv_{\mathcal{A}} Q$, where Q_ξ is the query expressed by ξ. This step is Π_2^p-hard for CQ, and is undecidable when it comes to FO.

Bounded output. Another complication is introduced by views. To decide whether a query plan ξ is bounded for a query Q using $\mathcal{V}$ under $\mathcal{A}$, we need to verify that ξ conforms to $\mathcal{A}$. This *may* require to check whether a view $V \in \mathcal{V}$ has "bounded output".

Example 3.3. Recall schema $\mathcal{R}_0$, query Q_0, and access schema $\mathcal{A}_0$ of Example 1.1.
(a) Suppose that instead of V_1, a CQ view V_2 is given:

$$V_2(\mathsf{pid}) = \exists x'_p\ \mathsf{person}(\mathsf{pid}, x'_p, \text{"NASA"}).$$

Given an instance $\mathcal{D}$ of $\mathcal{R}_0$, $V_2(\mathcal{D})$ consists of people who work at NASA. Extend $\mathcal{A}_0$ to $\mathcal{A}_1$ by including $\varphi_3 = \mathsf{like}((\mathsf{pid}, \mathsf{id}) \to (\mathsf{pid}, \mathsf{id}, \mathsf{type}), 1)$, *i.e.,* $(\mathsf{pid}, \mathsf{id})$ is a key of relation like. Then Q_0 has a rewriting Q_2 using V_2:

$$Q_2(\mathsf{mid}) = \exists x_p, y_m\ \big(V_2(x_p) \land \mathsf{like}(x_p, \mathsf{mid}, \text{"movie"}) \land \mathsf{movie}(\mathsf{mid}, y_m, \text{"Universal"}, \text{"2014"}) \land \mathsf{rating}(\mathsf{mid}, 5)\big).$$

One can verify that Q_2 is a bounded rewriting of Q_0 using V_2 under $\mathcal{A}_1$ if and only if there exists a constant N_1 such that for all instances $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}_1$, then $|V_2(\mathcal{D})| \leqslant N_1$; that is, NASA has at most N_1 employees. For if it holds, then we can extract a set S of at most N_0 mids by using constraint φ_1 of $\mathcal{A}_1$ on movie, and select pairs $(\mathsf{pid}, \mathsf{mid})$ from $V_2(\mathcal{D}) \times S$ that are in a tuple (pid, mid, "movie") in the like relation, by making use of access constraint φ_3 given above. For each mid that passes the test, we check its rating via the index in φ_2, by accessing at most 1 tuple in rating. Putting these together, we access at most $N_1 \cdot N_0 + 2 \cdot N_0$ tuples from $\mathcal{D}$. Conversely, if the output of $V_2(\mathcal{D})$ is not bounded, then Q has no bounded rewriting using V_2 under $\mathcal{A}_1$.

(b) In contrast, when rewriting some queries, we *do not always* have to check whether a view has bounded output. As an example, consider a rewriting $Q(x) = Q_3(x) \land V_3(x)$ of query Q over a database schema $\mathcal{R}$, where V_3 is a view, and Q_3 has a bounded query plan under an access schema $\mathcal{A}$ and does not use any view. Then Q has a bounded rewriting under $\mathcal{A}$ no matter whether $|V_3(\mathcal{D})|$ is bounded or not for instances $\mathcal{D}$ of $\mathcal{R}$. Indeed, all fetching operations are conducted by Q_3; for each x-value a computed by $Q_3(x)$, we only need to validate whether $a \in V_3(\mathcal{D})$. This involves only cached $V_3(\mathcal{D})$, without accessing $\mathcal{D}$, and hence, $|V_3(\mathcal{D})|$ does not need to be bounded. □

To check whether views have a bounded output when it is necessary, we study *the bounded output problem*, denoted by $\mathsf{BOP}(\mathcal{L})$ and stated as follows:

— Input: A database schema $\mathcal{R}$, an access schema $\mathcal{A}$ and a query $V \in \mathcal{L}$, both over $\mathcal{R}$.
— QUESTION: Is there a constant N such that for all instances $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$, $|V(\mathcal{D})| \leqslant N$?

The analysis of the bounded output problem is also nontrivial.

THEOREM 3.4. *Problem* $\mathsf{BOP}(\mathcal{L})$ *is*

(1) coNP-*complete when* $\mathcal{L}$ *is* CQ, UCQ *or* $\exists\mathsf{FO}^+$*; and*
(2) undecidable when $\mathcal{L}$ *is* FO.

When database schema $\mathcal{R}$ *and access schema* $\mathcal{A}$ *are both fixed,* BOP *remains* coNP-*hard for* CQ, UCQ *and* $\exists\mathsf{FO}^+$*, and is still undecidable for* FO. □

Proof: We first show that BOP is coNP-complete for CQ, UCQ and $\exists\mathsf{FO}^+$, and then prove that it is undecidable for FO.

(1) CQ**,** UCQ **and** $\exists\mathsf{FO}^+$. We show that BOP is coNP-hard for CQ and is in coNP for $\exists\mathsf{FO}^+$. The proof is based on a characterization of bounded-output $\exists\mathsf{FO}^+$queries, *i.e.,* a query Q in $\exists\mathsf{FO}^+$for which there exists a constant N such that $|Q(\mathcal{D})| \leqslant N$ for any $\mathcal{D} \models \mathcal{A}$. To introduce the characterization, we first present two notations.

Notations. When considering a CQ Q posed on instances that satisfy a set $\mathcal{A}$ of access constraints, it will often be convenient to regard Q as an UCQ consisting of special CQ's Q_e, referred to as the *element queries of* Q *under* $\mathcal{A}$. The idea of element queries was mentioned in [Fan et al. 2015] but was not explored there. To define element queries we use the tableau formalism of CQ (cf. [Abiteboul et al. 1995], Chapter 4). As remarked earlier, the tableau representation of a CQ $Q(\bar{x})$ is of the form $(T_Q, \bar{u})$.

Consider an instance $\mathcal{D}$ of $\mathcal{R}$ such that $\mathcal{D} \models \mathcal{A}$. Let $\bar{a} \in Q(\mathcal{D})$. This implies that there exists a homomorphism $h : T_Q \to \mathcal{D}$ such that $h(\bar{u}) = \bar{a}$ and $h(T_Q) \models \mathcal{A}$. It is easy to verify that there is a conjunction ψ of equality conditions among variables and constants in Q such that when considering $Q_e = Q \wedge \psi$, we have that for the tableau $(T_{Q_e}, \bar{u}')$ of Q_e, (i) $h : T_{Q_e} \to \mathcal{D}$ is a homomorphism such that $h(\bar{u}') = \bar{a}$; and (ii) $T_{Q_e} \models \mathcal{A}$, where we view T_{Q_e} as an instance of $\mathcal{R}$, by treating variables as constants. We call such Q_e's element queries and say that Q_e *satisfies* $\mathcal{A}$ because $T_{Q_e} \models \mathcal{A}$. In general, we say that a CQ Q *satisfies* $\mathcal{A}$ if its tableau satisfies $\mathcal{A}$. Observe that any element query Q_e of Q is contained in Q. Indeed, any Q_e is obtained from Q by adding equality conditions and Q_e is therefore more specific than Q. Conversely, Q is $\mathcal{A}$-contained in the union of all of its element queries. That is, $Q \sqsubseteq_{\mathcal{A}} Q_{e_1} \cup \cdots \cup Q_{e_n}$. Indeed, given an instance $\mathcal{D}$ of $\mathcal{R}$, for any $\bar{a} \in Q(\mathcal{D})$ there exists an element query Q_e such that $\bar{a} \in Q_e(\mathcal{D})$. Hence, $Q \equiv_{\mathcal{A}} Q_{e_1} \cup \cdots \cup Q_{e_n}$.

Note that Q has at most exponentially many element queries under $\mathcal{A}$, since there are $O(2^{|Q|})$ possible ψ. Furthermore, an element query may not be satisfiable. Indeed, this happens when the conditions in ψ equate two different constants in $Q_e = Q \wedge \psi$. The satisfiability of element queries can be checked in PTIME. Therefore, in the sequel we consider *w.l.o.g.* only satisfiable element queries.

For instance, consider $\mathcal{R}$ with a single relation $R(X, Y)$, query $Q(x) = R(y, x_1) \wedge R(y, x_2) \wedge R(y, x_3) \wedge R(x_3, x) \wedge (x_1 = 1) \wedge (x_2 = 2) \wedge (y = k)$, for a constant k and access schema $\mathcal{A} = \{R(X \to Y, 2)\}$. Example element queries of Q include $Q_1(x) = Q(x) \wedge (x_1 = x_2)$, $Q_2(x) = Q(x) \wedge (x_2 = x_3)$, $Q_3(x) = Q(x) \wedge (x_1 = x_3)$ and $Q_4(x) =$

$Q(x) \wedge ((x_1 = x_3) \wedge (x_1 = x_2) \wedge (x_2 = x_3) \wedge (x_3 = x))$. Note that Q_1 and Q_4 are not satisfiable.

As we will show below, element queries also make the bounded output analysis easier. When the tableau of Q does not satisfy $\mathcal{A}$, it is unclear what variables in Q have a bound on their valuations. Taking $Q(x)$ above as an example, we do not know whether there exists a bound on the valuation of x_3. Indeed, the access constraints only bound variables in atoms that occur in the Y attributes of R. In contrast, when considering element queries $Q_2(x)$ and $Q_3(x)$, we can easily see the bounds on valuations of x_3. Indeed, x_3 is bound to constant "2" in Q_2 and to constant "1" in Q_3.

Let Q be a CQ that satisfies $\mathcal{A}$. For example, Q could be an element query. To simplify the discussion, we assume *w.l.o.g.* that relation atoms in Q do not contain constants. Instead, all constants appear in equality conditions of the form $x = a$ for some variable x and constant a. We denote by $\mathsf{cvars}(Q)$ the set of *constant variables* in Q that are (transitively) equal to some constant due to the equality conditions in Q, and by $\mathsf{vars}(Q)$ the set of remaining variables in Q, *i.e.,* those that are not equal to some constant.

We also need a notion of covered variables [Fan et al. 2015]. We define the set of *covered variables of* Q *under* $\mathcal{A}$, denoted by $\mathsf{cov}(Q, \mathcal{A})$, and computed as follows:

(1) $\mathsf{cov}_0(Q, \mathcal{A}) := \emptyset$;
(2) For $i > 0$, do the following steps until no further variables in $\mathsf{vars}(Q)$ can be added:
- $\mathsf{cov}_i(Q, \mathcal{A}) := \mathsf{cov}_{i-1}(Q, \mathcal{A})$;
- if there exist an atom $R(\bar{x}, \bar{y}, \bar{z})$ in Q and an access constraint $R(X \to Y, N)$ in $\mathcal{A}$, where $\bar{x}$ corresponds to X and $\bar{y}$ corresponds to Y, and if all non-constant variables in $\bar{x}$ are in $\mathsf{cov}_{i-1}(Q, \mathcal{A})$, then $\mathsf{cov}_i(Q, \mathcal{A})$ is expanded by including all the non-constant variables in $\bar{y}$ that are not already in $\mathsf{cov}_{i-1}(Q, \mathcal{A})$.

We denote by $\mathsf{cov}(Q, \mathcal{A})$ the result set of the process. Note that $\mathsf{cov}(Q, \mathcal{A})$ consists of non-constant variables only. Indeed, constant variables have bounded output (as they equal some constant) and hence do not affect the boundedness of a query.

Example 3.5. Consider the above element query $Q_2(x) = Q(x) \wedge (x_2 = x_3)$. The constant variables in $\mathsf{cvars}(Q_2)$ are y, x_1, x_2, x_3. The only non-constant variable is x, *i.e.,* $\mathsf{vars}(Q_2) = \{x\}$. Let us compute $\mathsf{cov}(Q_2, \mathcal{A})$. Initially, $\mathsf{cov}_0(Q_2, \mathcal{A}) := \emptyset$. The only atom in Q_2 that contains the non-constant variable x is $R(x_3, x)$. If we consider access constraint $R(X \to Y, 2) \in \mathcal{A}$, all non-constant variables in $R(x_3, x)$ corresponding to the X-attribute belong to $\mathsf{cov}_0(Q_2, \mathcal{A})$. Indeed, no non-constant variables are present in the X-attribute of atom $R(x_3, x)$. Hence, $\mathsf{cov}_1(Q_2, \mathcal{A}) = \{x\}$, *i.e.,* the non-constant variable x is added. Since x is the only variable in $\mathsf{vars}(Q_2)$, $\mathsf{cov}(Q_2, \mathcal{A}) = \mathsf{cov}_1(Q_2, \mathcal{A}) = \{x\}$. $\square$

<u>*Characterizations*</u>. Given these, we start with bounded-output queries that satisfy $\mathcal{A}$.

LEMMA 3.6. *A* CQ *query* $Q(\bar{v})$ *that satisfies* $\mathcal{A}$ *has bounded output if and only if all non-constant variables in* $\bar{v}$ *belong to* $\mathsf{cov}(Q, \mathcal{A})$. $\square$

Proof: ($\Leftarrow$) First assume that all non-constant variables in $\bar{v}$ belong to $\mathsf{cov}(Q, \mathcal{A})$. Let $Q'(\bar{u})$ be the CQ obtained from $Q(\bar{v})$ by removing all existential quantifiers, *i.e.,* $Q(\bar{v}) = \exists \bar{z}\, Q'(\bar{u})$, where $\bar{z}$ consists of all variables (constant or non-constant) in $\bar{u} \setminus \bar{v}$. It is easy to see that $\mathsf{cov}(Q, \mathcal{A}) = \mathsf{cov}(Q', \mathcal{A})$. Indeed, no distinction is made between free and quantified variables in the definition of covered variables of a query under access constraints. We show that for all variables $x \in \mathsf{cov}(Q', \mathcal{A})$, $Q''_x(x) = \exists \bar{u} \setminus \{x\}\, Q'(\bar{u})$ has bounded output, by induction on the computation of $\mathsf{cov}(Q', \mathcal{A})$. This suffices, for if the statement holds, then $Q(\bar{v})$ has bounded output, since $Q(\bar{v}) = \exists \bar{z}\, Q'(\bar{u})$ and $Q'(\bar{u})$ is contained in $Q''_{u_1}(u_1) \wedge \cdots \wedge Q''_{u_k}(u_k) \wedge u_{k+1} = c_{k+1} \wedge \ldots \wedge u_n = c_n$, where $(u_1, \ldots, u_k)$ are non-constant variables in $\bar{u}$, "specialized query" $Q''_{u_j}(u_j)$ takes parameter u_j, and for each $i \in [k+1, n]$, u_i is a constant variable in $\bar{u}$ that is equal to constant c_i.

For the base case, $i = 0$ and $\mathsf{cov}_0(Q', \mathcal{A}) = \emptyset$. Clearly, $\exists \bar{u}\, Q'(\bar{u})$ is a Boolean query and hence has bounded output.

Assume that the induction hypothesis holds for any $j \in [0, i-1]$. That is, for any variable $y \in \mathsf{cov}_{i-1}(Q', \mathcal{A})$, $Q''_y(y) = \exists \bar{u} \setminus \{y\}\, Q'(\bar{u})$ has bounded output.

We next show that the statement holds for each variable in $\mathsf{cov}_i(Q', \mathcal{A})$. Let y be a variable in $\mathsf{cov}_i(Q', \mathcal{A}) \setminus \mathsf{cov}_{i-1}(Q', \mathcal{A})$. Suppose that y is added to $\mathsf{cov}_i(Q', \mathcal{A})$ via access constraint $R(X \to Y, N) \in \mathcal{A}$ and atom $R(\bar{x}, \bar{y}, \bar{z})$ in Q'. Then $y \in \bar{y}$, and any (non-constant) variable $x \in \bar{x}$ must be in $\mathsf{cov}_{i-1}(Q', \mathcal{A})$. From the induction hypothesis we know that $Q''_x(x) = \exists \bar{u} \setminus \{x\}\, Q'(\bar{u})$ has bounded output. That is, there exists a natural number N_x such that for any instance $\mathcal{D}$ satisfying $\mathcal{A}$, $|Q''_x(\mathcal{D})| \leqslant N_x$. Moreover, since $\exists \bar{u} \setminus \bar{x}\, Q'(\bar{u})$ is contained in $Q'''(\bar{x}) = \bigwedge_{x_i \in \bar{x}} Q''_{x_i}(x_i)$, and for any $\mathcal{D} \models \mathcal{A}$, $|Q'''(\mathcal{D})| \leqslant M = \prod_{x_i \in \bar{x}} N_{x_i}$, we can see that $\exists \bar{u} \setminus \bar{x}\, Q'(\bar{u})$ also has bounded output. From the definition of access constraints, we can further deduce that $\exists \bar{u} \setminus \bar{y}\, Q'(\bar{u})$ generates at most $M \times N$ tuples when evaluated on $\mathcal{D}$. In particular, this holds for $Q''_y(y) = \exists \bar{u} \setminus \{y\} Q'(\bar{u})$; thus the statement also holds for y. The argument works for any y in $\mathsf{cov}_i(Q', \mathcal{A}) \setminus \mathsf{cov}_{i-1}(Q', \mathcal{A})$. Hence for any $y \in \mathsf{cov}_i(Q', \mathcal{A})$, $Q''_y(y) = \exists \bar{u} \setminus \{y\}\, Q'(\bar{u})$ has bounded output.

($\Rightarrow$) Conversely, assume that there exists a (non-constant) variable $v \in \bar{v}$ such that $v \notin \mathsf{cov}(Q, \mathcal{A})$. Note that v is a free variable in $Q'(\bar{v})$. Let $Q'(v) = \exists \bar{v} \setminus \{v\} Q(\bar{v})$. It suffices to show that Q' does not have bounded output. We have that $(v) \in Q'(T_Q)$, where $(T_Q, \bar{u}_Q)$ is the tableau representation of Q. We next construct instances D_K of $\mathcal{R}$ for all natural numbers $K > 0$ such that $|Q'(T_Q \cup D_K)| > K \times |Q'(T_Q)|$ and $T_Q \cup D_K \models \mathcal{A}$. Hence, Q' (and thus also Q) does not have bounded output.

We illustrate the construction of D_K for $K = 1$. Let D_1 consist of a copy of T_Q. That is, D_1 is T_Q except that every variable z that is not in $\mathsf{cov}(Q, \mathcal{A})$ is replaced by a primed copy z'. Note that when considering tableaux, we do not need to differentiate between constant and non-constant variables, since constant variables correspond to constants in the tableau representation. We can show that $\{(v), (v')\} \subseteq Q'(T_Q \cup D_1)$, since $(v) \in Q'(T_Q)$ and $v \notin \mathsf{cov}(Q, \mathcal{A})$. Indeed, because $(v) \in Q'(T_Q)$, there exists a homomorphism h from Q' to T_Q. Then we can obtain a homomorphism h_1 from Q' to D_1 as follows: for each variable x in Q', if $h(x) \in \mathsf{cov}(Q, \mathcal{A})$ or $h(x)$ is a constant, then $h_1(x) = h(x)$; otherwise $h(x)$ is a variable z such that $z \notin \mathsf{cov}(Q, \mathcal{A})$, and we define $h_1(x) = z'$, the primed copy of z. We can verify that h_1 is a homomorphism of Q' to D_1. Since $(v) \in Q'(T_Q)$ and $v \notin \mathsf{cov}(Q, \mathcal{A})$, we know that $(v') \in Q'(D_1)$. By the monotonicity of CQ, we have that $\{(v), (v')\} \subseteq Q'(T_Q \cup D_1)$. Thus $|Q'(T_Q \cup D_1)| > |Q'(T_Q)|$. It remains to show that $T_Q \cup D_1$ satisfies $\mathcal{A}$. We show this by contradiction. Suppose that $(T_Q \cup D_1) \not\models R(X \to Y, N)$ for some access constraint $R(X \to Y, N)$ in $\mathcal{A}$. This means that there exist $N+1$ tuples $t_1, \ldots, t_{N+1}$ in $T_Q \cup D_1$ such that $t_1[X] = \cdots = t_{N+1}[X]$, but $t_i[Y] \neq t_j[Y]$ for all $i \neq j$, $i, j \in [1, N+1]$. We distinguish the following three cases:

(a) When $t_1[X]$ consists of constants and variables in $\mathsf{cov}(Q, \mathcal{A})$. In this case, each $t_i[Y]$ also consists of constants and variables in $\mathsf{cov}(Q, \mathcal{A})$ by the access constraint $R(X \to Y, N)$ and the computation of $\mathsf{cov}(Q, \mathcal{A})$. Since all variables in $t_i[X \cup Y]$ ($i \in [1, N+1]$) are contained in $\mathsf{cov}(Q, \mathcal{A})$, these variables are also in T_Q. By the construction of $T_Q \cup D_1$, there must exist $N+1$ tuples $s_1, \ldots, s_{N+1}$ in T_Q such that $s_i[X \cup Y] = t_i[X \cup Y]$ for $i \in [1, N+1]$. This, however, contradicts the assumption that $T_Q \models \mathcal{A}$. Note that by the construction of D_1, there also exist $N+1$ tuples $s'_1, \ldots, s'_{N+1}$ in D_1 such that $s'_i[X \cup Y] = t_i[X \cup Y]$ for

I_{01} =

A
1
0

$I_{\vee}$ =

B	A_1	A_2
0	0	0
1	0	1
1	1	0
1	1	1

$I_{\wedge}$ =

B	A_1	A_2
0	0	0
0	0	1
0	1	0
1	1	1

$I_{\neg}$ =

A	$\bar{A}$
0	1
1	0

Fig. 2. Relation instances used in the proof of Theorem 3.4.

$i \in [1, N+1]$. For example, consider database schema $\mathcal{R} = \{R(X, Y, Z)\}$, access schema $\mathcal{A} = \{R((X, Y) \to Z, 1)\}$, and $Q = R(1, 1, z_1) \wedge R(1, z_1, z_2) \wedge R(1, z_3, z_4)$. Then $\mathsf{cov}(Q, \mathcal{A}) = \{z_1, z_2\}$, $D_1 = \{R(1, 1, z_1), R(1, z_1, z_2), R(1, z_3', z_4')\}$, and $T_Q \cup D_1 = \{R(1, 1, z_1), R(1, z_1, z_2), R(1, z_3, z_4), R(1, z_3', z_4')\}$. Since all variables in $R(1, z_1, z_2)$ are in $\mathsf{cov}(Q, \mathcal{A})$, T_Q contains the tuple $R(1, z_1, z_2)$, and D_1 also contains $R(1, z_1, z_2)$.

(b) When $t_1[X]$ consists of constants and variables in T_Q, but at least one of these variables is not in $\mathsf{cov}(Q, \mathcal{A})$. By the construction of $T_Q \cup D_1$, only tuples in T_Q can contain variables, which are in T_Q, but are not in $\mathsf{cov}(Q, \mathcal{A})$, then $t_1, \ldots, t_{N+1}$ are tuples in T_Q. This contradicts again the assumption that $T_Q \models \mathcal{A}$. For the example in case (a), since $z_3, z_4 \notin \mathsf{cov}(Q, \mathcal{A})$, only T_Q contains the tuple $R(1, z_3, z_4)$.

(c) When $t_1[X]$ contains a primed copy x' of a variable x in T_Q. In this case, $t_1, \ldots, t_{N+1}$ are tuples in D_1. Similar to case (a), we can prove that $D_1 \not\models \mathcal{A}$. But since D_1 is a copy of T_Q, where every variable z that is not in $\mathsf{cov}(Q, \mathcal{A})$ is replaced by a primed copy z', we have that $D_1 \models \mathcal{A}$, a contradiction. For the example in case (a), since the primed variables z_3' and z_4' can only appear in D_1, $R(1, z_3', z_4')$ only exists in D_1.

Putting these together, we can conclude that $T_Q \cup D_1 \models A$.

For $K > 1$, D_K is defined to consist of K distinct copies of T_Q. Along the same lines, one can verify that $Q'(T_Q \cup D_K)$ contains at least K distinct copies of v, and thus $|Q'(T_Q \cup D_K)| > K \times |Q'(T_Q)|$. Moreover, $T_Q \cup D_K \models \mathcal{A}$.

Hence, if $Q(\bar{u})$ has bounded output, then each variable $u \in \bar{u}$ must be in $\mathsf{cov}(Q, \mathcal{A})$. □

From Lemma 3.6 it follows that we can characterize bounded-output queries in $\exists\mathsf{FO}^+$even when they do not necessarily satisfy $\mathcal{A}$. Indeed, recall from Section 2 that every $\exists\mathsf{FO}^+$query Q is equivalent to a UCQ query $Q_1 \cup \cdots \cup Q_n$. Furthermore, each CQ Q_i is $\mathcal{A}$-equivalent to a UCQ consisting of Q_i's element queries. That is, $Q \equiv_{\mathcal{A}} \bigcup_{i \in [1,n]} (Q^e_{i,1} \cup \cdots \cup Q^e_{i,n_i})$ where $Q_i \equiv_{\mathcal{A}} Q^e_{i,1} \cup \cdots \cup Q^e_{i,n_i}$ and each $Q^e_{i,j}$ $(j \in [1, n_i])$ is an element query of Q_i under $\mathcal{A}$. Obviously, Q has bounded output if and only if each element query $Q^e_{i,j}$ has bounded output. Furthermore, by definition, each element query $Q^e_{i,j}$ is a CQ that satisfies $\mathcal{A}$. Thus the characterization below is immediate.

LEMMA 3.7. *For a query $Q(\bar{x})$ in* CQ *(*UCQ, $\exists$FO$^+$*) and an access schema $\mathcal{A}$, $Q(\bar{x})$ has bounded output if and only if for every element query $Q_e(\bar{x}')$ of $Q(\bar{x})$, all (non-constant) variables in $\bar{x}'$ belong to* $\mathsf{cov}(Q_e, \mathcal{A})$. □

We are now ready to show that BOP is coNP-hard for CQ and is in coNP for $\exists\mathsf{FO}^+$.

Lower bound. We show that BOP is coNP-hard for CQ by reduction from the complement of the 3SAT problem. The 3SAT problem is to decide, given a propositional formula $\psi = C_1 \wedge \cdots \wedge C_r$ defined over variables $X = \{x_1, \ldots, x_m\}$, whether there exists a truth assignment for X that satisfies ψ. Here for each $i \in [1, r]$, clause C_i is of the form $\ell^i_1 \vee \ell^i_2 \vee \ell^i_3$, and for each $j \in [1, 3]$, literal ℓ^i_j is either a variable x_l in X or the negation $\neg x_l$ of x_l. It is known that 3SAT is NP-complete (cf. [Garey and Johnson 1979]).

Given an instance ψ of 3SAT, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, and a CQ query $Q(w)$ such that $Q(w)$ has bounded output if and only if ψ is false.

(a) The database schema $\mathcal{R}$ contains the following two kinds of relation schemas: (i) $R_{01}(A)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, and $R_\neg(A, \bar{A})$, to store constant relations encoding truth values, disjunction, conjunction and negation of variables, respectively, as shown in Figure 2; and (ii) $R_o(I, X)$ to constrain the output.

(b) The access schema $\mathcal{A}$ contains (i) four constraints to ensure valid instances of Figure 2: $R_{01}(\emptyset \to A, 2)$, $R_\vee(\emptyset \to (B, A_1, A_2), 4)$, $R_\wedge(\emptyset \to (B, A_1, A_2), 4)$, $R_\neg(\emptyset \to (A, \bar{A}), 2)$; intuitively, they constrain the number of tuples in the corresponding instances; and (ii) one access constraint $R_o(I \to X, 2)$ to bound the output.

(c) The query Q in CQ is defined as follows:

$$Q(w) = \exists \bar{x}, w_1, k\bigg(Q_c() \wedge Q_X(\bar{x}) \wedge Q_\psi(\bar{x}, w_1) \wedge R_{01}(w_1) \wedge R_o(k, 1) \wedge R_o(k, w_1) \wedge R_o(k, w)\bigg),$$

where Q_c, Q_X, and Q_ψ are in CQ. Query Q_c is to ensure that the instances of R_{01}, $R_\vee$, $R_\wedge$, and $R_\neg$ contain all the tuples shown in Figure 2. For example, to include the two tuples in I_{01}, Q_c contains $R_{01}(0) \wedge R_{01}(1)$. Together with the constraints of $\mathcal{A}$, this implies that whenever $Q(\mathcal{D}) \neq \emptyset$ for an instance $\mathcal{D} \models \mathcal{A}$, $Q_c(\mathcal{D}) = \{()\}$ and hence $\mathcal{D}$ consists of the instances I_{01}, $I_\vee$, $I_\wedge$, $I_\neg$ of Figure 2, and a non-empty instance I_o of R_o.

Query $Q_X(\bar{x})$ is to ensure that $\bar{x}$ is a truth-assignment of X. From the definition of Q_c and the constraint $R_{01}(\emptyset \to A, 2)$, $Q_X(\bar{x})$ can be defined as $\bigwedge_{1 \leqslant i \leqslant m} R_{01}(x_i)$.

Query $Q_\psi(\bar{x}, w_1)$ is defined such that when given a truth-assignment μ_X encoded by $\bar{x}$, it sets $w_1 = 1$ if $\psi(\mu_X)$ is true and sets $w_1 = 0$ otherwise. It is easily verified that Q_ψ can be expressed in CQ by leveraging R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$.

Finally, consider the sub-query $R_o(k, 1) \wedge R_o(k, w_1) \wedge R_o(k, w)$. If Q_ψ sets $w_1 = 1$ then we know from $R_o(I \to X, 2) \in \mathcal{A}$ that w can be any value. In contrast, if Q_ψ sets $w_1 = 0$, then w can only be 0 or 1. In other words, w is bounded if and only if $w_1 = 0$.

The correctness of the reduction follows from Lemmas 3.6 and 3.7. More specifically, we show that the variable w is constant in every element query $Q_e(w)$ of $Q(w)$ if and only if ψ is not satisfiable. To see this, we need to inspect element queries of $Q(w)$. First, observe that for the sub-query $R_{01}(0) \wedge R_{01}(1) \wedge \bigwedge_{1 \leqslant i \leqslant m} R_{01}(x_i)$ to satisfy $R_{01}(\emptyset \to A, 2)$, every element query Q_e of Q must set each x_i either to 0 or 1. That is, every element query Q_e encodes a truth assignment μ_X of X. Similarly, by the access constraints on $R_\vee$, $R_\wedge$ and $R_\neg$ and the presence of Q_c, in every element query Q_e, Q_ψ correctly evaluates ψ for the truth assignment μ_X encoded in Q_e. Moreover, $R_o(I \to X, 2)$ cannot be used to put w in $\mathsf{cov}(Q_e, \mathcal{A})$, since the variable k cannot be in $\mathsf{cov}(Q_e, \mathcal{A})$ given the access constraints. However, in Q_e either $R_o(k, 1)$ and $R_o(k, w)$ co-occur (when $w_1 = 1$) or $R_o(k, 1)$ and $R_o(k, 0)$ co-occur (when $w_1 = 0$). In the latter case, w has become a constant variable; thus Lemma 3.6 applies and $Q_e(w)$ has bounded output. In the former case, w remains a non-constant variable that is not in $\mathsf{cov}(Q_e, \mathcal{A})$. Hence, when $w_1 = 1$ is in Q_e, Q_e is not bounded. Thus $Q_e(w)$ has bounded output if and only if the truth assignment μ_X encoded in Q_e makes ψ false. As a consequence, Q has bounded output if and only if ψ is not satisfiable.

Note that in the reduction above, $\mathcal{R}$ and $\mathcal{A}$ are fixed, *i.e.,* they do not depend on ψ.

Upper bound. We give an NP algorithm to check the complement of BOP for $\exists\mathsf{FO}^+$. From Lemma 3.7, we know that given a query $Q(\bar{x})$ in $\exists\mathsf{FO}^+$, to check whether $Q(\bar{x})$ does not have bounded output, we only need to guess an element query $Q_e(\bar{x}')$ of Q in which there is a variable x in $\bar{x}'$ that does not belong to $\mathsf{cov}(Q_e, \mathcal{A})$. Note that Q is equivalent to a UCQ $Q_\vee$, and an element query $Q_e(\bar{x}')$ of Q is an element query of a disjunct of $Q_\vee$. The NP algorithm thus (i) guesses disjunctions in $Q(\bar{x})$ to obtain a CQ query $Q'(\bar{x})$; and (ii) guesses a valuation ν of Q' to get a candidate element query $\nu(Q')$. It then checks whether $\nu(Q') \models \mathcal{A}$ and whether there exists a variable x such that $x \in \nu(\bar{x})$ but $x \notin \mathsf{cov}(\nu(Q'), \mathcal{A})$. It is easy to show that all element queries can be obtained in this way and that computing $\mathsf{cov}(\nu(Q'), \mathcal{A})$ is in PTIME. If the guesses pass this test then we have found a counterexample for Q to be of bounded output. Otherwise, we reject the guess. Hence, this algorithm decides whether Q has no bounded output and it is in NP. We can thus conclude that deciding BOP is in coNP for $\exists$FO$^+$.

(2) FO. We show that BOP is undecidable for FO queries by reduction from the complement of the satisfiability problem for FO queries, which is undecidable (cf. [Di Paola 1969]). The satisfiability problem for FO is to decide, given an FO query Q, whether there exists a database $\mathcal{D}$ such that $Q(\mathcal{D}) \neq \emptyset$.

Given an FO query Q_1, we construct a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, and an FO query Q such that Q_1 is not satisfiable if and only if Q has bounded output. More specifically, (1) the relational schema $\mathcal{R}$ contains all relation names used by Q_1, and one new unary relation schema $R(X)$; (2) $\mathcal{A} = \emptyset$; and (3) query Q is defined as $Q(x) = R(x) \wedge Q_1()$. Then Q_1 is not satisfiable if and only if there exists a constant N such that over instances $\mathcal{D}$ of $\mathcal{R}$, $|Q(\mathcal{D})| \leqslant N$. Indeed, since $R(x)$ is not bounded, $Q(x)$ is bounded only when $Q_1()$ returns empty, *i.e.,* when Q_1 is not satisfiable.

The undecidability remains intact when $\mathcal{R}$ and $\mathcal{A}$ are fixed. Indeed, the satisfiability problem for FO queries over a fixed relational schema is still undecidable. It is verified by reduction from the Post Correspondence Problem, and the reduction uses a database schema consisting of two fixed relation schemas (Proof of Theorem 6.3.1 in [Abiteboul et al. 1995]). Hence the proof for BOP(FO) remains valid under fixed $\mathcal{R}$ and $\mathcal{A} = \emptyset$. □

Using Lemma 3.2 and Theorem 3.4, we are ready to prove Theorem 3.1.

Proof of Theorem 3.1. We first study VBRP for CQ, UCQ and $\exists$FO$^+$, and then for FO.

(1) When $\mathcal{L}$ is CQ, UCQ, or $\exists$FO$^+$. It suffices to show that VBRP is Σ_3^p-hard for CQ, and that VBRP is in Σ_3^p for $\exists$FO$^+$.

Lower bound. We show that VBRP(CQ) is Σ_3^p-hard by reduction from the $\exists^*\forall^*\exists^*$3CNF problem, which is Σ_3^p-complete [Stockmeyer 1976]. The latter problem is to decide, given a sentence $\phi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, whether ϕ is true, where $X = \{x_1, \ldots, x_m\}$, $Y = \{y_1, \ldots, y_n\}$, $Z = \{z_1, \ldots, z_p\}$, and ψ is a 3SAT instance. Assume *w.l.o.g.* that $m \geqslant 2$.

Given an instance $\phi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, we define a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, a CQ query Q, a set $\mathcal{V}$ of CQ views, and a natural number M, such that Q has an M-bounded rewriting in CQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if ϕ is true.

(1) The relational schema $\mathcal{R}$ consists of the following relation schemas: (a) $R_{01}(A)$, $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$, and $R_\neg(A, \bar{A})$ are to encode the Boolean domain and operations, which we have seen in the proof of Theorem 3.4, with intended instances shown in Figure 2; (b) $R_Y(I_1, I_2, Y)$ is to store one truth-assignment of Y; (c) $R_o(I, Y)$ is to store a particular tuple, which the query plans can check only via fetch operations; and (d) $R_I(I, K)$ is to store the keys for the relation R_o.

(2) The access schema $\mathcal{A}$ consists of (a) four access constraints, similar to those used

in the proof of Theorem 3.4, to ensure that R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ encode Boolean domain and relations: $R_{01}(\emptyset \to A, 2), R_\vee(A_1 \to (A_2, B), 2), R_\wedge((A_1, A_2) \to B, 1)$, and $R_\neg(A \to \bar{A}, 1)$; (b) an access constraint $R_Y((I_1, I_2) \to Y, 1)$ to ensure that we only handle one truth-assignment of Y at a time; and (c) two access constraints $R_o(I \to Y, 1)$ and $R_I(I \to K, 1)$ for R_o and R_I, respectively, stating that I is a key for R_o and R_I.

It should be noted that the access constraints for $R_\vee$ and $R_\wedge$ are different. In $R_\vee$, we require that when the values corresponding to A_1 are bounded, the values corresponding to A_2 and B are bounded. While in $R_\wedge$, we require that only when both of the values corresponding to A_1 and A_2 are bounded, the values corresponding to B are bounded. As will be elaborated shortly, this subtle difference is important for our construction.

(3) The query Q in CQ is defined as follows:

$$Q() = \exists \bar{y}, k \left(Q_c() \wedge Q_Y(\bar{y}) \wedge (\bigwedge_{1 \leqslant j \leqslant n} R_Y(j, 1, y_j)) \wedge R_I(y_1, k) \wedge R_o(k, 1) \right).$$

Here Q_c is the same CQ as its counterpart given in the proof of Theorem 3.4, to ensure that the instances of R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$ contain all the tuples shown in Figure 2. Query $Q_Y(\bar{y})$ is defined as $\bigwedge_{1 \leqslant i \leqslant n} R_{01}(y_i)$. It is easy to see that for all $\mathcal{D} \models \mathcal{A}$, if $Q(\mathcal{D}) \neq \emptyset$, then the tuples in $\mathcal{D}$ corresponding to R_Y encode a valid truth-assignment of Y.

(4) The set $\mathcal{V}$ of CQ views consists of a single view V:

$$V(\bar{x}, k) = \exists w, \bar{x}', \bar{y}, \bar{z} \left(Q_c() \wedge Q_2(w, \bar{x}, \bar{x}') \wedge Q_3(w, \bar{y}, \bar{z}) \wedge Q_4(\bar{y}, w, k) \wedge Q_5(\bar{x}, w) \wedge Q_\psi(\bar{x}', \bar{y}, \bar{z}, 1) \right).$$

Intuitively, the view is defined in such a way that if a query plan ξ that uses V does not "fix" the values of $\bar{x}$, then ξ will not conform to $\mathcal{A}$, since the values that k can take will not be bounded. Here by fixing values we mean that V appears in the query plan in the form of $\sigma_{X=\bar{c}}(V)$, where X are the attributes corresponding to $\bar{x}$ and $\bar{c}$ is a constant tuple. Furthermore, we will see that $\bar{c}$ must consist of Boolean values for $\sigma_{X=\bar{c}}(V)$ to be of use for answering Q. In other words, $\bar{c}$ encodes a truth-assignment of X.

To construct V in this way, we separate the values of $\bar{x}$ from k by using a new copy $\bar{x}'$ of $\bar{x}$, which are used in the component queries of V. Moreover, we link the possible values for k to those of a variable y_1, and connect the possible values of y_1 to the values that variable w can take. The latter is shown to be unbounded when $\bar{x}$ is not fixed. Hence, when $\bar{x}$ is not fixed, k will be unbounded.

We next show how this is achieved by detailing each of the sub-queries in V.

(a) Query $Q_c()$ is the same as the one in Q (see the proof of Theorem 3.4 for details).

(b) We define $Q_2(w, \bar{x}, \bar{x}') = \bigwedge_{1 \leqslant i \leqslant m} R_\wedge(x_i', w, x_i)$. It is to ensure that if the values of $\bar{x}$ are Boolean, then $\bar{x}'$ and $\bar{x}$ take the same values. By inspecting instance $I_\wedge$ of $R_\wedge$ (shown in Figure 2), this only holds when $w = 1$. Indeed, if $w = 1$, by the access constraint on $R_\wedge$ and the presence of $Q_c()$ in V, we have that for any $\mathcal{D} \models \mathcal{A}$, if $Q_c(\mathcal{D}) \neq \emptyset$ then $\sigma_{A=1}(Q_2(\mathcal{D}))$ consists of tuples of the form $(1, \bar{x}, \bar{x})$, provided that $\bar{x}$ takes Boolean values. Here A denotes the first attribute in the result schema of Q_2. When either $w = 0$ or w and $\bar{x}$ do not take Boolean values, the access constraint on $R_\wedge$ only imposes a cardinality restriction, and the values in $\bar{x}'$ and $\bar{x}$ are not necessarily the same.

(c) We define $Q_3(w, \bar{y}, \bar{z}) = \exists \bar{y}', \bar{z}' \left(\bigwedge_{1 \leqslant k \leqslant n} R_\vee(y_k', w, y_k) \wedge \bigwedge_{1 \leqslant k \leqslant p} R_\vee(z_k', w, z_k) \right)$.

This query is to ensure that if $w = 0$ or $w = 1$ then the values of $\bar{y}$ and $\bar{z}$ must be Boolean values as well. As before this is due to the presence of $R_\vee(A_1 \to (A_2, B), 2)$ and $Q_c()$. In other words, for any $\mathcal{D} \models \mathcal{A}$ such that $Q_c(\mathcal{D}) \neq \emptyset$, $\sigma_{A=0/1}(Q_3(\mathcal{D}))$ consists of tuples of the form $(0/1, \bar{y}, \bar{z})$, $\bar{y}$ and $\bar{z}$ are tuples of Boolean values, and A denotes the first attribute in the result schema of Q_3. If w can take arbitrary values, however, then the values for $\bar{y}$ and $\bar{z}$ are not constrained.

(d) We define $Q_4(\bar{y}, w, k) = \left(\bigwedge_{1 \leqslant j \leqslant n} R_Y(j, w, y_j) \right) \wedge R_I(y_1, k)$.

This is to fetch the truth-assignment of Y and the value of k. Since the $\bar{y}$ values have to agree with their counterparts in Q_3, as argued before for Q_3, these values will be Boolean only when $w = 0$ or $w = 1$.Thus only in these cases $Q_4(\mathcal{D}) \neq \emptyset$ implies that a truth assignment of Y is embedded in $\mathcal{D}$.

(e) Query $Q_\psi(\bar{x}', \bar{y}, \bar{z}, 1)$ is to check whether ψ is true given the values $\bar{x}'$, $\bar{y}$, and $\bar{z}$. It makes use of R_{01}, $R_\vee$, $R_\wedge$ and $R_\neg$, and is expressed in CQ (see the proof of Theorem 3.4). It is only when $\bar{x}'$, $\bar{y}$ and $\bar{z}$ take Boolean values that this query correctly encodes ψ.

(f) The last query $Q_5(\bar{x}, w)$ is to ensure that if $V(\bar{x}, k)$ is used in a query plan for Q and conforms to $\mathcal{A}$, then it can only be used when all variables in $\bar{x}$ are assigned a constant Boolean value. Furthermore, when this is the case, w must be 1. As described above, this implies that $\bar{x}' = \bar{x}$, $\bar{y}$ and $\bar{z}$ take Boolean values, and Q_ψ correctly evaluates ψ. It is to encode this that we make use of the difference of the access constraints on $R_\vee$ and $R_\wedge$. Intuitively, the constraint on $R_\vee$ is used to check whether each variable in $\bar{x}$ takes a constant value, since it only takes the attribute A_1 as input. In contrast, since the access constraint on $R_\wedge$ takes both A_1 and A_2 as input, we use it to encode the conjunction of the results of checking each variable in $\bar{x}$. Query Q_5 encodes the tautology $\bigwedge_{1 \leqslant k \leqslant m} (x_k \vee x''_k \vee \neg x''_k)$. That is, $Q_5(\bar{x}, w) =$

$$\exists \bar{x}'', \bar{v}, \bar{v}', \bar{v}'', \bar{v}''' \Bigg(\bigwedge_{1 \leqslant k \leqslant m} (R_\vee(v_k, x_k, x''_k) \wedge R_\vee(v''_k, v_k, v'_k) \wedge R_\neg(x''_k, v'_k)) \wedge R_\wedge(v'''_2, v''_1, v''_2) \wedge (\bigwedge_{2 \leqslant k \leqslant m-2} R_\wedge(v'''_{k+1}, v'''_k, v''_{k+1})) \wedge R_\wedge(w, v'''_{m-1}, v''_m) \Bigg).$$

In particular, it encodes the truth value of the tautology in w. Hence, when all variables involved are Boolean, we necessarily have that $w = 1$. We argue next that when considering query plans for Q that involve $V(\bar{x}, k)$, we must call $Q_5(\bar{x}, w)$ with Boolean values for the variables in $\bar{x}$.

Indeed, first consider $\mathcal{D} \models \mathcal{A}$ such that $Q_c(\mathcal{D}) \neq \emptyset$ and consider $\sigma_{X=\mu_X}(Q_5(\mathcal{D}))$, where X consists of attributes corresponding to $\bar{x}$, and μ_X is a truth-assignment of X. In this case, the access constraint $R_\vee(A_1 \to (A_2, B), 2)$ ensures that all the values of $\bar{x}''$, $\bar{v}$, $\bar{v}'$, $\bar{v}''$, are Boolean. Similarly, $R_\wedge((A_1, A_2) \to B, 1)$ ensures that all values of $\mathsf{var}(v)'''$ are Boolean. Moreover, by $Q_c(\mathcal{D}) \neq \emptyset$, the Boolean operations are correctly encoded in $\mathcal{D}$. Hence, Q_5 correctly evaluates the tautology $\bigwedge_{1 \leqslant k \leqslant m} (x_k \vee x''_k \vee \neg x''_k)$ and assigns $w = 1$. In other words, when all $\bar{x}$ values are fixed Boolean values in Q_5, all previous queries in V work as desired as these required Boolean values for $\bar{x}$ and $w = 1$.

Suppose next that we still fix all $\bar{x}$ values, but not all of them take Boolean values. In this case, Q_5 requires the existence of certain tuples in the instances of $R_\vee$, $R_\wedge$ or $R_\neg$ that are not required by Q. That is, there exists $\mathcal{D} \models A$ for which $Q(\mathcal{D}) \neq \emptyset$ but $Q_5(\mathcal{D}) = \emptyset$ (and thus $V(\mathcal{D}) = \emptyset$). Clearly, using V in this way does not help us answer Q. Hence when all variables in $\bar{x}$ are fixed, we may assume that these values are Boolean.

It remains to rule out the case when some variables in $\bar{x}$ are not fixed. Suppose that we set all variables in $\bar{x}$ to a Boolean value, except for x_1. Let $X' = X \setminus \{x_1\}$ and consider an instance $\mathcal{D} \models \mathcal{A}$ and $\sigma_{X'=\mu_{X'}}(Q_5)(\mathcal{D})$ for some truth-assignment $\mu_{X'}$ of X'. Clearly, the query result contains tuples of the form $(a, \mu_{X'}, w)$ for constants a and w. Since a can be arbitrary, access constraint $R_\vee(A_1 \to (A_2, B), 2)$ only implies that at most two tuples s and t in $\mathcal{D}$ exist and are associated to $R_\vee$ such that $s[A_1] = t[A_1] = a$. However, it does not impose any restrictions on the other values in these two tuples. These values can thus be non-Boolean. Similarly, $R_\neg(A \to \bar{A}, 1)$ does not impose value restrictions (except for a cardinality constraint) when $R_\neg(x_1'', v_1')$ can bind x_1'' and v_1' with arbitrary values. The same holds for $R_\wedge((A_1, A_2) \to B, 1)$ and $R_\wedge(v_2''', v_1'', v_2'')$. Although v_2'' takes only Boolean values (recall that we fixed x_2 to a Boolean value), v_1'' can be arbitrary and so can be v_2'''. A similar argument shows that all v_i''' can be arbitrary and so can be w. It should be noted that w can take an arbitrary value for any possible binding of x_1 to the underlying database. Hence, $\sigma_{X'=\mu_{X'}}(Q_5)$ does not have bounded output.

For example, for $\bar{x} = (x_1, x_2)$, $Q_5(\bar{x}, w) = \exists x_1'', x_2'', v_1, v_2, v_1', v_2', v_1'', v_2''\, R_\vee(v_1, x_1, x_1'') \wedge R_\vee(v_1'', v_1, v_1') \wedge R_\neg(x_1'', v_1') \wedge R_\vee(v_2, x_2, x_2'') \wedge R_\vee(v_2'', v_2, v_2') \wedge R_\neg(x_2'', v_2') \wedge R_\wedge(v_2''', v_1'', v_2'') \wedge R_\wedge(w, v_1''', v_2'')$. When $x_1 = 1$ and x_2 is not fixed, we can verify that w is unbounded as follows. We insert the following tuples into the instance $\mathcal{D}$ of $\mathcal{R}$: we add tuples $(a_1, a_1, a_1), \ldots, (a_n, a_n, a_n)$ to $I_\vee$, $(a_1, a_1), \ldots, (a_n, a_n)$ to $I_\neg$, and $(a_1, 1, a_1), \ldots, (a_n, 1, a_n)$ to $I_\wedge$. Note that we still have that $\mathcal{D} \models \mathcal{A}$ and moreover, $\{(1, a_1, a_1), \ldots, (1, a_n, a_n)\} \subseteq Q_5(\mathcal{D})$. Hence the possible values of w are unbounded. Along the same lines, one can see that $\sigma_{X'=\mu_{X'}}(V)$ does not have bounded output either and hence, cannot be used in a query plan that conforms to $\mathcal{A}$. Indeed, this readily follows from Q_4, which now can bind y_1 with arbitrary values since $R_Y(1, w, y_1)$ can be mapped to various tuples with distinct w-values; and similarly $R_I(y_1, k)$ can be mapped to various tuples, resulting in an unbounded number of k values.

In summary, Q_5 ensures that whenever V appears in a query plan that conforms to $\mathcal{A}$, it must have all of its $\bar{x}$ values fixed to some Boolean values.

(5) We set $M = 6$, *i.e.,* we only allow query plan trees with at most six nodes.

To show the correctness of the reduction, we first argue that if Q has an M-bounded rewriting using V under $\mathcal{A}$, then this rewriting can only be of a very specific form. Indeed, since $Q(\mathcal{D})$ depends on the instance $\mathcal{D}$ (*i.e.,* for some $\mathcal{D}$, $Q(\mathcal{D}) = \emptyset$, while for others $Q(\mathcal{D}) \neq \emptyset$), the query plan ξ cannot be one of the two trivial plans that always return $\emptyset$ or $()$. Suppose that the query plan does not use V, then the query plan can only access the database via fetch operations. However, since Q uses all 7 relation atoms in $\mathcal{R}$, the query plan must contain at least 7 fetch operations, which exceed the bound M. Therefore, the query plan has to use V. Furthermore, since V does not contain R_o, whereas $Q(\mathcal{D})$ depends on the tuples in $\mathcal{D}$ corresponding to R_o, the plan ξ needs to fetch data from R_o. Consider such a fetch operation fetch$(I \in S_j, R_o, Y)$. We distinguish between the following two cases: (i) S_j is equal to a constant c; or (ii) S_j is the result of some more complex query plan. Note that case (i) is not helpful for answering Q as the value k used in the atom $R_o(k, 1)$ in Q is arbitrary and may thus be distinct from the constant c. We can thus assume that we are in case (ii). Moreover, the atom $R_o(k, 1)$ in Q asks for a tuple with its second attribute to be set to 1. This requirement needs to be encoded in plan ξ as well, *e.g.,* by means of a constant selection condition $\sigma_{Y=1}$. This selection must occur after the fetch operation. Observe also that since Q is Boolean, whereas the fetch operation, the constant selection, and V are not, ξ must contain a projection of the form $\pi_\emptyset$. This projection clearly must come after the selection operation in ξ. From this we know that fetch$(I \in S_j, R_o, Y)$ has at least one selection and projection as ancestor in the query plan tree.

We next analyze the query plan ξ_j for S_j. Consider two options: (a) S_j takes V as a descendant in the query plan tree; and (b) S_j does not have V as a descendant.

In case (a) the plan ξ_j for S_j must contain a projection π_A so that S_j is unary. Indeed, recall that R_o is binary and the access constraint takes the first attribute of R_o as input, while V is not unary. Moreover, as argued above, the only way that V can be used in ξ_j that conforms to $\mathcal{A}$ is when it occurs as $\sigma_{X=\mu_X^0}(V)$, *i.e.,* all its $\bar{x}$-values are fixed Boolean values by means of a truth-assignment μ_X^0 of X. This selection condition needs to be accounted for in ξ_j. Note also that this constant selection should not be expanded to include the last attribute in V. Indeed, this would make S_i equal to a constant (case (i) above), which is not helpful in answering Q. From this we know that $\mathsf{fetch}(I \in S_j, R_o, Y)$ has at least V, a selection and a projection as descendants. Put together with our earlier observation, these account for the six possible nodes in ξ_j. In fact, this completely fixes possible query plans. Indeed, the plan ξ_j must be of the form $S_1 = \pi_\emptyset(S_2)$; $S_2 = \sigma_{Y=1}(S_3)$; $S_3 = \mathsf{fetch}(I \in S_4, R_o, Y)$; $S_4 = \pi_A(S_5)$; $S_5 = \sigma_{X=\mu_X^0}(S_6)$ and $S_6 = V$, for some truth-assignment μ_X^0 of X. Furthermore, as argued above, S_4 should not just be a constant value, and the projection π_A should be imposed on the last attribute of V (the other ones are fixed by means of the selection condition in S_5).

In case (b), observe that the overall query plan must use V. Here this implies that V must occur in a subtree of the query plan different from the subtree rooted at $\mathsf{fetch}(I \in S_j, R_o, Y)$. At least one node is required to glue these subtrees together. For the query plan ξ_j for S_j, since S_j is not equal to a constant, we still need to distinguish the following two cases: (b1) S_j is $\mathsf{fetch}(\emptyset, R_{01}, A)$, *i.e.,* the only possible query plan of size 1 that does not use V; (b2) the size of the query plan ξ_j for S_j is at least 2. For case (b1), similar to case (i) above, we can show that it is not helpful for answering Q. Then we only need to consider case (b2). However, we have at least two nodes in the query plan tree for S_j, one for V, and at least one to glue the subtrees together (as argued above), accounting for four nodes. Combined with the (minimal) three nodes needed for $\mathsf{fetch}(I \in S_j, R_o, Y)$ and its ancestors, this results in a query plan of at least seven nodes, exceeding the bound $M = 6$. Hence, case (b2) cannot occur.

As a consequence, the only possible query plans are of the form as given in case (a).

We can thus conclude that if Q has a 6-bounded query plan ξ in CQ using V under $\mathcal{A}$, then ξ is $\mathcal{A}$-equivalent to $Q'_{\mu_X^0} = \pi_\emptyset\Big(\sigma_{\bar{x}=\mu_X^0}(V(\bar{x},k)) \bowtie R_o(k,1)\Big)$ for some truth-assignment μ_X^0 of X. We next show that $Q \equiv_\mathcal{A} Q'_{\mu_X^0}$ for some μ_X^0 if and only if ϕ is true. For convenience, we express $Q'_{\mu_X^0}$ as CQ $Q'_{\mu_X^0} = \exists k\,(V(\mu_X^0,k) \wedge R_o(k,1))$.

($\Leftarrow$) Suppose that ϕ is true and let μ_X^0 be a truth-assignment of X such that $\forall Y \exists Z \psi(\mu_X^0, Y, Z) = \mathsf{true}$. Consider $Q'_{\mu_X^0} = \exists k\,(V(\mu_X^0,k) \wedge R_o(k,1))$ and its unfolding

$$\exists k\Bigg(\exists w,\bar{x}',\bar{y},\bar{z}\Bigg(Q_c() \wedge \bigwedge_{1\leqslant k\leqslant m} R_\wedge(x'_i,w,\mu_X^0(x_i)) \wedge \exists \bar{y}',\bar{z}'\Bigg(\bigwedge_{1\leqslant k\leqslant n} R_\vee(y'_k,w,y_k) \wedge \bigwedge_{1\leqslant k\leqslant p} R_\vee(z'_k,w,z_k)\Bigg) \wedge \Bigg(\bigwedge_{1\leqslant j\leqslant n} R_Y(j,w,y_j)\Bigg) \wedge R_I(y_1,k) \wedge Q_5(\mu_X^0,w) \wedge Q_\psi(\bar{x}',\bar{y},\bar{z},1)\Bigg) \wedge R_o(k,1)\Bigg).$$

Since μ_X^0 is a truth-assignment of X, $Q_5(\mu_X^0,w)$ will assign $w=1$. As a consequence $\bar{x}' = \mu_X^0$, $\bar{y}$ and $\bar{z}$ take Boolean values, and the unfolding of $Q'_{\mu_X^0}$ is $\mathcal{A}$-equivalent to

$$\exists k\Bigg(\bar{y},\bar{z}\Bigg(Q_c() \wedge Q_Y(\bar{y}) \wedge Q_Z(\bar{z}) \wedge \Bigg(\bigwedge_{1\leqslant j\leqslant n} R_Y(j,1,y_j)\Bigg) \wedge R_I(y_1,k) \wedge Q_\psi(\mu_X^0,\bar{y},\bar{z},1)\Bigg) \wedge R_o(k,1)\Bigg), \quad (\dagger)$$

where $Q_Y(\bar{y})$ and $Q_Z(\bar{z})$ encode that $\bar{y}$ and $\bar{z}$ take Boolean values, just as in Q.

Consider an instance $\mathcal{D} \models \mathcal{A}$ such that $Q(\mathcal{D}) \neq \emptyset$. As remarked earlier, this implies that the tuples in $\mathcal{D}$ corresponding to R_Y encode a truth assignment μ_Y of Y. Moreover, tuples $(\mu_Y(y_1), k)$ and $(k, 1)$ are present in $\mathcal{D}$ (for relations R_I and R_o, respectively). Hence, if $Q(\mathcal{D}) \neq \emptyset$ then $Q'_{\mu^0_X}(\mathcal{D}) \neq \emptyset$ if and only if $\exists \bar{z}\, Q_\psi(\mu^0_X, \mu_Y, \bar{z}, 1)$ evaluates to true. Since $\forall Y \exists Z \psi(\mu^0_X, Y, Z)$ is true, we know that $\exists Z \psi(\mu^0_X, \mu_Y, Z)$ is true. Hence, $Q(\mathcal{D}) \neq \emptyset$ implies that $Q'_{\mu^0_X}(\mathcal{D}) \neq \emptyset$. In other words, $Q \sqsubseteq_{\mathcal{A}} Q_{\mu^0_X}$. For the converse, $Q_{\mu^0_X} \sqsubseteq_{\mathcal{A}} Q$, note that if $Q(\mathcal{D}) = \emptyset$, then so is $Q'_{\mu^0_X}(\mathcal{D})$. Indeed, the query shown in (†) is just like Q but with some additional restrictions ($Q_Z(\bar{z})$ and $Q_\psi(\mu^0_X, \bar{y}, \bar{z}, 1)$). Hence, we can conclude that $Q \equiv_{\mathcal{A}} Q'_{\mu^0_X}$, and thus Q has a 6-bounded query rewriting using V under $\mathcal{A}$.

($\Rightarrow$) Suppose that ϕ is false, but by contradiction Q has a 6-bounded rewriting ξ using V under $\mathcal{A}$. As argued above, $\xi \equiv_{\mathcal{A}} Q'_{\mu^0_X}$ for some truth-assignment μ^0_X of X. Since ϕ is false, there must exist a truth-assignment μ^0_Y of Y such that $\exists Z \psi(\mu^0_X, \mu^0_Y, Z) =$ false. Let $\mathcal{D}$ be an instance of $\mathcal{R}$ such that $\mathcal{D} \models \mathcal{A}$, $Q(\mathcal{D}) \neq \emptyset$, and the tuples in $\mathcal{D}$ corresponding to R_Y encode μ^0_Y. By $\exists Z \psi(\mu^0_X, \mu^0_Y, Z) =$ false, $Q_\psi(\mu^0_X, \mu^0_Y, \bar{z}, 1)(\mathcal{D}) = \emptyset$. Then $Q'_{\mu^0_X}(\mathcal{D}) = \emptyset$, and hence, $Q \not\equiv_A Q'_{\mu^0_X}$. Since this argument works for any truth-assignment μ_X of X, Q is not $\mathcal{A}$-equivalent to any Q'_{μ_X} for μ_X of X. As these are the only possible 6-bounded rewritings, Q does not have a 6-bounded rewriting using V under $\mathcal{A}$.

Upper bound. We next provide an Σ^p_3 algorithm for VBRP(∃FO$^+$), as follows:

(1) guess a query plan ξ such that $|\xi| \leqslant M$;
(2) check whether ξ conforms to $\mathcal{A}$; if not, then reject the guess; otherwise continue;
(3) rewrite ξ into a query Q' in ∃FO$^+$by substituting the view definition for each view used in ξ;
(4) check whether $Q' \equiv_{\mathcal{A}} Q$. If so, then return true; otherwise, return reject the guess.

It is easy to see the correctness of the algorithm. For its complexity, we will show that step (2) can be done in P^{NP}. Moreover, step (3) can be done in PTIME since ξ is a tree, and $|Q'|$ is bounded by $O(|\xi| \cdot |\mathcal{V}|)$. Step (4) requires checking whether $Q' \equiv_{\mathcal{A}} Q$. This was shown to be in Π^p_2 (Lemma 3.2). Putting these together, the algorithm is in Σ^p_3.

It should be remarked that the non-deterministic algorithm given above just aims to prove the upper bound of VBRP(∃FO$^+$). More practical algorithms for bounded rewriting using views can be developed along the same lines as the bounded plan generation algorithm of [Cao and Fan 2016], possibly collaborating with a DBMS optimizer.

We next show that step (2) can be done in P^{NP}.

LEMMA 3.8. *Given a query plan ξ, it is in P^{NP} to decide whether ξ conforms to $\mathcal{A}$.* □

Proof: To check whether ξ conforms to $\mathcal{A}$, it suffices to verify that for each fetch($X \in S_j, R, Y$) operation in ξ, the following conditions hold: (a) there exists an access constraint $R(X \to Y', N)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$; and (b) there exists a constant N_1 such that for all instances $\mathcal{D}$ of $\mathcal{R}$ that satisfy $\mathcal{A}$, $|S_j| \leqslant N_1$ in the computation of $\xi(\mathcal{D})$.

For each fetch($X \in S_j, R, Y$) operation, it is in PTIME to check condition (a). We use the following algorithm to check condition (b). Let ξ' be the sub-tree of ξ rooted at S_j,

(1) express ξ' as an equivalent query Q_j in ∃FO$^+$;
(2) unfold Q_j by replacing each view with its definition, yielding Q'_j in ∃FO$^+$;
(3) check whether Q'_j has bounded output; if so, return true; otherwise, return false.

The correctness of the algorithm is immediate. For its complexity, observe that steps (1) and (2) are in PTIME, and step (3) is in coNP by Theorem 3.4. Since there are at most $O(|\xi|)$ fetch operations in ξ, the algorithm is in P^{NP}. □

(2) When $\mathcal{L}$ is FO. We show that VBRP is undecidable for FO queries by reduction from the complement of the satisfiability problem for FO queries, just like BOP for FO (see the proof of Theorem 3.4 for the satisfiability problem).

Given an FO query Q_1, we construct a relational schema $\mathcal{R}$, an access schema $\mathcal{A}$, an FO query Q, a set $\mathcal{V}$ of FO views, and a natural number M, such that Q has an M-bounded rewriting in FO using $\mathcal{V}$ under $\mathcal{A}$ if and only if there exists no database $\mathcal{D}$ such that $Q_1(\mathcal{D}) \neq \emptyset$. More specifically, (1) $\mathcal{R}$ contains all relation names used by Q_1, and one new unary relation schema $R(X)$; (2) $\mathcal{A} = \emptyset$; (3) $Q(x) = R(x) \wedge Q_1()$; these are the same as their counterparts in the proof of Theorem 3.4; (4) $\mathcal{V} = \emptyset$; and (5) $M = 1$.

Since $\mathcal{V} = \emptyset$, $\mathcal{A} = \emptyset$, and $M = 1$, the only possible 1-bounded rewritings of Q are the constant query $Q_\emptyset$, which returns $\emptyset$ on all databases, or Q_c for some constant c, which returns $\{(c)\}$ on all databases. If the query plan is Q_c, then for all instances $\mathcal{D}$ of $\mathcal{R}$, we have that $Q_c(\mathcal{D}) = \{(c)\}$. Then we can construct a database $\mathcal{D}_1$ such that the instance of relation schema R does not contain the constant c. However, by the definition of Q we know that $(c) \notin Q(\mathcal{D}_1)$, which is a contradiction. Hence the only possible 1-bounded rewriting of Q is $Q_\emptyset$. It is easy to verify that $Q(x) \equiv_{\mathcal{A}} Q_\emptyset$ if and only if $Q(x) \equiv Q_\emptyset$ if and only if for any instance $\mathcal{D}$ of $\mathcal{R}$, $Q_1(\mathcal{D}) = \emptyset$, *i.e.,* when Q_1 is not satisfiable. □

3.2. The Impact of Various Parameters

One might think that fixing some parameters of VBRP would simplify the analysis. As will be seen in Section 4, in practice we often have predefined database schema $\mathcal{R}$, access schema $\mathcal{A}$, bound M and views $\mathcal{V}$, while queries and instances of $\mathcal{R}$ vary.

Unfortunately, fixing $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ does not simplify the analysis of VBRP for FO.

COROLLARY 3.9. *There exist fixed $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ such that it is undecidable to decide, whether an* FO *query Q has an M-bounded rewriting in* FO *using $\mathcal{V}$ under $\mathcal{A}$.* □

Proof: Recall that VBRP(FO) is shown undecidable by reduction from the complement of the satisfiability problem for FO (see the proof of Theorem 3.1), using fixed $\mathcal{V} = \emptyset$, $\mathcal{A} = \emptyset$ and $M = 1$. As argued in the proof of Theorem 3.4, the reduction remains valid when $\mathcal{R}$ is also fixed. From this Corollary 3.9 follows. □

We now study the impact of parameters on VBRP for CQ, UCQ and $\exists$FO$^+$. Our main conclusion is that fixing $\mathcal{R}$, $\mathcal{A}$ and M does not simplify the analysis of VBRP. When the set $\mathcal{V}$ of views is also fixed, VBRP is simpler for these positive queries, to an extent.

Fixing $\mathcal{R}$, $\mathcal{A}$ and M. Fixing database schema, access schema and plan size does not help us. Indeed, the Σ_3^p lower bound for CQ is verified by using fixed $\mathcal{R}$, $\mathcal{A}$ and M (Theorem 3.1). From this the corollary below follows.

COROLLARY 3.10. *There exist fixed $\mathcal{R}$, $\mathcal{A}$ and M such that it is Σ_3^p-complete to decide, given a query Q in $\mathcal{L}$ and a set $\mathcal{V}$ of $\mathcal{L}$-definable views over $\mathcal{R}$, whether Q has an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under $\mathcal{A}$ when $\mathcal{L}$ is one of* CQ, UCQ *and* $\exists$FO$^+$. □

Fixing $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$. Suppose that besides $\mathcal{R}$, $\mathcal{A}$ and M, the set $\mathcal{V}$ of views is also fixed. This puts VBRP in C^p_{2k+1} for CQ, UCQ and $\exists\mathsf{FO}^+$, where C^p_{2k+1} is the complexity class defined as $\mathsf{coNP}\vee\bigvee_{i=1}^{k}(\mathsf{NP}\wedge\mathsf{coNP})$ [Wagner 1987]. Here $\mathsf{NP}\wedge\mathsf{coNP}$ is also known as D^p, where a languages L' is in D^p if and only if there exist two languages $L'_1 \in \mathsf{NP}$ and $L'_2 \in \mathsf{coNP}$ such that $L' = L'_1 \cap L'_2$. A language L' is in $\mathsf{C}_1 \vee \mathsf{C}_2$ for complexity classes C_1 and C_2 if there exist two languages $L'_1 \in \mathsf{C}_1$ and $L'_2 \in \mathsf{C}_2$ such that $L' = L'_1 \cup L'_2$. Hence, C^p_{2k+1} consists of languages that can be written as the union of k D^P languages and a coNP language. It resides in the Boolean NP-hierarchy and is contained in $\Delta^p_2 = P^{\mathsf{NP}}$.

Note that the membership of VBRP in C^p_{2k+1}, when $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ are fixed, provides an interesting insight. It reveals how NP and coNP oracles can be combined to decide VBRP. By contrast, if only a Δ^p_2 upper bound had been provided, one could only get that polynomially many calls to NP oracles suffice to decide VBRP.

THEOREM 3.11. *For each natural number k, there exist fixed $\mathcal{R}$, $\mathcal{A}$, M, $\mathcal{V}$ such that it is C^p_{2k+1}-complete to decide, given a query Q in $\mathcal{L}$ over $\mathcal{R}$, whether Q has an M-bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ under $\mathcal{A}$, when $\mathcal{L}$ is* CQ, UCQ *or* $\exists\mathsf{FO}^+$. □

To show Theorem 3.11 we need some notations, which will also be used in Section 4.

(a) For a query Q, denote by QP_Q the set of all candidate query plans using $\mathcal{V}$ that are no larger than M (see Section 2).

(b) For $\xi \in \mathsf{QP}_Q$, we write $\xi \sqsubseteq_\mathcal{A} Q$ if $Q_\xi \sqsubseteq_\mathcal{A} Q$, where Q_ξ denotes the query expressed by ξ (see Section 2); similarly we write $Q \sqsubseteq_\mathcal{A} \xi$ if $Q \sqsubseteq_\mathcal{A} Q_\xi$, and $\xi \sqsubseteq_\mathcal{A} \xi'$ for $\xi' \in \mathsf{QP}_Q$ if $Q_\xi \sqsubseteq_\mathcal{A} Q_{\xi'}$. We write $\xi \equiv_\mathcal{A} \xi'$ if $\xi \sqsubseteq_\mathcal{A} \xi'$ and $\xi' \sqsubseteq_\mathcal{A} \xi$, and $\xi \sqsubset_\mathcal{A} \xi'$ if $\xi \sqsubseteq_\mathcal{A} \xi'$ but $\xi \not\equiv_\mathcal{A} \xi'$.

Proof: We show that VBRP is C^p_{2k+1}-hard for CQ and in C^p_{2k+1} for $\exists\mathsf{FO}^+$in this setting.

Lower bound. The lower bound proof is based on a characterization of C^p_{2k+1} given in [Wagner 1987], stated as follows: a language L is in C^p_{2k+1} if and only if there exist $2k+1$ languages $L_0, L_1, \ldots, L_{2k}$, each of which is in NP, such that $L_0 \supseteq L_1 \supseteq L_2 \supseteq \cdots \supseteq L_{2k}$ and $L = \bar{L}_0 \cup \bigcup_{i=1}^{k}(L_{2i-1} \cap \bar{L}_{2i})$. We show that every such language can be reduced to an instance of VBRP(CQ), establishing hereby its C^p_{2k+1}-hardness.

To start the reduction, take any L in C^p_{2k+1} and write it as $\bar{L}_0 \cup \bigcup_{i=1}^{k}(L_{2i-1} \cap \bar{L}_{2i})$. Since for each $i \in [0, 2k]$, L_i is in NP, we have reductions f_i from L_i to 3SAT. In other words, for each string $\bar{\sigma} \in \Sigma^*$, $\bar{\sigma} \in L_i$ if and only if $f_i(\bar{\sigma})$ is a satisfiable 3SAT instance. Note that $L_i \supseteq L_{i+1}$ implies that whenever $f_{i+1}(\bar{\sigma})$ is satisfiable, then so is $f_i(\bar{\sigma})$. We use this in the proof below to ensure that only $k+1$ possible query plans need to be considered. Following [Wagner 1987], it can be verified that $\bar{\sigma} \in L$ if and only if

$$|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}| \text{ is even.}$$

By $L_0 \supseteq L_1 \supseteq \cdots \supseteq L_{2k}$, $|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}|$ is an even number only either when $f_0(\bar{\sigma})$ is unsatisfiable, or when the largest index that corresponds to a satisfiable 3SAT instance is of the form $f_{2\ell-1}(\bar{\sigma})$ for some ℓ. In the latter case, all 3SAT instances corresponding to $f_i(\bar{\sigma})$, for $0 \leqslant i \leqslant 2\ell-1$, are satisfiable, yielding an even number (2ℓ) of satisfiable instances. Conversely, if $f_{2\ell}(\bar{\sigma})$ is satisfiable then so are all $f_i(\bar{\sigma})$ for $0 \leqslant i \leqslant 2\ell$, yielding an odd number ($2\ell+1$) of satisfiable instances. One can see that $\bar{\sigma} \notin L_0$ iff $|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}|$ is zero; and $\bar{\sigma} \in L_{2\ell-1} \cap \bar{L}_{2\ell}$ iff $|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}|$ is equal to 2ℓ. Thus deciding whether $\bar{\sigma} \in L$ reduces to checking whether $|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}|$ is even, and vice versa.

We next show that deciding whether "$|\{i \mid f_i(\bar{\sigma})$ is satisfiable, $i \in [0, 2k]\}|$ is even" can be reduced to checking whether a CQ query Q has an 1-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. Given $2k+1$ 3SAT instances $\Theta = \{f_i(\bar{\sigma}) \mid i \in [0, 2k]\}$, we define a CQ query Q_Θ that depends on the 3SAT instances, a fixed database schema $\mathcal{R}$, $M = 1$, k views $\mathcal{V} = \{V_1, \ldots, V_k\}$, each of which is fixed, and a fixed access schema $\mathcal{A}$ such that Q_Θ has an 1-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$ if and only if $|\{i \mid f_i(\bar{\sigma})$ is satisfiable, $i \in [0, 2k]\}|$ is even. We assume *w.l.o.g.* that the 3SAT instances have the same number of variables, n, and that each instance has a disjoint set of variables. Let $X_i = \{x_1^i, \ldots, x_n^i\}$ be the set of variables used by the 3SAT instance $f_i(\bar{\sigma})$, for $i \in [0, 2k]$.

(1) The database schema $\mathcal{R}$ consists of $R_{01}(B)$, $R_\vee(B, A_1, A_2)$ $R_\wedge(B, A_1, A_2)$, $R_\neg(A, \bar{A})$, and $R_s(V_0, \ldots, V_{2k}, U)$. The first four relations are to encode Boolean operations and domain with intended instances shown in Figure 2. The last relation is to hold instances indicating which 3SAT instances are satisfiable, as will become clear shortly.

(2) We next define the CQ query Q_Θ. We first encode all 3SAT instances in Θ:

$$Q_\Theta^{\mathsf{3SAT}}(\bar{v}) = \exists \bar{x}_0, \bar{x}_1 \ldots, \bar{x}_{2k} \big(\bigwedge_{i=0}^{2k} Q_{f_i(\bar{\sigma})}(\bar{x}_i, v_i) \big),$$

where $\bar{x}_i = (x_1^i, \ldots, x_n^i)$, $\bar{v} = (v_0, v_1, \ldots, v_{2k})$ and $Q_{f_i(\bar{\sigma})}$ encodes $f_i(\bar{\sigma})$ by leveraging conjunction, disjunction, negation and Boolean domain encoded by instances of $R_\wedge$, $R_\vee$, $R_\neg$ and R_{01}, respectively. Given a truth-assignment μ_{X_i} of X_i, $Q_{f_i}(\mu_{X_i}, v_i)$ sets $v_i = 0$ if μ_{X_i} is not a witness of the satisfiability of $f_i(\bar{\sigma})$, and sets $v_i = 1$ otherwise.

To ensure that the Boolean operations and domain are properly encoded by instances of $R_\wedge$, $R_\vee$, $R_\neg$ and R_{01}, we consider Q_c, the same CQ as its counterpart given in the proof of Theorem 3.4. In addition, we define a Boolean query Q_s which demands the existence of the following $\frac{(2k+1)(2k+2)}{2}$ atoms:

$R_s(1, 0, 0, \ldots, 0, i)$	for $i = 0$	($f_0(\bar{\sigma})$ is satisfiable)
$R_s(1, 1, 0, \ldots, 0, i)$	for $i = 0, 1$	($f_1(\bar{\sigma})$ and $f_0(\bar{\sigma})$ are satisfiable)
$R_s(1, 1, 1, \ldots, 0, i)$	for $i = 0, 1, 2$	($f_2(\bar{\sigma})$, $f_1(\bar{\sigma})$ and $f_0(\bar{\sigma})$ are satisfiable)
$\vdots$	$\vdots$	$\vdots$
$R_s(1, 1, 1, \ldots, 1, i)$	for $i = 0, 1, \ldots, 2k$	(all instances in Θ are satisfiable)

The semantics of these atoms is as follows. A constant 1 (resp. 0) in attribute V_i of R_s, for $i \in [0, 2k]$, indicates that $f_i(\bar{\sigma})$ is satisfiable (resp. unsatisfiable), and the last attribute indicates the corresponding indices of instances in Θ that are satisfiable. Finally, we define

$$Q_\Theta(u) = \exists \bar{v} \left(Q_\Theta^{\mathsf{3SAT}}(\bar{v}) \wedge R_s(\bar{v}, u) \wedge Q_c \wedge Q_s \right).$$

(3) The access schema $\mathcal{A}$ consists of one constraint on each relation such that the instances of $R_\wedge$, $R_\vee$, $R_\neg$, R_{01} and R_s contain the number of tuples required by Q_c and Q_s, respectively (see, *e.g.,* the counterpart for R_{01} in the proof of Theorem 3.4).

As a consequence, for any instance $\mathcal{D} \models \mathcal{A}$ we can distinguish the following three cases: (i) $Q_\Theta(\mathcal{D}) = \emptyset$ because $\mathcal{D} \not\models Q_c \wedge Q_s$; (ii) $Q_\Theta(\mathcal{D}) = \emptyset$ but $\mathcal{D} \models Q_c \wedge Q_s$; or (iii) $Q_\Theta(\mathcal{D}) \neq \emptyset$ and $\mathcal{D} \models Q_c \wedge Q_s$. Note that in cases (ii) and (iii), $\mathcal{D} = (I_\wedge, I_\vee, I_\neg, I_{01}, I_s)$, where $I_\wedge$, $I_\vee$, $I_\neg$, I_{01} are as shown in Figure 2, and I_s consists of the $\frac{(2k+1)(2k+2)}{2}$ tuples enumerated in Q_s. Moreover, in case (ii) we have that $Q_\Theta(\mathcal{D}) = \emptyset$ if and only if none of the 3SAT instances in Θ is satisfiable. In case (iii), $Q_\Theta(\mathcal{D}) = \{0, 1, \ldots, \ell\}$, where ℓ denotes the largest index taken from $[0, 2k]$ corresponding to a satisfiable instance $f_\ell(\bar{\sigma})$ in Θ.

(4) Finally, $\mathcal{V}$ consists of the following k views. For $i \in [1, k]$ we define

$$V_i(u) = R_s(\underbrace{1, \ldots, 1}_{2i \text{ times}}, 0, \ldots, 0, u) \wedge Q_c \wedge Q_s.$$

In other words, for $\mathcal{D} \models \mathcal{A}$, either $V_i(\mathcal{D})$ is empty or $V_i(\mathcal{D}) = \{0, 1, \ldots, 2i-1\}$. As a consequence, whenever $V_i(\mathcal{D})$ is non-empty, $Q_\Theta(\mathcal{D}) \equiv_\mathcal{A} V_i(\mathcal{D})$ if and only if $\ell = 2i-1$ is the largest index for which $f_\ell(\bar{\sigma})$ is satisfiable. Note that apart from $Q_\emptyset$, no other 1-bounded rewriting for Q_Θ exists that does not use views. Indeed, the only other possible such 1-bounded rewriting is of the form Q_c for some constant c, which returns $\{(c)\}$ on all databases. However, when $\mathcal{D} \not\models Q_c \wedge Q_s$, $Q_\Theta(\mathcal{D})$ is empty and hence $Q_\Theta(\mathcal{D}) \neq Q_c(\mathcal{D})$. Therefore, the only possible 1-bounded rewriting for Q_Θ is $Q_\emptyset$, V_1, ..., or V_k.

For the correctness of the reduction, observe that $\bar{\sigma} \in L$ if and only if Q_Θ has an 1-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, where $\Theta = \{f_i(\bar{\sigma}) \mid i \in [0, 2k]\}$. Indeed, $\bar{\sigma} \in L$ if and only if $|\{i \mid f_i(\bar{\sigma}) \text{ is satisfiable}, i \in [0, 2k]\}|$ is even if and only if for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}$ then either (a) $Q_\Theta(\mathcal{D}) = \emptyset$ or (b) $Q_\Theta(\mathcal{D}) = \{0, 1, \ldots, 2i-1\}$ for some $i \in [1, k]$ if and only if either (a) $Q_\Theta \equiv_\mathcal{A} Q_\emptyset$ (empty query) or (b) $Q_\Theta \equiv_\mathcal{A} V_i$.

Upper bound. Let Q be an $\exists$FO$^+$query and consider fixed $\mathcal{R}$, $\mathcal{V}$, $\mathcal{A}$ and M. Observe that there are only a constant number of possible query plans for Q with size bounded by M. Furthermore, for each constant-size plan ξ, it is in PTIME to check whether ξ conforms to $\mathcal{A}$. Indeed, from the proof of Lemma 3.8 we know that this is in PTIME as long as ξ has bounded output. By Lemma 3.7, when ξ has a constant size, checking bounded output of ξ is in PTIME since there are a constant number of element queries of ξ and checking the condition on covered variables (as stated in Lemma 3.7) is in PTIME.

Denote by QP_Q the set of candidate query plans of length at most M. Remove from QP_Q all plans that do not conform to $\mathcal{A}$, and denote the set of remaining plans also as QP_Q; as argued above, this can be done in PTIME. Note that the empty query plan $\xi_\emptyset$ is in QP_Q. Hence, Q has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$ if and only if either (a) Q is not satisfiable, in which case $Q \equiv_\mathcal{A} \xi_\emptyset$, or (b) Q is satisfiable and $Q \equiv_\mathcal{A} \xi$ for some non-empty $\xi \in \mathsf{QP}_Q$. We next show that case (a) can be decided in coNP and (b) deciding $Q \equiv_\mathcal{A} \xi$ is in $\mathsf{D}^p = \mathsf{NP} \wedge \mathsf{coNP}$ for a given ξ. Hence, we can decide whether Q has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$ in $\mathsf{coNP} \vee \bigvee_{i=1}^{k} (\mathsf{NP} \wedge \mathsf{coNP}) = \mathsf{C}^p_{2k+1}$, where k denotes the number of non-empty query plans in QP_Q that conform to $\mathcal{A}$.

We first verify that deciding whether Q is not satisfiable is in coNP. Indeed, the complement problem that decides whether Q is satisfiable is in NP: simply guess disjuncts in Q, resulting in a CQ query Q' and guess a valuation ν of the tableau representation $(T_{Q'}, \bar{u})$ of Q'. If $\nu(T_{Q'}) \models \mathcal{A}$ then Q is satisfiable. Otherwise, reject the guess.

To show that $\xi \sqsubseteq_\mathcal{A} Q$ is in NP, for each element query Q_{ξ_e} of Q_ξ, guess disjuncts in Q, resulting in a CQ query Q_e, and guess a candidate homomorphism from Q_e to Q_{ξ_e}. There are only a constant number of such element queries; so we can guess candidate homomorphism from Q to all Q_{ξ_e} in one guess. It remains to verify whether the candidate mappings are homomorphism from Q_e to each element query Q_{ξ_e}. If so, $Q_{\xi_e} \sqsubseteq Q_e$ and hence $\xi \sqsubseteq_\mathcal{A} Q$. If not, we reject the guess. This is clearly an NP process.

Furthermore, $Q \sqsubseteq_\mathcal{A} \xi$ can be decided in coNP, since its complement problem to decide $Q \not\sqsubseteq_\mathcal{A} \xi$ is in NP. Indeed, guess disjuncts in Q, resulting in a CQ query Q', and a valuation ν of the tableau representation $(T_{Q'}, \bar{u})$ of Q'. Next, verify whether $\nu(T_{Q'}) \models \mathcal{A}$ but $\nu(\bar{u}) \notin \xi(\nu(T_{Q'}))$. The latter step can be done in PTIME because ξ is of constant size. If successful, we have guessed a counterexample for $Q \sqsubseteq_\mathcal{A} \xi$. Hence, $Q \sqsubseteq_\mathcal{A} \xi$ can be decided in coNP and deciding whether $Q \equiv_\mathcal{A} \xi$ is in $\mathsf{NP} \wedge \mathsf{coNP}$, as desired. □

A simple characterization. We next give a sufficient and necessary condition for query Q to have a bounded rewriting. This condition is generic: Q is not necessarily a CQ, and $\mathcal{R}, M, \mathcal{A}$ and $\mathcal{V}$ do not have to be fixed. We use the following notations. For candidate plan $\xi \in \mathsf{QP}_Q$, we say that ξ is a *maximum plan* with $(\mathcal{A}, \mathcal{V})$ if (a) $\xi \sqsubseteq_{\mathcal{A}} Q$, and (b) there exists no $\xi' \in \mathsf{QP}_Q$ such that $\xi' \sqsubseteq_{\mathcal{A}} Q$ and $\xi \sqsubset_{\mathcal{A}} \xi'$. We say that ξ is *unique* in QP_Q if there exists no another maximum plan $\xi' \in \mathsf{QP}_Q$ such that $\xi \not\equiv_{\mathcal{A}} \xi'$.

LEMMA 3.12. *A query Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$ if and only if there exists a unique maximum plan $\xi \in \mathsf{QP}_Q$ up to $\mathcal{A}$-equivalence such that $Q \sqsubseteq_{\mathcal{A}} \xi$.* □

Proof: First assume that there exists a maximum candidate plan $\xi \in \mathsf{QP}_Q$ with $(\mathcal{A}, \mathcal{V})$, and $Q \sqsubseteq_{\mathcal{A}} \xi$. Then $\xi \equiv_{\mathcal{A}} Q$. Hence Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$ by the definition of maximum plans. Conversely, assume that Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$. Then there exists a query plan $\xi \in \mathsf{QP}_Q$ such that $\xi \equiv_{\mathcal{A}} Q$. We show that ξ is maximum and unique. Suppose by contradiction that ξ is not maximum. Then there exists another plan $\xi' \in \mathsf{QP}_Q$ such that $\xi' \sqsubseteq_{\mathcal{A}} Q$ and $\xi \sqsubset_{\mathcal{A}} \xi'$. Then $\xi \sqsubset_{\mathcal{A}} \xi' \sqsubseteq_{\mathcal{A}} Q$, contradicting the assumption that $\xi \equiv_{\mathcal{A}} Q$. Similarly, if ξ is not unique, then there exists another maximum plan ξ_1 such that $\xi \not\equiv_{\mathcal{A}} \xi_1$, Then by the definition of maximum plans, $\xi_1 \sqsubseteq_{\mathcal{A}} Q$. Since $\xi \equiv_{\mathcal{A}} Q$, $\xi_1 \sqsubseteq_{\mathcal{A}} \xi$; hence $\xi_1 \sqsubset_{\mathcal{A}} \xi$ since $\xi \not\equiv_{\mathcal{A}} \xi_1$; this contradicts to the assumption that ξ_1 is maximum. □

4. BOUNDED REWRITING FOR ACQ

To further understand the inherent complexity of VBRP, in this section we study VBRP under the following two practical conditions.

(1) Acyclic conjunctive queries, denoted by ACQ. A CQ Q is *acyclic* if its hypergraph has hypertree-width 1 [Gottlob et al. 1999]. The *hypergraph* of Q is a hypergraph (V_h, E_h) in which V_h consists of variables in Q and E_h has an edge for each set of variables that occur together in a relation atom in Q. Acyclic conjunctive queries are commonly used in practice since query evaluation and containment for ACQ are in PTIME (see [Abiteboul et al. 1995] about ACQ). As an example, query Q_0 of Example 1.1 is an ACQ.

(2) Fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$. We consider predefined database schema $\mathcal{R}$, access schema $\mathcal{A}$, bound M and views $\mathcal{V}$. After all, for an application, $\mathcal{R}$ is designed first, M is determined by our resources (*e.g.,* available processors and time constraints), access constraints are discovered from sample instances of $\mathcal{R}$, and views are selected based on the application [Armbrust et al. 2013]. These are determined before we start answering queries. Thus it is practical to assume fixed $\mathcal{R}$, $\mathcal{A}$, M, and $\mathcal{V}$.

In this setting, we study bounded rewriting of ACQ. Given an ACQ Q, we want to find an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$ in CQ (see Section 2) such that the query Q_ξ expressed by ξ is an ACQ. Our main conclusion is that the intractability of VBRP is rather robust, even for ACQ under fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$. Nonetheless, we characterize when VBRP(ACQ) is tractable and identify tractable special cases.

Intractability. One might think that VBRP would become simpler for ACQ, since query evaluation and containment for ACQ are in PTIME, not to mention fixed $\mathcal{R}$, $\mathcal{A}, M$ and $\mathcal{V}$. Unfortunately, VBRP remains intractable for ACQ under fixed $\mathcal{R}$, $\mathcal{A}, M$ and $\mathcal{V}$, even under quite restrictive access constraints in a fixed $\mathcal{A}$.

THEOREM 4.1. *Given fixed $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$,* VBRP(ACQ) *is* coNP*-hard when $\mathcal{A}$ has one of the following forms:*

(1) $\mathcal{A}$ consists of a single access constraint of the form $R(A \to B, N)$ and $N \geqslant 2$; or
(2) $\mathcal{A}$ consists of two constraints $R(A \to B, 1)$ and $R'(\emptyset \to (E, F), N)$, and $N \geqslant 6$; or

(3) $\mathcal{A}$ consists of two constraints $R((A,B) \to C, 1)$ and $R'(\emptyset \to E, N)$, and $N \geqslant 2$. □

Proof: We defer the proofs for the three cases to the electronic appendix due to the lack of space. The idea is to show that $Q \equiv_{\mathcal{A}} \emptyset$ if and only if Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$. That is, the only M-bounded query plan for Q using $\mathcal{V}$ under $\mathcal{A}$ is the empty query plan. As a consequence, the query plan does not use $\mathcal{V}$, and hence the proofs work for any fixed set $\mathcal{V}$ of views. The only information needed in the reduction is the size $|\mathcal{V}|$. Therefore, we do not specify which views are used in the reduction as any set of views will do. In addition, the proofs use fixed $\mathcal{R}$, $\mathcal{A}$ and M, and we construct an ACQ query Q by only using relations involved in $\mathcal{A}$.

(1) When $\mathcal{A}$ consists of a single $R(A \to B, N)$ and $N \geqslant 2$. We show that VBRP(ACQ) is coNP-hard in this setting by reduction from the complement of the precoloring extension problem, which is NP-complete [Kratochvíl 1993]. Given an undirected graph $G = (V_G, E)$, a precoloring μ_0 is a coloring of a *subset* W of the nodes of V_G with colors in $\{r, g, b\}$. The precoloring extension problem is to decide whether μ_0 can be extended to a coloring μ of the *entire* set of nodes in V_G with colors in $\{r, g, b\}$. That is, whether there exists a coloring μ of all nodes in V_G such that $\mu(v) = \mu_0(v)$ for each $v \in W$ and $\mu(v) \neq \mu(w)$ whenever $(v, w) \in E$. The reduction is given in the electronic appendix.

(2) When $\mathcal{A}$ consists of two access constraints $R(A \to B, 1)$ and $R'(\emptyset \to (E, F), N)$, and $N \geqslant 6$. We show the lower bound in this setting by reduction from the complement of the 3-Colorability problem, which is NP-complete (cf. [Garey and Johnson 1979]).

(3) When $\mathcal{A}$ consists of $R((A,B) \to C, 1)$ and $R'(\emptyset \to E, N)$, and $N \geqslant 2$. We show the lower bound by reduction from the complement of the 3SAT problem (see the proof of Theorem 3.4 for the definition of 3SAT). □

Characterization. In light of Theorem 4.1, we next characterize when VBRP($\mathcal{C}$) is tractable for sub-classes $\mathcal{C}$ of ACQ, and give an upper bound for VBRP(ACQ).

THEOREM 4.2. *When $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ are fixed, (1) for any sub-class $\mathcal{C}$ of* ACQ*,* VBRP($\mathcal{C}$) *is in* PTIME *if and only if for each query $Q \in \mathcal{C}$, it is in* PTIME *to check whether $Q \equiv_{\mathcal{A}} \xi$, where ξ is a query plan of size at most M, and (2)* VBRP(ACQ) *is in* coNP. □

The result tells us that ACQ and fixed parameters together simplify the analysis of VBRP (unless P = NP), to an extent, as opposed to the Σ_3^p-completeness of Theorem 3.1 and C_{2k+1}^p-completeness of Theorem 3.11. Putting Theorems 4.1 and 4.2 together, we can see that the cases of VBRP(ACQ) stated in Theorem 4.1 are coNP-complete.

The proof of Theorem 4.2 is based on Lemma 3.12 and the lemma below, which gives the complexity of basic operations for computing maximum query plans.

LEMMA 4.3. *For fixed $\mathcal{R}, \mathcal{A}, M, \mathcal{V}$, given a* CQ *$Q$ and query plans $\xi, \xi' \in \mathsf{QP}_Q$, it is in*

(a) PTIME *to check whether ξ conforms to $\mathcal{A}$,*
(b) PTIME *to check whether $\xi \sqsubseteq_{\mathcal{A}} Q$ if Q is an* ACQ*,*
(c) NP *to check whether $Q \not\sqsubseteq_{\mathcal{A}} \xi$, and*
(d) PTIME *to check whether $\xi' \sqsubseteq_{\mathcal{A}} \xi$ for $\xi' \in \mathsf{QP}_Q$.* □

Proof: When M is a constant, the set QP_Q of all candidate query plans for Q using $\mathcal{V}$ that are no larger than M consists of a constant number of query plans ξ. Moreover, observe the following. For each plan $\xi \in \mathsf{QP}_Q$, let Q_ξ be the CQ expressing ξ, after unfolding the views in ξ, *i.e.,* substituting the view definition for each view used in

ξ. Then $|Q_\xi|$ is bounded by $O(M \cdot |\mathcal{V}|)$, and the number of variables in Q_ξ is at most $O(M \cdot |\mathcal{V}| \cdot |\mathcal{R}|)$. Recall that each element query of Q_ξ can be represented as $Q_\xi \wedge \phi$, where ϕ is a conjunction of equality atoms between variables used in Q_ξ (see the proof of Theorem 3.4). Hence Q_ξ has $2^{O((M \cdot |\mathcal{V}| \cdot |\mathcal{R}|)^2)}$ many element queries. When $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ are fixed, $O(M \cdot |\mathcal{V}| \cdot |\mathcal{R}|)$ and $2^{O((M \cdot |\mathcal{V}| \cdot |\mathcal{R}|)^2)}$ are bounded by constants. Hence the size of Q_ξ is a constant, and Q_ξ has a constant number of element queries. Similarly, we can show that Q has $2^{O(|Q|^2)}$ many element queries, *i.e.,* exponentially many.

We next verify the claims of Lemma 4.3 one by one.

(a) We use the algorithm given in the proof of Lemma 3.8 to check whether ξ conforms to $\mathcal{A}$. We show that the algorithm is in PTIME for CQ in this setting. It suffices to show that its step (3) is in PTIME here instead of coNP. For each $\mathsf{fetch}(X \in S_j, R, Y)$ operation in ξ, let ξ_1 be the sub-tree of ξ rooted at S_j, and rewrite ξ_1 into a CQ Q_1 by unfolding views in ξ_1. As shown above, Q_1 has a constant number of element queries, and the size of each element query is bounded by a constant. Then by Lemma 3.7, step (3) of the algorithm can be done in PTIME. Thus checking whether ξ conforms to $\mathcal{A}$ is in PTIME.

(b) It is easy to show that $\xi \sqsubseteq_{\mathcal{A}} Q$ if and only if for each element query Q_e of Q_ξ, $Q_e \sqsubseteq Q$ (see the proof of Theorem 3.4). Since Q is an ACQ, one can check whether $Q_e \sqsubseteq Q$ in $O(|Q| \cdot |Q_e|^2)$ time, by using the Acyclic Containment algorithm from [Chekuri and Rajaraman 2000]. As Q_ξ has a constant number of element queries, checking whether $\xi \sqsubseteq_{\mathcal{A}} Q$ is in PTIME. Note that if Q is a CQ instead of an ACQ, this is not in PTIME.

(c) In contrast, Q has $2^{O(|Q|^2)}$ element queries, and checking whether $Q \not\sqsubseteq_{\mathcal{A}} \xi$ is in NP, rather than in PTIME as in (b). This can be done as follows: guess an element query Q_e of Q, and check whether $Q_e \not\sqsubseteq \xi$. As remarked earlier, ξ can be expressed by a CQ Q_ξ of size bounded by a constant. Thus the number of candidate homomorphic mappings from Q_ξ to Q_e is at most $O(|Q_e|^{|Q_\xi|})$, which is a polynomial. Thus we can check $Q_e \not\sqsubseteq Q_\xi$ in PTIME by enumerating all candidate mappings and verifying whether one of them is indeed a homomorphism from Q_ξ to Q_e. Hence it is in NP to check whether $Q \not\sqsubseteq_{\mathcal{A}} \xi$.

(d) For any $\xi' \in \mathsf{QP}_Q$, we first rewrite ξ' into a query $Q_{\xi'}$ in CQ by unfolding views in ξ'. Then $\xi' \sqsubseteq_{\mathcal{A}} \xi$ if and only if $Q_{\xi'} \sqsubseteq_{\mathcal{A}} Q_\xi$. As argued above, $Q_{\xi'}$ has a constant number of element queries, and Q_ξ and all element queries of $Q_{\xi'}$ have a constant size. Hence checking $Q_{\xi'} \sqsubseteq_{\mathcal{A}} Q_\xi$ is in PTIME, and so is checking $\xi' \sqsubseteq_{\mathcal{A}} \xi$. □

Proof of Theorem 4.2. Based on the lemmas, we prove Theorem 4.2. We first present an algorithm, denoted by $\mathsf{Alg_{ACQ}}$, to check whether an ACQ Q has an M-bounded rewriting. For fixed $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$, we show that the algorithm is in coNP for general ACQ queries. However, when we focus on specific sub-classes $\mathcal{C}$ of ACQ, the algorithm runs in PTIME. More specifically, classes $\mathcal{C}$ have the following property: for each query $Q \in \mathcal{C}$, it is in PTIME to check whether $Q \equiv_{\mathcal{A}} \xi$. Here, ξ is a query plan of size at most M.

From Lemma 3.12, we know that a query Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$ if and only if there exists a unique maximum query plan $\xi \in \mathsf{QP}_Q$ (up to $\mathcal{A}$-equivalence) such that $Q \sqsubseteq_{\mathcal{A}} \xi$. To develop $\mathsf{Alg_{ACQ}}$, we first show that given any ACQ Q, its unique maximum plan ξ (up to $\mathcal{A}$-equivalence) can be computed in PTIME, if it exists. It is computed by the algorithm given below, denoted by $\mathsf{Alg_{MP}}$:

(1) generate the set QP_Q of all candidate query plans for Q of length at most M, using relation atoms in $\mathcal{R}$ and views in $\mathcal{V}$;
(2) remove from QP_Q all plans $\xi \in \mathsf{QP}_Q$ such that its CQ query Q_ξ is not acyclic;
(3) remove from QP_Q all $\xi \in \mathsf{QP}_Q$ such that $\xi \not\sqsubseteq_{\mathcal{A}} Q$ or ξ does not conform to $\mathcal{A}$;

(4) remove from QP_Q all query plans $\xi \in \mathsf{QP}_Q$ such that there exists another query plan ξ_1 satisfying $\xi_1 \sqsubseteq_{\mathcal{A}} Q$ and $\xi \sqsubset_{\mathcal{A}} \xi_1$;
(5) if QP_Q is nonempty and all remaining plans in QP_Q are $\mathcal{A}$-equivalent to each other, return a plan ξ in QP_Q; otherwise return "no".

The correctness of algorithm is obvious. For its complexity, step (1) is in PTIME since there exist a constant number of plans in QP_Q, and each of them has size bounded by a constant. Step (2) is in PTIME since checking whether a CQ query is acyclic can be done in PTIME by using, *e.g.,* GYO algorithm [Graham 1979; Yu and Özsoyoğlu 1979]). Step (3) consists of (i) checking whether ξ conforms to $\mathcal{A}$; and (ii) checking whether $\xi \not\sqsubseteq_{\mathcal{A}} Q$. These are in PTIME by Lemma 4.3(a) and (b). Step (4) checks $\mathcal{A}$-containment between query plans and $\mathcal{A}$-containment of queries plans in Q. These are in PTIME by Lemma 4.3(b) and (d). In contrast, when Q is CQ, steps (3) and (4) have to call an NP oracle for a constant number of times. This explains why VBRP(ACQ) differs from VBRP(CQ) (Theorem 3.11) unless P = NP. Step (5) checks $\mathcal{A}$-containment of query plans, in PTIME by Lemma 4.3(d). Putting these together, algorithm $\mathsf{Alg}_{\mathsf{MP}}$ is in PTIME.

Capitalizing on $\mathsf{Alg}_{\mathsf{MP}}$, algorithm $\mathsf{Alg}_{\mathsf{ACQ}}$ works as follows. Given an ACQ Q, it first checks whether Q has a unique maximum plan ξ in QP_Q, by invoking $\mathsf{Alg}_{\mathsf{MP}}$. If such a query plan does not exist, then Q does not have an M-bounded query rewriting by Lemma 3.12, and hence $\mathsf{Alg}_{\mathsf{ACQ}}$ returns false. Otherwise, it checks whether $Q \equiv_{\mathcal{A}} \xi$; it returns true if so, and false otherwise.

We now prove the two statements of Theorem 4.2 by analyzing $\mathsf{Alg}_{\mathsf{ACQ}}$.

(1) Sub-classes $\mathcal{C}$. Consider a sub-class $\mathcal{C}$ of ACQ such that for each $Q \in \mathcal{C}$, checking whether $Q \equiv_{\mathcal{A}} \xi$ is in PTIME. As argued above, $\mathsf{Alg}_{\mathsf{MP}}$ is in PTIME. Then $\mathsf{Alg}_{\mathsf{ACQ}}$ is in PTIME, and hence so is VBRP($\mathcal{C}$). Conversely, given $Q \in \mathcal{C}$ and ξ, we can check whether $Q \equiv_{\mathcal{A}} \xi$ as follows: (1) compute a unique maximum plan $\xi_Q \in \mathsf{QP}_Q$; (2) check whether $\xi_Q \equiv_{\mathcal{A}} \xi$ and Q has an M-bounded rewriting; return true if so, and false otherwise. The correctness and time complexity follow from Lemmas 3.12, 4.3(c), (d), and the assumption that VBRP($\mathcal{C}$) is in PTIME. In fact, to ensure that VBRP($\mathcal{C}$) is in PTIME, we only need to show that deciding $Q \sqsubseteq_{\mathcal{A}} \xi$ is in PTIME. Indeed, by Lemma 4.3, checking $\xi \sqsubseteq_{\mathcal{A}} Q$ is in PTIME, and hence we also have that deciding $Q \equiv_{\mathcal{A}} \xi$ is in PTIME. However, the converse does not hold. That is, when VBRP($\mathcal{C}$) is in PTIME, it is not necessary that deciding $Q \sqsubseteq_{\mathcal{A}} \xi$ is in PTIME. In particular, if $\xi \not\sqsubseteq_{\mathcal{A}} Q$ and if Q has no M-bounded rewriting, then we cannot further infer that $Q \sqsubseteq_{\mathcal{A}} \xi$.

(2) ACQ. For a general query Q in ACQ, checking whether $Q \sqsubseteq_{\mathcal{A}} \xi$ is in coNP by Lemma 4.3(c). Since $\mathsf{Alg}_{\mathsf{MP}}$ is in PTIME, $\mathsf{Alg}_{\mathsf{ACQ}}$ is in coNP, and so is VBRP(ACQ). □

Theorem 4.2 helps us identify sub-classes of ACQ for which VBRP is tractable, such as ACQ under "FDs", *i.e.,* when all the access constraints in $\mathcal{A}$ are of the form $R(X \to Y, 1)$. As remarked earlier, FDs with associated indices are common access constraints, and can be discovered by using existing tools for mining FDs (*e.g.,* [Huhtala et al. 1999]).

COROLLARY 4.4. *When $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$ are fixed,* VBRP *is in* PTIME *for* ACQ *if $\mathcal{A}$ consists of* FDs *only.* □

Proof: By Theorem 4.2, it suffices to show that checking whether $Q \sqsubseteq_{\mathcal{A}} \xi$ is in PTIME, where ξ denotes the unique maximum query plan, if it exists.

Given a set $\mathcal{A}$ of access constraints of the FD form and an ACQ Q, we chase the tableau T of Q by $\mathcal{A}$ as follows [Aho et al. 1979]: for each $R(X \to Y, 1) \in \mathcal{A}$, if there exist tuples $R(\bar{x}, \bar{y}_1, \bar{z}_1)$ and $R(\bar{x}, \bar{y}_2, \bar{z}_2)$ in T such that (a) $\bar{x}$ corresponds to X, and (b) $\bar{y}_1$ and $\bar{y}_2$ correspond to Y, and $\bar{y}_1 \neq \bar{y}_2$, then we unify $\bar{y}_1 = \bar{y}_2$ in T. These yield a tableau

$T_{\mathcal{A}}$ satisfying $\mathcal{A}$. Let $Q^{\mathcal{A}}$ be the query expressed by $T_{\mathcal{A}}$. One can see that $Q^{\mathcal{A}}$ is unique up to homomorphism [Maier et al. 1979], $Q^{\mathcal{A}} \equiv_{\mathcal{A}} Q$ and $Q^{\mathcal{A}}$ satisfies $\mathcal{A}$.

Observe the following: (a) $Q \sqsubseteq_{\mathcal{A}} \xi$ is equivalent to $Q^{\mathcal{A}} \sqsubseteq \xi$; the latter is in terms of conventional query containment $\sqsubseteq$; and (b) $Q^{\mathcal{A}} \sqsubseteq \xi$ can be checked in PTIME since ξ is of constant size. Indeed, at most $O(|Q^{\mathcal{A}}|^{|\xi|})$ homomorphic mappings need to be checked. From these it follows that whether $Q \sqsubseteq_{\mathcal{A}} \xi$ can be checked in PTIME. □

In contrast to Corollary 4.4, VBRP remains intractable for CQ under FDs, although the analysis is simpler compared with Theorem 3.11 (unless P = NP).

PROPOSITION 4.5. *For fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$,* VBRP(CQ) *is* NP-*complete if $\mathcal{A}$ consists of* FDs *only. It remains* NP-*complete when none of $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$ is fixed.* □

Proof: We first show the lower bound, followed by the upper bound.

Lower bound. We show that in this setting, VBRP(CQ) is NP-hard by reduction from the 3SAT problem (see the proof of Theorem 3.4 for 3SAT). Given an instance ψ of 3SAT, we define a CQ Q, an access schema $\mathcal{A}$ of the FD form, a bound M, and a set $\mathcal{V}$ of CQ views, such that Q has an M-bounded rewriting in CQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if ψ is satisfiable. We ensure that $M, \mathcal{A}, \mathcal{R}$ and $\mathcal{V}$ do not depend on ψ, *i.e.,* they are fixed.

(1) The database schema $\mathcal{R}$ consists of $R_\vee(B, A_1, A_2)$, $R_\wedge(B, A_1, A_2)$ and $R_\neg(A, \bar{A})$ to encode the Boolean operations, as in the proof of Theorem 3.4 (see Figure 2). Observe that we do not include R_{01} in $\mathcal{R}$. The reason is that we cannot enforce instances of R_{01} to coincide with I_{01} (see Figure 2) using access constraints of the FD form.

(2) The access schema $\mathcal{A}$ contains the following three constraints to ensure that $R_\vee$, $R_\wedge$, $R_\neg$ can be used to encode the Boolean operations: $R_\vee((A_1, A_2) \to B, 1), R_\wedge((A_1, A_2) \to B, 1), R_\neg(A \to \bar{A}, 1)$. All these constraints have the form of $R(X \to Y, 1)$.

(3) The query Q is defined as $Q() = Q_c() \wedge Q_\psi(\bar{x}, 1)$, where (a) $Q_c()$ is the same as its counterpart given in the proof of Theorem 3.4, except for the sub-query in Q_c related to R_{01}. It is to ensure that the instances of $R_\vee$, $R_\wedge$, and $R_\neg$ contain all the tuples shown in Figure 2, and (b) $Q_\psi(\bar{x}, 1)$ is similar to its counterpart given in the proof of Theorem 3.4, to encode all truth assignments μ of $\bar{x}$ such that $\psi(\mu(\bar{x})) = \mathsf{true}$, expressed in terms of $R_\vee$, $R_\wedge$ and $R_\neg$. In contrast to the query used in the proof of Theorem 3.4, Q_ψ extracts the Boolean domain from $R_\neg$ rather than R_{01}. Note that if an instance $\mathcal{D}$ of $\mathcal{R}$ is equal to the instances shown in Figure 2 (excluding I_{01}), then $Q(\mathcal{D})$ is nonempty if and only if ψ is satisfiable. Of course, when $\mathcal{D}$ is an instance of $\mathcal{R}$ that satisfies $\mathcal{A}$ but it contains more tuples than those shown in Figure 2, $Q_c(\mathcal{D}) \neq \emptyset$ and $Q(\mathcal{D}) = \emptyset$ still ensure that ψ is unsatisfiable, but $Q_c(\mathcal{D}) \neq \emptyset$ and $Q(\mathcal{D}) \neq \emptyset$ do not imply that ψ is satisfiable. Indeed, $\bar{x}$ may be non-Boolean, and Q_ψ does not correctly evaluate ψ in this case.

(4) The set $\mathcal{V}$ consists of a single CQ view: $V() = Q_c()$, which is the same as the one given in Q. Finally, we let $M = 1$.

Since all access constraints in $\mathcal{A}$ are FDs and $M = 1$, the only possible query plans are $\emptyset$ and V. One can verify that Q has an 1-bounded rewriting in CQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if $V \equiv_{\mathcal{A}} Q$ if and only if ψ is true. Note that $M, \mathcal{A}, \mathcal{R}$ and $\mathcal{V}$ are fixed.

Upper bound. We give the following NP algorithm to check VBRP(CQ) when none of $\mathcal{R}, \mathcal{A}, \mathcal{V}$ and M is fixed, and when $\mathcal{A}$ consists of FD-like access constraints only:

(1) chase the tableau T_Q of Q by $\mathcal{A}$ as described in the proof of Corollary 4.4; this yields tableau T_{Q_1} that satisfies $\mathcal{A}$; let Q_1 be the CQ represented by T_{Q_1};

(2) guess a query plan ξ such that $|\xi| \leqslant M$, a CQ query Q_2 such that the tableau of Q_2 satisfies $\mathcal{A}$ and $|Q_2| \leqslant M \cdot |V| \cdot |\mathcal{R}|$, a homomorphism h_1 from Q_1 to Q_2, and a homomorphism h_2 from Q_2 to Q_1;
(3) check whether ξ conforms to $\mathcal{A}$; if not, then reject the guess; otherwise, continue;
(4) rewrite ξ into a CQ query Q' by unfolding views in ξ;
(5) chase the tableau $T_{Q'}$ of Q' by $\mathcal{A}$, which yields $T_{Q'_1}$ that satisfies $\mathcal{A}$;
(6) syntactically check whether the tableau T_{Q_2} of Q_2 is the same as $T_{Q'_1}$, *i.e.,* a tuple template t is in T_{Q_2} if and only if t is in $T_{Q'_1}$; if not, then reject the guess; otherwise, continue;
(7) check whether h_1 and h_2 are homomorphic mappings, and whether h_1 witnesses $Q_2 \sqsubseteq Q_1$ and h_2 witnesses $Q_1 \sqsubseteq Q_2$; if so, return true; otherwise reject the guess.

The algorithm is obviously correct. For its complexity, we need the following Lemma.

LEMMA 4.6. *If $\mathcal{A}$ consists of* FDs *only, it is in* PTIME *to decide whether a plan $\xi \in$ QP$_Q$ conforms to $\mathcal{A}$.* □

Using the lemma, we show that the algorithm for VBRP(CQ) is in NP. Since the chase can be done in PTIME, steps (1) and (5) are in PTIME. By Lemma 4.6, step (3) can be done in PTIME. Step (4) is in PTIME. Step (6) does syntactic checking and is in PTIME. Because homomorphic mappings can be verified in PTIME, step (7) is also in PTIME.

This concludes the proof of Proposition 4.5, modulo the proof of Lemma 4.6. □

We next verify Lemma 4.6.

Proof of Lemma 4.6. To check whether ξ conforms to $\mathcal{A}$, we check whether for each fetch$(X \in S_j, R, Y)$ operation in ξ, the following conditions hold: (a) there is an access constraint $R(X \to Y', 1)$ in $\mathcal{A}$ such that $Y \subseteq X \cup Y'$; and (b) there exists a constant N such that for all instances $\mathcal{D}$ of $\mathcal{R}$ that satisfy $\mathcal{A}$, $|S_j| \leqslant N$ in the computation of $\xi(\mathcal{D})$.

For each fetch$(X \in S_j, R, Y)$ operation, it is in PTIME to check condition (a). We use the following algorithm to check condition (b). Let ξ_j be the sub-tree of ξ rooted at S_j, and Q_j be the query expressed by ξ_j. The algorithm works as follows:

(1) unfold Q_j by replacing each view with its definition, yielding Q'_j in CQ;
(2) chase the tableau $T_{Q'_j}$ of Q'_j by $\mathcal{A}$ as described in the proof of Corollary 4.4; this yields tableau $T_{Q''_j}$ that satisfies $\mathcal{A}$; let Q''_j be the CQ represented by $T_{Q''_j}$;
(3) check whether Q''_j has bounded output; if so, return true; otherwise, return false.

From the proof of Corollary 4.4, we can see that $Q''_j \equiv_{\mathcal{A}} Q_j$. Then the correctness of the algorithm follows. For its complexity, observe that step (1) is in PTIME. Since the chase can be done in PTIME, step (2) is in PTIME. Because the tableau $T_{Q''_j}$ satisfies $\mathcal{A}$, and computing cov$(Q''_j, \mathcal{A})$ is in PTIME, step (3) is in PTIME by Lemma 3.6. Since there are at most $O(|\xi|)$ fetch operations in ξ, the algorithm is in PTIME. □

Along the same lines as Corollary 4.4, one can verify that for fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$, VBRP is in PTIME for the sub-class of ACQ queries such that their tableau representations satisfy the cardinality constraints in $\mathcal{A}$. A special case of this is when $\mathcal{A} = \emptyset$, *e.g.,* the setting of [Armbrust et al. 2013], when access constraints are not employed at all.

Theorem 4.2 remains intact on any class $\mathcal{C}$ of queries as long as it is in PTIME to compute a maximum plan in QP$_Q$ for all queries in $\mathcal{C}$. Examples include

(1) *self-join-free* CQ, *i.e.,* the class of CQ queries that contain no repeated relation names, and

(2) CQ *with a fixed number of variables*, *i.e.,* for each constant k, the class of CQ queries that have at most k free variables.

By Theorem 4.2 and Lemma 4.3, VBRP is also in PTIME in these two cases.

The results of the section tell us that the intractability of VBRP(ACQ) is robust. The proof of Theorem 4.1 shows that $\mathcal{A}$ is the crucial parameter here, while $\mathcal{V}$ and M could be empty and 0, respectively. Not all is lost. There are practical cases when VBRP(ACQ) and even VBRP(CQ) are tractable. Moreover, we can cope with the hardness by means of effective syntax (Section 5) and approximate query answering (Section 8).

5. AN EFFECTIVE SYNTAX

We have seen that the undecidability of VBRP for FO and the intractability for CQ are rather robust. Can we still make practical use of bounded rewriting analysis when querying big data? We next show that the answer is affirmative.

We develop effective syntax for FO queries that have a bounded rewriting, to syntactically check the existence of bounded rewriting in PTIME without sacrificing the expressive power. More specifically, for any database schema $\mathcal{R}$, views $\mathcal{V}$, access schema $\mathcal{A}$ and bound M, we identify two classes of FO queries, (a) a class of queries *topped by* $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, which "covers" all FO queries over $\mathcal{R}$ that have an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, up to $\mathcal{A}$-equivalence, and (b) a class of *size-bounded* queries, which "covers" all the views of $\mathcal{V}$ in FO that have bounded output for all instances $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$. The second class is to effectively check bounded output (see Section 3.1). We show that it is in PTIME to syntactically check whether a query is topped or size-bounded.

Below we first present the main results of the section in Section 5.1. We then define topped queries and size-bounded queries in Sections 5.2 and 5.3, respectively.

5.1. Practical Use of Bounded Rewriting

The main results of the section are as follows.

THEOREM 5.1. *For any $\mathcal{R}$, $\mathcal{V}$ and M, and under any $\mathcal{A}$,*

(a) each FO *query Q with an M-bounded rewriting using $\mathcal{V}$ is $\mathcal{A}$-equivalent to a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$;*

(b) every FO *query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has an M-bounded rewriting in* FO *using $\mathcal{V}$ under $\mathcal{A}$, which can be identified in* PTIME *in M, $|Q|$, $|\mathcal{V}|$ and $|\mathcal{A}|$; and*

(c) it takes PTIME *in $M, |\mathcal{R}|, |Q|, |\mathcal{V}|$ and $|\mathcal{A}|$ to check whether an* FO *query Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, which uses an oracle that checks whether* FO *views in $\mathcal{V}$ have bounded output in* PTIME *in $|Q|$.*

Here $\mathcal{A}$, Q and $\mathcal{V}$ are all defined over the same $\mathcal{R}$. □

That is, topped queries are a key sub-class of FO queries with a bounded rewriting, and can be efficiently checked. Moreover, the bounded rewriting can also be efficiently generated. For the existence of the oracle, we show the following.

THEOREM 5.2. *For any $\mathcal{R}$ and under any $\mathcal{A}$,*

(a) each FO *query Q over $\mathcal{R}$ that has bounded output is $\mathcal{A}$-equivalent to a size-bounded query under $\mathcal{A}$;*

(b) each size-bounded query has bounded output under $\mathcal{A}$; and

(c) it takes PTIME *in $|Q|$ to check whether an* FO *query Q is a size-bounded query.*

Here $\mathcal{A}$ and Q are defined over the same $\mathcal{R}$. □

Before we define topped and size-bounded queries, we remark the following. (1) Theorems 5.1 and 5.2 just aim to demonstrate the existence of effective syntax for FO queries with bounded rewriting. There are other forms of effective syntax for such FO queries. (2) Theorem 5.1 does not contradict to Corollary 3.9 due to the requirement of $\mathcal{A}$-equivalence in its condition (a), which is undecidable for FO.

Practical use of bounded query rewriting. Capitalizing on the effective syntax, we can develop algorithms (a) to check whether a given FO query Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in PTIME; and if so, (b) to generate a bounded query plan ξ for Q using $\mathcal{V}$. The existence of these algorithms is warranted by Theorems 5.1 and 5.2.

We can then support bounded rewriting on top of commercial DBMS as follows. Given an application, a database schema $\mathcal{R}$ and a resource bound M are first determined, based on the application and available resources, respectively. Then, a set $\mathcal{V}$ of views can be selected following [Armbrust et al. 2013], and a set $\mathcal{A}$ of access constraints can be discovered. After these are in place, given an FO query Q posed on an instance $\mathcal{D}$ of $\mathcal{R}$ that satisfies $\mathcal{A}$, we check whether Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. If so, we generate a bounded query plan ξ for Q using $\mathcal{V}$, by using the algorithms described above. Then we can compute $Q(\mathcal{D})$ by executing ξ with the existing DBMS. Since a commercial DBMS may not execute ξ directly, this can be carried out by translating ξ into an equivalent SQL query Q_ξ, which is passed to the underlying DBMS, as suggested in [Cao and Fan 2016]. By "implementing" fetch operations in terms of index joins and using join hints or virtual views to enforce the join orders, we can enforce DBMS to evaluate Q_ξ by exactly following ξ. Moreover, incremental methods for maintaining the views [Armbrust et al. 2013] and the indices of $\mathcal{A}$ [Cao and Fan 2016] have already been developed, in response to updates to $\mathcal{D}$. Putting these together, we can expect to efficiently answer a number of FO queries in (possibly big) $\mathcal{D}$ by leveraging bounded rewriting.

5.2. Topped Queries for Bounded Rewriting

We next define topped queries and outline a proof of Theorem 5.1.

It is nontrivial to define an effective syntax, as shown below.

Example 5.3. Consider a database schema $\mathcal{R}_1$ with two relations $R(A, B)$ and $T(C, E)$, an access schema $\mathcal{A}_2$ consisting of $R(A \to B, N)$ and $T(C \to E, N)$, and $\mathcal{V}_3$ with a single view $V_3(x, y) = R(y, y) \wedge T(x, y)$. Given a value for x, V_3 returns a bounded number of y values due to the access constraint on T. Consider FO query: $q_3(z) = q_4(z) \wedge \neg\exists w\, R(z, w)$, where $q_4(z) = \exists x \exists y\, ((R(y, y) \wedge T(x, y)) \wedge (x = 1)) \wedge R(y, z)$. Then q_3 has a 13-bounded rewriting as in Fig. 3, which is for an $\mathcal{A}$-equivalent query:

$$q_3'(z) = q_4(z) \wedge \neg(q_4(z) \wedge \exists w\, R(z, w)).$$

Observe the following. (1) Query q_3' becomes bounded because it propagates z-values from q_4 to "$\neg\exists w\, R(z, w)$". (2) Such propagated values allow us to fetch bounded data for relation atoms, *i.e.,* $R(z, w)$. (3) The part of the plan for a sub-query of q_3 may have to embed the part of the plan for another sub-query. For instance, (i) q_4 has a 5-bounded rewriting in q_3 (the left part of Fig. 3); (ii) $\exists w\, R(z, w)$ has a 7-bounded rewriting in q_3 (the right part of Fig. 3), which embeds the 5-bounded plan for q_4; and (iii) the size of the plan for q_3 is the sum of the sizes of plans for q_4 and $\exists w\, R(z, w)$, *i.e.,* $5 + 7 + 1 = 13$.

This shows that to cover queries such as q_3, topped queries have to support value propagation among sub-queries, and keep track of the sizes of plans for sub-queries. □

Topped queries. This observation motivates us to define topped queries by characterizing value propagation among their sub-queries. To do this, we define topped queries with two binary functions $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ and $\mathsf{size}((Q_s(\bar{x}), Q(\bar{z}))$ that take two

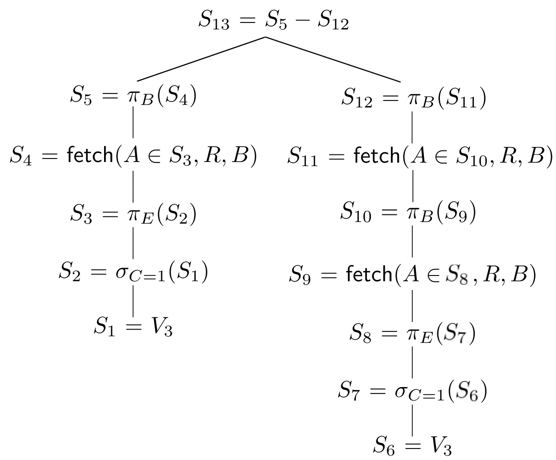

Fig. 3. A bounded plan for q_3 of Example 5.3

queries $Q_s(\bar{x})$ and $Q(\bar{z})$ as input parameters. Below we first provide intuition behind $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ and $\mathsf{size}((Q_s(\bar{x}), Q(\bar{z}))$. Using the functions, we then define topped queries, and complete the definition by giving the syntactic form of the two functions.

(1) Boolean function $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ returns true if the following condition is satisfied: if $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $Q_s(\bar{x})$ has a bounded rewriting, then $Q_s(\bar{x}) \wedge Q(\bar{z})$ also has a bounded rewriting. Intuitively, $Q(\bar{z})$ is a (sub-)query we are inspecting, and $Q_s(\bar{x})$ keeps track of sub-queries from which values are propagated to $Q(\bar{z})$.

We use $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ to check whether we can propagate values from Q_s to Q, and get a bounded rewriting of Q in $Q_s \wedge Q$. For instance, by $\mathsf{covq}(q_4(z), \exists w\, R(z, w))$ = true for $q_3(z)$ in Example 5.3, in which $q_4(z)$ is Q_s and $\exists w\, R(z, w)$ is Q, it indicates that if q_4 has a bounded rewriting, then by propagating values to free variable z of q_4, we can have a bounded rewriting for sub-query $\exists w\, R(z, w)$ in $q_4(z) \wedge \exists w\, R(z, w)$.

Note that only values of the free variables of $Q_s(\bar{x})$ can be propagated to $Q(\bar{z})$, and $Q(\bar{z})$ can only take values for its free variables as input from $Q_s(\bar{x})$. In other words, $Q(\bar{z})$ only takes values of the variables in $\bar{x} \cap \bar{z}$ from $Q_s(\bar{x})$.

In particular, Q_s may include views from $\mathcal{V}$. As will be shown shortly, function $\mathsf{covq}(Q_s, Q)$ distinguishes views that need to have bounded output from those that do not have to, to ensure that a bounded number of values are propagated from Q_s to Q over any instance $\mathcal{D} \models \mathcal{A}$, *i.e.,* Q does have a bounded rewriting sub-plan in $Q_s \wedge Q$.

(2) Function $\mathsf{size}((Q_s(\bar{x}), Q(\bar{z}))$ is a natural number that maintains an upper bound of the size of minimum sub-plans for sub-query $Q(\bar{z})$ in $Q_s(\bar{x}) \wedge Q(\bar{z})$. We will use $\mathsf{size}(Q_s, Q)$ to ensure that our query plans do not exceed a given bound M.

For instance, in Example 5.3, $\mathsf{size}(q_4, \exists w\, R(z, w))$ = 7, which is the size of the sub-plan for evaluating $\exists w\, R(z, w)$ in $q_4 \wedge \exists\, w R(z, w)$ by using values propagated from q_4.

We now define *topped queries* using the two functions. An FO query Q over $\mathcal{R}$ is *topped* by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ if (1) $\mathsf{covq}(Q_\varepsilon, Q)$ = true; and (2) $\mathsf{size}(Q_\varepsilon, Q) \leqslant M$. Here Q_ε is a "tautology query" such that for any Q, $Q_\varepsilon \wedge Q$ = Q and Q_ε has a 0-bounded plan. It is an extension of functions $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$ for function parameter Q_s.

Intuitively, we compute $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$ starting with $Q_s = Q_\varepsilon$, and conclude that Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ if the two conditions above are satisfied.

Functions $\mathsf{covq}(\cdot, \cdot)$ **and** $\mathsf{size}(\cdot, \cdot)$. We next define the functions inductively based on the structure of FO query Q. In the process, we also give a bounded query plan. We will ensure that if $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = $\mathsf{covq}(Q_\varepsilon, Q_s(\bar{x}))$ = true and $Q_s(\bar{x})$ has a $\mathsf{size}(Q_\varepsilon, Q_s(\bar{x}))$-bounded rewriting, then $Q_s(\bar{x}) \wedge Q(\bar{z})$ has a $\mathsf{size}(Q_\varepsilon, Q_s(\bar{x}) \wedge Q(\bar{z}))$-bounded rewriting.

The definition of $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ is separated into 7 cases below. In particular, we define $\mathsf{covq}(Q_s(\bar{x}), Q_\varepsilon)$ = true, $\mathsf{size}(Q_s(\bar{x}), Q_\varepsilon) = 0$.

(1) $Q(\bar{z})$ is $z = c$. We define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$.

(2) $Q(\bar{z})$ is $V(\bar{z})$. We can access cached views; thus, we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = 1. That is, constant queries and views have 1-bounded rewriting and therefore, are taken as topped queries.

(3) $Q(\bar{z})$ is $Q'(\bar{z}) \wedge C$, where C is one of $(x = y)$, $(x \neq y)$, $(x = c)$ and $(x \neq c)$. We define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = $\mathsf{covq}(Q_s(\bar{x}), Q'(\bar{z}))$; and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{z}))+1$ when $\mathsf{covq}(Q_s(\bar{x}), Q'(\bar{z}))$ = true, and as $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$ otherwise. Given a bounded plan ξ' for Q', a bounded plan for Q is $(T = \xi', \sigma_C(\xi'))$, increasing the size of ξ' by 1.

(4) $Q(\bar{z})$ is $Q_1(\bar{z}_1) \wedge Q_2(\bar{z}_2)$, where Q_2 is not an (in)equality. Let $\mu_i = \mathsf{covq}(Q_s(\bar{x}), Q_i(\bar{z}_i))$, $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}_i))$, $s = \mathsf{size}(Q_\varepsilon, Q_s(\bar{x}))$, $\mu' = \mathsf{covq}(Q_s(\bar{x}) \wedge Q_1(\bar{z}_1), Q_2(\bar{z}_2))$, $s' = \mathsf{size}(Q_s(\bar{x}) \wedge Q_1(\bar{z}_2), Q_2(\bar{z}_2))$ $(i \in \{1,2\})$. We distinguish the following cases:

(a) if μ_1 = true, $Q_2(\bar{z}_2)$ is of the form $\exists \bar{w}\, R(\bar{z}_1, \bar{z}'_2, \bar{w})$, there exists access constraint $R(Z_1 \to Z'_2, N)$ is in $\mathcal{A}$ with $Z_1 \cup Z'_2$ = Z_2 and if $Q_s(\bar{x}) \wedge Q_1(\bar{z}_1)$ *has bounded output* under $\mathcal{A}$, then we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + 1$; otherwise

(b) if $\mu_1 = \mu_2$ = true, then $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $2s+s_1+s_2+\lambda_{(\bar{z}_1,\bar{z}_2)}$, where $\lambda_{(\bar{z}_1,\bar{z}_2)}$ is 1 (resp. 4) if $\bar{z}_1 \cap \bar{z}_2$ is empty (resp. not empty); otherwise

(c) if $\mu_1 = \mu'$ = true and $|Q_2| \leqslant K$ for some predefined constant K (here $|Q_2|$ is the size of Q_2), then $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $s_1 + s'$.

(d) otherwise we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = false and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$.

In case (4) we characterize value propagation via conjunction in the queries. More specifically, when $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true, we have three cases below.

(a) If Q_1 has a bounded plan ξ_1 with Q_s, and if Q_2 is (a projection of) a relation atom covered by an access constraint $R(Z_1 \to Z'_2, N)$ in $\mathcal{A}$, then $Q(\bar{z})$ also has a bounded plan with $Q_s(\bar{x})$ and $Q_1(\bar{z}_1)$, as long as $Q_s(\bar{x}) \wedge Q_1(\bar{z}_1)$ has *bounded output*. Indeed, a plan for Q_2 is $(T = \xi_1, \mathsf{fetch}(X \in T, R, Z'_2))$ of size $|\xi_1| + 1$. We instantiate the Z_1 attributes of R with the output of $Q_1(\bar{z}_1)$, and ensure that the input T of fetch, *i.e.,* the output of $Q_s(\bar{x}) \wedge Q_1(\bar{z}_1)$, has bounded size. This case requires bounded output analysis.

For instance, consider $q_2 = \exists x\,((R(y,y) \wedge T(x,y)) \wedge (x = 1))$ and $R(y,z)$ in sub-query q_4 of q_3 of Example 5.3. The y-values from q_2 are propagated to $R(y,z)$ in this case.

By cases (2), (3) and (7c) (will be seen shortly), one can verify that $\mathsf{covq}(Q_\varepsilon, q_2)$ = true and $\mathsf{size}(Q_\varepsilon, q_2) = 2$ under $\mathcal{A}_2$ and $\mathcal{V}_3$ of Example 5.3. Now consider query $q'_2 = q_2 \wedge R(y,z)$. By case (4a), we have that $\mathsf{covq}(Q_\varepsilon, q'_2)$ = $\mathsf{covq}(Q_\varepsilon, q_2)$ = true and $\mathsf{size}(Q_\varepsilon, q'_2)$ = $\mathsf{size}(Q_\varepsilon, q_2) + 1 = 3$. Thus q'_2 is topped by $(\mathcal{R}_1, \mathcal{V}_3, \mathcal{A}_2, 3)$ (recall $\mathcal{R}_1$, $\mathcal{V}_3$ and $\mathcal{A}_2$ from Example 5.3).

(b) If both Q_1 and Q_2 have bounded sub-plans with Q_s, *e.g.,* ξ_1 and ξ_2, respectively, then Q also has a bounded plan with Q_s, whose size depends on the forms of $Q_1(\bar{z}_1)$ and $Q_2(\bar{z}_2)$, as reflected in different values of $\lambda_{(\bar{z}_1,\bar{z}_2)}$. More specifically, if $\bar{z}_1$ and $\bar{z}_2$ are disjoint, then Q is a production of Q_1 and Q_2 and thus has a query plan $(T_1 = \xi_1$, $T_2 = \xi_2$, $T_3 = T_1 \times T_2)$, of size $|\xi_1| + |\xi_2| + 1$. Otherwise, *i.e.,* if $\bar{z}_1 \cap \bar{z}_2 \neq \emptyset$, then Q is a join of Q_1 and Q_2 and thus has a plan $(T_1 = \xi_1$, $T_2 = \xi_2$, $T_3 = \rho(T_2)$, $T_4 = T_1 \times T_2$, $T_5 = \sigma_{Z_1 \cap Z_2 = \rho(Z_1 \cap Z_2)}(T_4))$, of size $|\xi_1| + |\xi_2| + 4$. Here ρ renames attributes in $Z_1 \cap Z_2$.

Note that ξ_1 and ξ_2 are sub-plans of Q_1 and Q_2 with Q_s, respectively, which use the output of Q_s to compute Q_1 and Q_2. Hence, $|\xi_1|$ and $|\xi_2|$ are characterized by $s + s_1$ and $s + s_2$, respectively, including the size of the plan for Q_s.

For example, consider $Q_s = S(x)$, $Q_1 = R(x, y)$ and $Q_2 = R(x, z)$ over relation schemas S(C) and $R(A, B)$ with access constraints $S(\emptyset \to C, N)$ and $R(A \to B, N)$. Then $\mathsf{covq}(Q_s, Q_1 \wedge Q_2) = \mathsf{true}$ since $\mathsf{covq}(Q_s, Q_1) = \mathsf{covq}(Q_s, Q_2) = \mathsf{true}$ (by cases (7a) and (7b), which will be discussed shortly); $\mathsf{size}(Q_s, Q_1 \wedge Q_2) = B_1 + B_2 + 4$, where $B_i = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_i)$ (for $i = 1, 2$) is an upper bound of the size of the sub-plan for Q_i with Q_s that will be used by the plan for Q with Q_s; and 4 is the number of steps to join sub-plans for Q_1 and Q_2 with Q_s together. Note that $\mathsf{size}(Q_\varepsilon, Q_s)$ is counted twice as it will be used by the sub-plans for both Q_1 and Q_2 with Q_s.

(c) If Q_1 has a bounded plan with Q_s while Q_2 has a bounded plan with $Q_s \wedge Q_1$ instead of Q_s alone, *e.g.,* plans ξ_1 and ξ_2', respectively, then Q has a bounded query plan of size $|\xi_1| + |\xi_2'|$, where $|\xi_1| = \mathsf{size}(Q_s(\bar{x}), Q_1(\bar{z}_1))$ and $|\xi_2'| = \mathsf{size}(Q_s \wedge Q_1(\bar{x}), Q_2(\bar{z}_2))$. Note that we extend $Q_s(\bar{x})$ with $Q_1(\bar{z}_1)$ only if $Q_1(\bar{z}_1)$ has a bounded plan using $\mathcal{V}$ with Q_s (*i.e.,* $\mathsf{covq}(Q_s, Q_1) = \mathsf{true}$). One can verify that this expansion policy assures that Q_s always has a bounded plan since we start with a tautology query $Q_s = Q_\varepsilon$.

Observe the following. (1) Q_s is expanded in case (c) above to propagate $\bar{z}_1$ from $Q_1 \wedge Q_s$ to Q_2 there. More specifically, if sub-query $Q_2(\bar{z}_2)$ of Q does not have a bounded rewriting with $Q_s(\bar{x})$ (*i.e.,* when $\mu_2 = \mathsf{false}$), we may extend $Q_s(\bar{x})$ with $Q_1(\bar{z}_1)$ to make $Q_2(\bar{z}_2)$ bounded when $\mu' = \mathsf{true}$. (2) We also restrict the size $|Q_2|$ for case (c) to ensure both functions $\mathsf{covq}(\cdot, \cdot)$ and $\mathsf{size}(\cdot, \cdot)$ are in PTIME. Indeed, to compute $\mathsf{covq}(Q_s, Q)$, we need to expand Q_s with various conjuncts of Q_2 if Q_2 is also a conjunction, by applying case (4b) or (4c) alternatively. For example, when Q_2 is $Q_{21} \wedge Q_{22}$, to compute $\mathsf{covq}(Q_s, Q)$ via cases (4b) and (4c), we may need to compute $\mathsf{covq}(Q_s, Q_{22})$, $\mathsf{covq}(Q_s \wedge Q_1, Q_{22})$, $\mathsf{covq}(Q_s \wedge Q_{21}, Q_{22})$ and $\mathsf{covq}(Q_s \wedge Q_1 \wedge Q_{21}, Q_{22})$. In the worst case, we may test $2^{|Q_2|}$ many difference cases. Hence we restrict the size of Q_2 by a predefined constant K, to bound the number of expansions of Q_s when computing $\mathsf{covq}(Q_\varepsilon, Q)$ and ensure that it is in PTIME. We remark that this restriction has no impact on the expressive power of topped queries up to equivalence, even when $K = 1$ (see the proof of Theorem 5.1 in the electronic appendix for more details).

(5) $Q(\bar{z})$ is $Q_1(\bar{z}_1) \vee Q_2(\bar{z}_2)$. If $\bar{z}_1 \neq \bar{z}$ or $\bar{z}_2 \neq \bar{z}$, we let $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{false}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$. Otherwise, let $\mu_i = \mathsf{covq}(Q_s(\bar{x}), Q_i(\bar{z}))$ and $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}))$ for $i \in \{1, 2\}$. Define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mu_1 \wedge \mu_2$, and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $s_1 + s_2 + 1$ if $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$ otherwise.

Intuitively, if Q_1 and Q_2 have bounded plans ξ_1 and ξ_2, respectively, then $Q(\bar{z})$ has a bounded plan $(T_1 = \xi_1, T_2 = \xi_2, T_1 \cup T_2)$, of size $|\xi_1| + |\xi_2| + 1$.

Note that when Q_1 and Q_2 do not share the same free variables $\bar{z}$, $Q_1 \vee Q_2$ can never be topped queries since $\mathsf{covq}(Q_s, Q_1 \vee Q_2) = \mathsf{false}$. This is to ensure that topped queries are safe-range and hence are "safe", *i.e.,* domain-independent (only domain-independent calculus queries are well-defined queries, *i.e.,* queries have determined query answers on every database instance, and have equivalent algebra forms and query plans [Gelder and Topor 1991]). For example, this will exclude "unsafe" queries like $Q(x, y) = \exists w_1, w_2\, R(w_1, x) \vee R(w_2, y)$ from the class of topped queries.

(6) $Q(\bar{z})$ is $Q_1(\bar{z}_1) \wedge \neg Q_2(\bar{z}_2)$. If $\bar{z}_1 \neq \bar{z}$ or $\bar{z}_2 \neq \bar{z}$, we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{false}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = +\infty$. Otherwise, let $\mu_i = \mathsf{covq}(Q_s(\bar{x}), Q_i(\bar{z}))$, $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}))$, $\mu_{12} = \mathsf{covq}(Q_s(\bar{x}), Q_1(\bar{z}) \wedge Q_2(\bar{z}))$, and $s_{12} = \mathsf{size}(Q_s(\bar{x}), Q_1(\bar{z}) \wedge Q_2(\bar{z}))$. Then we define

(a) if $\mu_1 \wedge \mu_2 = \mathsf{true}$, $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s_2 + 1$; otherwise

(b) if $\mu_1 \wedge \mu_{12} = \mathsf{true}$ and $|Q_2| \leqslant K$ for some predefined constant K, $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s_{12} + 1$; otherwise

(c) we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z})$ = false and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$.

It is case (6) that captures how sub-query Q_4 of Q_3 is propagated to $\exists w\, R(z, w)$ in Example 5.3. When $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true, we have one of the following three cases.

(a) When $\mu_1 = \mu_2$ = true, it is similar to case (5) above.

(b) If $\mu_1 = \mu_{12}$ = true, let ξ_1 and ξ_{12} be the plans for $Q_1(\bar{z})$ and $Q_1(\bar{z}) \wedge Q_2(\bar{z})$, respectively, with $Q_s(\bar{x})$. Since $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z})$ = $Q_1(\bar{z}) \wedge \neg(Q_1(\bar{z}) \wedge Q_2(\bar{z}))$, $Q(\bar{z})$ has bounded plan $(T_1 = \xi_1, T_2 = \xi_{12}, T_3 = T_1 - T_2)$, of size $|\xi_1| + |\xi_{12}| + 1$. For the same reason as the one given in case 4(c) above, we also require $|Q_2| \leqslant K$ here.

(c) Otherwise, $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = false, and thus $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$, *i.e.,* Q has no bounded rewriting.

For the same reason as (5), we only allow cases when Q_1 and Q_2 have the same free variables to be topped queries, to ensure that every topped query is safe-range.

(7) $Q(\bar{z})$ is $\exists \bar{w}\, Q'(\bar{w}, \bar{z})$ ($\bar{w}$ is possibly empty). Let μ' = $\mathsf{covq}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z}))$ and s' = $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z}))$. Then we consider the following three cases:

(a) if Q' is $R(\bar{w}, \bar{z})$ and there exists access constraint $R(\emptyset \to Z, N) \in \mathcal{A}$, then we define $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = 1;

(b) if Q' is $R(\bar{w}, \bar{z})$, $R(X \to Z', N) \in \mathcal{A}$, $X \cup Z' = Z$ and if $Q_s(\bar{x})$ has *bounded output* under $\mathcal{A}$, then $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$=true, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s' + 1$;

(c) otherwise, $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = $\mathsf{covq}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z}))$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z})) + 1$ if $\mathsf{covq}(Q_s(\bar{x}), Q(\bar{z}))$ = true, and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ = $+\infty$ otherwise.

Observe that $Q_s(\bar{x})$ may not have bounded output even when it has a bounded rewriting. Therefore, in case (b) above we have to ensure that $Q_s(\bar{x})$ has bounded output in order to propagate $\bar{x}$-value from $Q_s(\bar{x})$ to $R(\bar{z})$, for a fetch operation to use the $\bar{x}$-value.

Moreover, observe the following about case (7).

(a) When $Q(\bar{z})$ is a projection of a relation atom $\exists \bar{w}\, R(\bar{w}, \bar{z})$, if it is covered by $R(\emptyset \to Z, N)$ in $\mathcal{A}$, then $\mathsf{fetch}(\emptyset, R, Z)$ is an 1-bounded plan for $Q(\bar{z})$.

(b) If $Q(\bar{z})$ is $\exists \bar{w}\, R(\bar{w}, \bar{z})$ and is covered by $R(X \to Z', N)$, and $Q_s(\bar{x})$ has bounded output, then $Q_s \wedge Q$ has a plan $(T_1 = \xi_s, T_2 = \mathsf{fetch}(X \in T_1, R, Z'))$, where ξ_s is the plan for Q_s. And this is also a plan for Q with Q_s.

(c) Otherwise, $Q(\bar{z})$ has a bounded plan if $Q'(\bar{w}, \bar{z})$ has one. Let ξ' be the plan for Q' with Q_s. Then $(T_1 = \xi', T_2 = \pi_Z(T_1))$ of size $|\xi'|$ + 1 is a plan for $Q(\bar{z})$ with $Q_s(\bar{x})$.

Example 5.4. We next show that q_3 of Example 5.3 is topped by $(\mathcal{R}_1, \mathcal{A}_2, \mathcal{V}_3, 13)$. Denote the sub-queries of q_3 as follows:

q_1 = $V_3(x, y) \wedge (x = 1)$, q_2 = $\exists x\, q_1$, q'_2 = $q_2 \wedge R(y, z)$ (thus $q_4 = \exists y\, q'_2$), q'_4 = $\exists w\, R(z, w)$.

Then one can easily verify the following:

(a) $\mathsf{covq}(Q_\varepsilon, q_3)$ = $(\mathsf{covq}(Q_\varepsilon, q_4) \wedge \mathsf{covq}(Q_\varepsilon, q'_4)) \vee (\mathsf{covq}(Q_\varepsilon, q_4) \wedge \mathsf{covq}(Q_\varepsilon, q_4 \wedge q'_4))$,
(b) $\mathsf{covq}(Q_\varepsilon, q_4)$=$\mathsf{covq}(Q_\varepsilon, q'_2)$= $(\mathsf{covq}(Q_\varepsilon, q_2) \wedge \mathsf{covq}(Q_\varepsilon, R(y, z)) \vee (\mathsf{covq}(Q_\varepsilon, q_2) \wedge \mathsf{covq}(q_2, R(y, z))$,
(c) $\mathsf{covq}(Q_\varepsilon, q_2)$ = $\mathsf{covq}(Q_\varepsilon, q_1)$ = true,
(d) $\mathsf{covq}(q_2, R(y, z))$ = true (since q_2 has bounded output: $|q_2(\mathcal{D})| \leqslant N$ for any $\mathcal{D} \models \mathcal{A}$),
(e) from these it follows that $\mathsf{covq}(Q_\varepsilon, q_4)$ = true,
(f) $\mathsf{covq}(q_4, q'_4)$ = true (since q_4 has bounded output: $|q_4(\mathcal{D})| \leqslant N^2$ for any $\mathcal{D} \models \mathcal{A}$),
(g) $\mathsf{covq}(Q_\varepsilon, q_4 \wedge q'_4)$ = $(\mathsf{covq}(Q_\varepsilon, q_4) \wedge \mathsf{covq}(Q_\varepsilon, q'_4)) \vee (\mathsf{covq}(Q_\varepsilon, q_4) \wedge \mathsf{covq}(q_4, q'_4))$ = true.

Thus $\mathsf{covq}(Q_\varepsilon, q_3)$ = true. Along the same lines, one can verify that $\mathsf{size}(Q_\varepsilon, q_3) = 13$. Thus q_3 is topped by $(\mathcal{R}_1, \mathcal{A}_2, \mathcal{V}_3, 13)$. □

Proof sketch of Theorem 5.1. Having defined topped queries, we now outline a proof of Theorem 5.1 (we defer the details to the electronic appendix for the lack of space).

(a) Suppose that Q is an FO query with an M-bounded rewriting, *i.e.,* Q has an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$. We show that there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $\xi \equiv_\mathcal{A} Q_\xi$, by induction on M, verifying each step (case) of ξ.

(b) We show that every query Q topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has a $\mathsf{size}(Q_\varepsilon, Q)$-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. The proof needs the following lemma: if $\mathsf{covq}(Q_s, Q)$ = $\mathsf{covq}(Q_\varepsilon, Q_s)$ = true, and if Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan, then $Q_s \wedge Q$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$-bounded plan. This is verified by induction on the structure of Q.

For instance, when $Q(\bar{z})$ is $Q_1(\bar{z}_1) \wedge Q_2(\bar{z}_2)$, $\mathsf{covq}(Q_s, Q(\bar{z}))$ and $\mathsf{covq}(Q_\varepsilon, Q_s)$ are true and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan, we know that $\mathsf{covq}(Q_s, Q_1(\bar{z}_1))$ is also true. By the induction hypothesis we have that $Q_s \wedge Q_1(\bar{z}_1)$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1(\bar{z}_1))$-bounded plan. Moreover, either $\mathsf{covq}(Q_s, Q_2(\bar{z}_2))$ or $\mathsf{covq}(Q_s \wedge Q_1(\bar{z}_1), Q_2(\bar{z}_2))$ is true. In both cases, by the induction hypothesis, $Q_s \wedge Q_1 \wedge Q_2$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1 \wedge Q_2)$-bounded plan.

(c) It takes PTIME in $|\mathcal{R}|$, $|Q|$, $|\mathcal{V}|$, $|\mathcal{A}|$ and M to check whether an FO query is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. Indeed, we show that both $\mathsf{covq}(Q_\varepsilon, Q)$ and $\mathsf{size}(Q_\varepsilon, Q)$ are PTIME functions, which invoke a PTIME oracle to check bounded output for cases (4a) and (7b) of topped queries given above. Moreover, we show that it takes PTIME to generate an M-bounded rewriting using $\mathcal{V}$ for each query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. □

Remark. (a) To prove Theorem 5.1(1), it suffices to use $Q_s = Q_\varepsilon$, which yields a simpler form of effective syntax for bounded rewriting. We allow value propagation in cases (4c) and (6b) in order to cover queries that are commonly used in practice, which, nonetheless, leads to an effective syntax that is a little complicated. (b) The class of topped queries is quite different from the rules for $\bar{x}$-controllability ([Fan et al. 2014]; see Section 7) and the syntactic rules for bounded evaluability of CQ [Fan et al. 2015] and for FO [Cao and Fan 2016], particularly in the use of Q_s to check bounded output of views and the function $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ to ensure the bounded size of query plans.

5.3. Size Bounded Queries

We next define size-bounded queries and prove Theorem 5.2. We remark that there are other forms of effective syntax for FO queries with bounded output. To simplify the discussion, below we present a straightforward one.

Size-bounded queries. An FO query $Q(\bar{x})$ is *size-bounded* under an access schema $\mathcal{A}$ if it is of the following form:

$$Q(\bar{x}) = Q'(\bar{x}) \wedge \forall \bar{x}_1, \ldots, \bar{x}_{K+1} \left(Q'(\bar{x}_1) \wedge \cdots \wedge Q'(\bar{x}_{K+1}) \rightarrow \bigvee_{i,j \in [1,K+1], i \neq j} \bar{x}_i = \bar{x}_j\right),$$

where K is a natural number, and Q' is an FO query.

Intuitively, for any FO query Q', if Q' has output size bounded by K, then the Boolean conjunct $\forall \bar{x}_1, \ldots, \bar{x}_{K+1} \left(Q'(\bar{x}_1) \wedge \cdots \wedge Q'(\bar{x}_{K+1}) \rightarrow \bigvee_{i,j \in [1,K+1], i \neq j} \bar{x}_i = \bar{x}_j\right)$ of Q is true. Hence, $Q = Q'$ and Q also has output bounded by K. When Q' does not have output size bounded by K, the Boolean conjunct is false. Hence Q = false, and Q also has output size bounded by K in this case. The class of size-bounded queries includes all queries of such form, which obviously have bounded output size. Indeed, this is an effective syntax of queries with bounded output, verifying Theorem 5.2, as proved below. Note that we do not fix the number K, *i.e.,* queries with arbitrary natural number K are included in the class of size-bounded queries, as long as K is a natural number.

Proof of Theorem 5.2. This class of size-bounded queries suffices for Theorem 5.2.

(a) Consider an FO query $Q(\bar{x})$ having bounded output under $\mathcal{A}$. By the definition of queries with bounded-output (Section 3.1), there exists a natural number K such that for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}$, then $|Q(\mathcal{D})| \leqslant K$. Construct $Q'(\bar{x})$ from $Q(\bar{x})$ as

$$Q'(\bar{x}) = Q(\bar{x}) \wedge \forall \bar{x}_1, \ldots, \bar{x}_{K+1} \left(Q(\bar{x}_1) \wedge \cdots \wedge Q(\bar{x}_{K+1}) \rightarrow \bigvee_{i,j \in [1,K+1], i \neq j} (\bar{x}_i = \bar{x}_j)\right).$$

Obviously, $Q'(\bar{x})$ is a size-bounded query. Moreover, $Q'(\bar{x}) \equiv_{\mathcal{A}} Q(\bar{x})$, since $Q(\bar{x})$ has output bounded by K, and hence, for any $\mathcal{D} \models \mathcal{A}$, it is easy to see that $Q(\mathcal{D}) = Q'(\mathcal{D})$.

(b) Consider a size-bounded query $Q(\bar{x})$ of the form above. For any $\mathcal{D}$, if $Q'(\mathcal{D})$ contains more than K answer tuples, then $Q(\mathcal{D}) = \emptyset$. Otherwise, $Q(\mathcal{D}) = Q'(\mathcal{D})$ and $Q(\mathcal{D})$ includes at most K tuples. Hence $|Q(\mathcal{D})| \leqslant K$. That is, Q has bounded output.

(c) By the definition of size-bounded queries, it is immediate to syntactically check whether an FO query Q is size-bounded. It takes PTIME in the size $|Q|$ of Q. □

6. BOUNDED $\mathcal{L}_1$-TO-$\mathcal{L}_2$ QUERY REWRITING USING VIEWS

One might be tempted to think that it would be simpler to find a bounded rewriting of a query Q of $\mathcal{L}_1$ in another language $\mathcal{L}_2$ that is more expressive than $\mathcal{L}_1$. In this section, we formalize and study $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting using views.

More specifically, consider query languages $\mathcal{L}_1$ and $\mathcal{L}_2$, where $\mathcal{L}_1 \subseteq \mathcal{L}_2$, *i.e.,* for all queries $Q \in \mathcal{L}_1$, $Q \in \mathcal{L}_2$. We study the problem of *$\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting using views*, denoted by $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$ and stated as follows.

— INPUT: A database schema $\mathcal{R}$, a natural number M (in unary), an access schema $\mathcal{A}$, a query $Q \in \mathcal{L}_1$, and a set $\mathcal{V}$ of $\mathcal{L}_1$-definable views, all defined on $\mathcal{R}$.
— QUESTION: Under $\mathcal{A}$, does Q have an M-bounded rewriting in $\mathcal{L}_2$ using $\mathcal{V}$?

That is, $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$ is to decide whether Q has a query plan ξ such that (a) ξ is in $\mathcal{L}_2$, *i.e.,* $Q_\xi \in \mathcal{L}_2$ for the query Q_ξ expressed by ξ, (b) ξ conforms to $\mathcal{A}$, and (c) the size of ξ is at most M (see Section 2). Observe that $\mathsf{VBRP}(\mathcal{L}_1)$ is a special case of $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$, *i.e.,* $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_1)$, when $\mathcal{L}_1$ and $\mathcal{L}_2$ are required to be the same query language. We thus only need to consider cases when $\mathcal{L}_1 \subsetneq \mathcal{L}_2$, since we have already covered the cases when $\mathcal{L}_1 = \mathcal{L}_2$ in the previous sections. We show that $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$ makes our lives no easier than $\mathsf{VBRP}(\mathcal{L}_1)$. Indeed, its lower bound gets no better than its counterpart given in Theorem 3.1.

THEOREM 6.1. $\mathsf{VBRP}^+(\mathcal{L}_1, \mathcal{L}_2)$ *is* Σ_3^p*-hard*
— *when* $\mathcal{L}_1$ *is* CQ *and* $\mathcal{L}_2$ *is one of* UCQ, $\exists$FO$^+$ *or* FO*;*
— *when* $\mathcal{L}_1$ *is* UCQ *and* $\mathcal{L}_2$ *is* $\exists$FO$^+$ *or* FO*; and*
— *when* $\mathcal{L}_1$ *is* $\exists$FO$^+$ *and* $\mathcal{L}_2$ *is* FO. □

Proof: (1) Observe that $\mathsf{VBRP}^+(\mathsf{CQ}, \mathcal{L}_2)$ is a special case of $\mathsf{VBRP}^+(\mathsf{UCQ}, \mathcal{L}_2)$ and $\mathsf{VBRP}^+(\exists\mathsf{FO}^+, \mathcal{L}_2)$ since CQ $\subseteq$ UCQ and CQ $\subseteq$ $\exists$FO$^+$. Thus it suffices to show that $\mathsf{VBRP}^+(\mathsf{CQ}, \mathcal{L}_2)$ is Σ_3^p-hard when $\mathcal{L}_2$ is UCQ, $\exists$FO$^+$ or FO, from which it follows that VBRP^+(UCQ, $\exists$FO$^+$), VBRP^+(UCQ, FO) and VBRP^+($\exists$FO$^+$, FO) are also Σ_3^p-hard.

We show that $\mathsf{VBRP}^+(\mathsf{CQ}, \mathcal{L}_2)$ is Σ_3^p-hard by reduction from the $\exists^*\forall^*\exists^*$3CNF problem, which is Σ_3^p-complete [Stockmeyer 1976] (see the proof of Theorem 3.1 for

$\exists^*\forall^*\exists^*$3CNF). We adopt the reduction given for $\mathsf{VBRP}(\mathsf{CQ})$ in the proof of Theorem 3.1. That is, given a sentence $\phi = \exists X \forall Y \exists Z\, \psi(X, Y, Z)$, we define the same database schema $\mathcal{R}$, access schema $\mathcal{A}$, CQ Q, and views $\mathcal{V}$ for $\mathsf{VBRP}^+(\mathsf{CQ}, \mathcal{L}_2)$. We also set $M = 6$.

To verify that this makes a reduction for CQ-to-$\mathcal{L}_2$ rewriting, we show the following.

LEMMA 6.2. *For $\mathcal{R}, \mathcal{A}, \mathcal{V}, Q$ and M given in the proof of Theorem 3.1, Q has an M-bounded rewriting in $\mathcal{L}_2$ using $\mathcal{V}$ under $\mathcal{A}$ if and only if Q has an M-bounded rewriting in* CQ *using $\mathcal{V}$ under $\mathcal{A}$, where $\mathcal{L}_2$ ranges over* UCQ, $\exists\mathsf{FO}^+$*and* FO. □

This suffices. For if it holds, the problem for deciding whether the query Q has an M-bounded rewriting in $\mathcal{L}_2$ is equivalent to deciding whether Q has an M-bounded rewriting in CQ. Then the construction given in the proof of Theorem 3.1 is a reduction from the $\exists^*\forall^*\exists^*$3CNF problem to the latter problem. Hence $\mathsf{VBRP}^+(\mathsf{CQ}, \mathcal{L}_2)$ is Σ_3^p-hard.

Proof: We now prove Lemma 6.2. Obviously, if Q has an M-bounded rewriting in CQ, then Q has an M-bounded rewriting in $\mathcal{L}_2$. Conversely, assume by contradiction that Q has an M-bounded rewriting ξ (*i.e.,* query plan) in $\mathcal{L}_2$ but does not have an M-bounded rewriting in CQ, when $\mathcal{L}_2$ is UCQ, $\exists\mathsf{FO}^+$or FO. We show that it is impossible that ξ includes either union $\cup$ or set difference $\setminus$ operations, contradicting the assumption.

We start with the following observation. Since ξ is a query plan for Q, we have that $\xi \equiv_{\mathcal{A}} Q$ (see Section 2). Then ξ must contain the following operations:

- either a set union operation $\cup$ or a set difference $\setminus$ operation as assumed;
- the view V; by the definition of Q and V, for ξ to cover all relations needed to answer Q, ξ has to use V given the constraint imposed by bound $M = 6$;
- a fetch operation for R_o, because V does not contain relation R_o needed by Q;
- a constant selection $\sigma_{Y=1}$ on the relation atom $R_o(k, 1)$ in Q; and
- a projection of the form $\pi_\emptyset(S)$ for a relation S; this is because Q is a Boolean query, while the view V, the constant selection, and the fetch operation are not.

These five operations must appear in ξ. Given $M = 6$, an M-bounded plan ξ can contain at most one additional operation. We next show that this is impossible for ξ.

Consider the fetch operation in ξ: $\mathsf{fetch}(I \in S_j, R_o, Y)$, where S_j is the result of a previous operation in ξ, computed by a "query plan" ξ_{S_j} (see Section 2). To retrieve data from R_o, S_j cannot be empty. We show that ξ_{S_j} needs at least two more operations that are not among the five operations described above. That is, ξ needs at least 7 operations, exceeding the bound $M = 6$ and hence leading to a contradiction.

More specifically, consider the following cases of ξ_{S_j} (see Section 2 for query plans).

(a) If ξ_{S_j} is a constant c, it does not help us answer Q because the value k used in the atom $R_o(k, 1)$ in Q is arbitrary, and may not match the constant c.

(b) Now suppose that ξ_{S_j} is defined in terms of other five operations allowed in a query plan (see Section 2). We distinguish the following two cases:

- Assume that ξ_{S_j} does not have V as a descendant. Then as only one additional operation is allowed, $\mathsf{fetch}(\emptyset, R_{01}, A)$ is the only possible plan of size 1 that does not use V. Similar to case (a), one can verify that it does not help us answer Q.
- If ξ_{S_j} takes V as a descendant, then ξ_{S_j} also needs a projection π_A so that S_j is unary. Recall that the access constraint on R_o takes the first attribute of R_o as input, while V is not unary. Meanwhile, as argued in the proof of Theorem 3.1, the only way that V can be used in a query plan that conforms to $\mathcal{A}$ is when it occurs as $\sigma_{X=\mu_X^0}(V)$, *i.e.,* when all its $\bar{x}$-values are fixed Boolean values by means of a truth-assignment μ_X^0. Hence ξ_{S_j} also needs an additional selection operation on V. Therefore, when ξ_{S_j} has V as a descendant, ξ_{S_j} needs at least two more operations: one projection π_A and one selection on V.

Putting these together, we can conclude that if ξ is a 6-bounded query plan for Q, then ξ_{S_j} includes at least two operations, a contradiction to the size of ξ. Hence if ξ is a 6-bounded query plan for Q using $\mathcal{V}$ under $\mathcal{A}$, then ξ must be in CQ. □

One may wonder whether UCQ is "complete" for CQ-to-FO bounded rewriting using views. That is, for any natural number M, any set $\mathcal{V}$ of CQ views, and any CQ Q, if Q has an M-bounded rewriting in FO using $\mathcal{V}$, then Q has an M-bounded rewriting in UCQ using $\mathcal{V}$. Below we show that this is not the case, by giving a counterexample.

Example 6.3. Consider a database schema $\mathcal{R}$ consisting of six relations: $R(X, Y, Z)$, $T(X, Y)$, $K_1(X, Y)$, $K_2(X, Y)$, $K_3(X, Y)$, $K_4(X, Y)$; an access schema $\mathcal{A}$ consisting of five constraints: $T(X \to Y, 3)$, $K_1(X \to Y, 1)$, $K_2(X \to Y, 1)$, $K_3(X \to Y, 1)$, $K_4(X \to Y, 1)$; and a Boolean CQ Q defined as follows:

$$Q() = \exists x, y, z_1, z_2 \ \big(R(x, y, z_1) \wedge R(x, y, z_2) \wedge Q'(y, z_1, y, z_2)\big), \text{ where}$$
$$Q'(x_1, x_2, x_3, x_4) = \exists y' \ \big(T(y', x_1) \wedge T(y', x_2) \wedge T(y', x_3) \wedge T(y', x_4) \wedge K_1(x_1, 1) \wedge K_1(x_2, 2) \wedge K_2(x_3, 1) \wedge K_2(x_4, 2) \wedge K_3(x_1, 1) \wedge K_3(x_4, 2) \wedge K_4(x_2, 1) \wedge K_4(x_3, 2)\big).$$

We use a set $\mathcal{V}$ of three Boolean CQ views defined as follows:

$$V_1() = \exists x, y, z_1, z_2 \ \big(R(x, z_1, y) \wedge R(x, z_2, y) \wedge Q'(z_1, y, z_2, y)\big);$$
$$V_2() = \exists x, y_1, z_1, z_2, x_1, y_2, z_3, z_4 \ \big(R(x, y_1, z_1) \wedge R(x, y_1, z_2) \wedge Q'(y_1, z_1, y_1, z_2)\big) \wedge \big(R(x_1, z_3, y_2) \wedge R(x_1, z_4, y_2) \wedge Q'(z_3, y_2, z_4, y_2)\big);$$
$$V_3() = \exists x, y_1, y_2, z_1, z_2 \ \big(R(x, y_1, z_1) \wedge R(x, y_2, z_2) \wedge Q'(y_1, z_1, y_2, z_2)\big).$$

One can verify that $Q \not\sqsubseteq_{\mathcal{A}} V_1$, $V_1 \not\sqsubseteq_{\mathcal{A}} Q$, $V_2 \equiv_{\mathcal{A}} (V_1 \wedge Q)$ and $V_3 \equiv_{\mathcal{A}} (V_1 \cup Q)$. These can be verified by observing the following properties: $\mathcal{A}$ and Q ensure that for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}$, $Q'(\mathcal{D}) \neq \emptyset$, and suppose that ν is a valuation of the variables of Q' to values in $\mathcal{D}$, then we have that either $\nu(x_1) = \nu(x_3)$ or $\nu(x_2) = \nu(x_4)$. Indeed, by $T(X \to Y, 3) \in \mathcal{A}$, one can verify that one of the following holds: $\nu(x_1) = \nu(x_2)$, $\nu(x_1) = \nu(x_3)$, $\nu(x_1) = \nu(x_4)$, $\nu(x_2) = \nu(x_3)$, $\nu(x_2) = \nu(x_4)$, or $\nu(x_3) = \nu(x_4)$. However, by $K_1(X \to Y, 1) \in \mathcal{A}$, we have that $\nu(x_1) \neq \nu(x_2)$. Similarly, from $K_2(X \to Y, 1)$, $K_3(X \to Y, 1)$ and $K_4(X \to Y, 1)$ in $\mathcal{A}$, one can conclude that $\nu(x_3) \neq \nu(x_4)$, $\nu(x_1) \neq \nu(x_4)$ and $\nu(x_2) \neq \nu(x_3)$. From these it follows that either $\nu(x_1) = \nu(x_3)$ or $\nu(x_2) = \nu(x_4)$. By this property, we can verify $V_3 \equiv_{\mathcal{A}} (V_1 \cup Q)$ as follows. From the definition of V_1, Q and V_3, it is easy to see that $(V_1 \cup Q) \sqsubseteq_{\mathcal{A}} V_3$. It remains to show $V_3 \sqsubseteq_{\mathcal{A}} (V_1 \cup Q)$. For any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}$, $V_3(\mathcal{D}) \neq \emptyset$, and suppose that ν is a valuation of the variables of V_3 to values in $\mathcal{D}$, then by the property above we have that either $\nu(y_1) = \nu(y_2)$ or $\nu(z_1) = \nu(z_2)$. If $\nu(y_1) = \nu(y_2)$, we can construct the following valuation ν_1 of the variables of Q to values in $\mathcal{D}$: $\nu_1(x) = \nu(x)$, $\nu_1(y) = \nu(y_1)$, $\nu_1(z_1) = \nu(z_1)$, $\nu_1(z_2) = \nu(z_2)$, $\nu_1(y') = \nu(y')$, and $\nu_1(x_i) = \nu(x_i)$ $(i \in [1, 4])$. Thus $Q(\mathcal{D}) \neq \emptyset$. If $\nu(z_1) = \nu(z_2)$, we can similarly show that $V_1(\mathcal{D}) \neq \emptyset$. Putting all these together, we have that $V_3 \sqsubseteq_{\mathcal{A}} (V_1 \cup Q)$, and then $V_3 \equiv_{\mathcal{A}} (V_1 \cup Q)$. The other relations can be verified in a similar manner.

We show the following: using $\mathcal{V}$ under $\mathcal{A}$, query Q (a) has a 5-bounded rewriting in FO, but (b) it does not have a 5-bounded rewriting in UCQ. Here we set $M = 5$.

Rewriting in FO. We show that Q has a rewriting $Q_{\mathsf{FO}}() = (V_3 \setminus V_1) \cup V_2$ in FO. Obviously, $Q_{\mathsf{FO}}()$ has a 5-bounded query plan. It thus suffices to show that $Q_{\mathsf{FO}} \equiv_{\mathcal{A}} Q$.

We first show that $Q \sqsubseteq_{\mathcal{A}} Q_{\mathsf{FO}}$. Let T_Q be the tableau representation of Q. It is easy to verify that $T_Q \models \mathcal{A}$ and $Q(T_Q) = \mathsf{true}$. Moreover, by $Q \not\sqsubseteq_{\mathcal{A}} V_1$, we have that $V_1(T_Q) = \mathsf{false}$. By $Q(T_Q) = \mathsf{true}$, $V_2 \equiv_{\mathcal{A}} (V_1 \wedge Q)$ and $V_3 \equiv_{\mathcal{A}} (V_1 \cup Q)$, we have that $V_2(T_Q) = \mathsf{false}$ and $V_3(T_Q) = \mathsf{true}$. Thus $Q_{\mathsf{FO}}(T_Q) = \mathsf{true}$. This actually shows that for any instance $\mathcal{D}$ of $\mathcal{R}$, if $\mathcal{D} \models \mathcal{A}$ and $Q(\mathcal{D}) = \mathsf{true} \neq \emptyset$, then $Q_{\mathsf{FO}}(\mathcal{D}) = \mathsf{true}$. Thus $Q \sqsubseteq_{\mathcal{A}} Q_{\mathsf{FO}}$.

We next show that $Q_{\mathsf{FO}} \sqsubseteq_{\mathcal{A}} Q$. For any instance $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$ such that $Q_{\mathsf{FO}}(\mathcal{D}) = \mathsf{true}$, we need to show that $Q(\mathcal{D}) = \mathsf{true}$, by considering the following two cases:

- If $V_2(\mathcal{D}) = \mathsf{true}$, then $Q(\mathcal{D}) = \mathsf{true}$ since $V_2 \equiv_{\mathcal{A}} (V_1 \wedge Q)$.
- If $V_2(\mathcal{D}) = \mathsf{false}$, then $(V_3 \setminus V_1)(\mathcal{D}) = \mathsf{true}$ since $Q_{\mathsf{FO}}(\mathcal{D}) = \mathsf{true}$, *i.e.,* $V_3(\mathcal{D}) = \mathsf{true}$ and $V_1(\mathcal{D}) = \mathsf{false}$. Moreover, from $V_3 \equiv_{\mathcal{A}} (V_1 \cup Q)$, $V_3(\mathcal{D}) = \mathsf{true}$ and $V_1(\mathcal{D}) = \mathsf{false}$, we can deduce that $Q(\mathcal{D}) = \mathsf{true}$.

Rewriting in UCQ. In contrast, Q has *no* 5-bounded rewriting in UCQ using $\mathcal{V}$ under $\mathcal{A}$. We show that all possible 5-bounded rewritings of Q in UCQ cannot use fetch operations.

Indeed, since V_1, V_2, and V_3 are Boolean queries, we cannot use the output of these views or constants to fetch data of T, K_1, K_2, K_3 and K_4. In addition, observe that any rewriting of Q cannot impose selection and projection operations on the Boolean views. Moreover, for atoms in Q, values in the first attributes are not fixed. If any rewriting Q_ξ uses a constant c_1 to fetch values, by the definition of $\mathcal{A}$, we know that there exists an atom in Q_ξ such that c_1 appears in its first attribute. Then we can construct an instance $\mathcal{D}$ such that $Q(\mathcal{D}) \neq \emptyset$, and the first attributes of all instances do not contain the constant c_1. However, we have that $Q_\xi(\mathcal{D}) = \emptyset$, which is a contradiction. These leave us a small number of possible 5-bounded rewritings of Q in UCQ. Examining these possible rewritings will reveal that none of them makes a 5-bounded rewriting of Q using $\mathcal{V}$ under $\mathcal{A}$. As an example, consider a possible rewriting $Q_1 = (V_1 \cup V_2) \times V_1$. One can easily verify that $Q \not\equiv_{\mathcal{A}} Q_1$. To see this, it suffices to consider the tableau representation of V_1, denoted by T_1. It is easy to verify that $T_1 \models \mathcal{A}$ and $V_1(T_1) = \mathsf{true}$. Then by $Q_1 = (V_1 \cup V_2) \times V_1$ and $V_1(T_1) = \mathsf{true}$, we have that $Q_1(T_1) = \mathsf{true}$. However, from $V_1 \not\sqsubseteq_{\mathcal{A}} Q$ it follows that $Q(T_1) = \mathsf{false}$. Hence $Q_1 \not\equiv_{\mathcal{A}} Q$. □

7. RELATED WORK

This paper extends its conference version [Anonymous b] by including the detailed proofs of all results, which were not given in [Anonymous b]. Some of the proofs are nontrivial and are interesting in their own right. In addition, we study $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting (Section 6), a topic not considered in [Anonymous b].

We classify the other related work as follows.

Scale independence. The idea of scale independence originated from [Armbrust et al. 2009], which is to execute the workload in an application by doing a bounded amount of work, regardless of the size of datasets used. The idea was incorporated into PIQL [Armbrust et al. 2011], an extension of SQL by allowing users to specify bounds on the amount of data accessed. As pointed out by [Armbrust et al. 2013], to make complex PIQL queries scale independent, precomputed views and query rewriting using views should be employed. Techniques for view selection, indexing and incremental maintenance were also developed there.

The idea of scale independence was formalized in [Fan et al. 2014]. A query Q is defined to be *scale independent* in a dataset $\mathcal{D}$ *w.r.t.* a bound Θ if there exists a fraction $D_Q \subseteq D$ such that $Q(\mathcal{D}) = Q(D_Q)$ and $|D_Q| \leqslant \Theta$. Access constraints, a notion of $\bar{x}$-controllability (the bounded evaluability of a query $Q(\bar{x}, \bar{y})$ when provided with a value of $\bar{x}$), and a set of rules were also introduced in [Fan et al. 2014], to deduce dependencies on attributes needed for computing $Q(\mathcal{D})$; these yield a sufficient condition to determine the scale independence of FO queries when variables $\bar{x}$ are instantiated. In addition, [Fan et al. 2014] considered the problem of deciding whether for all instances $\mathcal{D}$ of a relational schema, we can compute $Q(\mathcal{D})$ by accessing cached views and at most Θ tuples, *in the absence of* access constraints. It was shown there that the problem is NP-complete for CQ, and undecidable for FO. The notion of $\bar{x}$-controllability was extended to views, giving two simple sufficient conditions to decide the scale independence of query rewriting under access constraints.

This work differs from the prior work in the following. (a) We formalize bounded rewriting using views in terms of query plans subject to a bound M determined by available resources. This formulation is quite different from the notion of $\bar{x}$-controllability [Fan et al. 2014]. (b) We incorporate access constraints to make the notion more practical; without such constraints, few queries have a bounded rewriting. Under the constraints, however, the analysis of bounded rewriting is more intriguing. For instance, VBRP(CQ) is Σ_3^p-complete, in contrast to NP-complete [Fan et al. 2014]. (c) We provide an effective syntax for FO queries with a bounded rewriting using views under access constraints, a sufficient and necessary condition. In contrast, the conditions of [Fan et al. 2014] via $\bar{x}$-controllability are sufficient but not necessary. Moreover, the rules of [Fan et al. 2014] do not distinguish whether views are used to just validate data or to fetch data from underlying datasets; this is critical for VBRP, and demands the bounded output analysis of views. Effective syntax, VBRP and VBRP$^+$ were not studied in [Armbrust et al. 2009; Armbrust et al. 2011; Armbrust et al. 2013].

Bounded evaluability. The notion of bounded evaluability was proposed in [Fan et al. 2015], based on a form of query plans that conform to access constraints. The problem for deciding whether a query is boundedly evaluable under access constraints is decidable but EXPSPACE-hard for CQ, and is undecidable for FO [Fan et al. 2015]. A notion of effective boundedness was studied for CQ [Cao et al. 2014], based on a *restricted form* of query plans that conduct all data fetching before any relational operations start. It was shown [Cao et al. 2014] that effective boundedness is in PTIME for CQ. It was also studied for graph pattern queries via simulation and subgraph isomorphism [Cao et al. 2015], which are quite different from relational queries.

Bounded rewriting is more challenging than bounded evaluability. (a) With views comes the need for reasoning about their output size $|\mathcal{V}(\mathcal{D})|$ (Section 3). (b) We adopt query plans in a form of query trees as commonly used in database systems, and allow users to specify a bound on the size of the plans based on their available resources (Section 2). In contrast, [Fan et al. 2015] considers query plans that are a sequence of relational and data fetching operations, of length possibly exponential in the sizes of queries and constraints. After experimenting with real-life data, we find that the plans of [Fan et al. 2015] are not very realistic, and worse yet, their CQ plans may actually encode non-recursive datalog queries without union, which yield exponential-size queries when expressed in CQ. It is because of the different notions of query plans adopted in this work and [Fan et al. 2015] that VBRP is Σ_3^p-complete for CQ, while bounded evaluability is EXPSPACE-hard [Fan et al. 2015].

Effective syntax. There has been a host of work on effective syntax (*e.g.,* [Gelder and Topor 1991; Stolboushkin and Taitslin 1995; Ullman 1982]), which started decades ago to characterize safe relational queries up to equivalence. For bounded query evaluation, an effective syntax was proposed for CQ [Fan et al. 2015], and another one for FO [Cao and Fan 2016]. In contrast, this work develops an effective syntax for bounded rewriting of FO queries using views under access constraints (Section 5). Such a syntax has not been studied before, and is quite different from their counterparts for bounded evaluability. (a) It is in PTIME to check whether an FO query is topped for rewriting, while for bounded evaluability, the syntactic condition of [Fan et al. 2015] is in PTIME to check for CQ, but Π_2^p-complete for UCQ, and is not defined for FO. (b) Effective syntax for query rewriting is more intriguing than its counterpart for bounded evaluability [Cao and Fan 2016]. As remarked earlier, we have to reason about the size $|\mathcal{V}(\mathcal{D})|$ of cached views. It is further complicated by user-imposed bound on the size of query plans, which was not considered in [Cao and Fan 2016]. (c) The class of effectively bounded queries of [Cao et al. 2014] does not make an effective syntax: not every boundedly evaluable CQ is necessarily equivalent to an effectively bounded CQ.

Query rewriting using views. Query rewriting using views has been extensively studied (*e.g.,* [Levy et al. 1995; Afrati 2011; Afrati et al. 2007; Rajaraman et al. 1995; Cohen et al. 1999; Nash et al. 2010]; see [Lenzerini 2002; Halevy 2001] for surveys). In contrast to conventional query rewriting using views, bounded rewriting requires controlled access to the underlying dataset $\mathcal{D}$ under access schema, in addition to cached $\mathcal{V}(\mathcal{D})$ (Section 2). This makes the analysis more challenging. For instance, it is Σ_3^p-complete to decide whether there exists a bounded rewriting for CQ with CQ views, as opposed to NP-complete in the conventional setting [Levy et al. 1995].

Related to $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting (Section 6) is the study of view determinacy (*e.g.,* [Nash et al. 2010; Gogacz and Marcinkowski 2016]), which studies complete rewriting languages. A language $\mathcal{L}$ is *complete* for $\mathcal{L}_1$-to-$\mathcal{L}_2$ rewritings if $\mathcal{L}$ can be used to rewrite a query $Q \in \mathcal{L}_1$ using views $\mathcal{V}$ in $\mathcal{L}_2$ whenever $\mathcal{V}$ determines Q [Nash et al. 2010]. As remarked above, we adopt a different semantics of query rewriting using views, by allowing controlled access to the underlying data under access schema, Moreover, we focus on VBRP^+ instead of complete languages. The results of view determinacy do not carry over to $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting and vice versa.

Access patterns. Related to the work is also query answering under access patterns, which require a relation to be only accessed by providing certain combinations of attributes [Rajaraman et al. 1995; Li 2003; Deutsch et al. 2007; Nash and Ludäscher 2004; Benedikt et al. 2016b; Calì and Martinenghi 2008] (see [Benedikt et al. 2016a] for a survey). Query rewriting using views under access patterns has been studied for CQ [Rajaraman et al. 1995], and for UCQ and $\mathsf{UCQ}^\neg$ (with negated relation atoms) under fixed views and integrity constraints [Deutsch et al. 2007]. This work differs from the prior work in the following. (a) Unlike access patterns, access constraints impose cardinality constraints and controlled data accesses via indices. (b) Moreover, in an access constraint $R(X \to Y, N)$, $X \cup Y$ may account for a small set of the attributes of R, while an access pattern has to cover all the attributes of R. As a result, we can fetch partial tuples from the underlying dataset via an access constraint, as opposed to access patterns that are to fetch entire tuples. This complicates the proofs of bounded rewriting. (c) Bounded rewriting allows access to the underlying data with controlled I/O, which is prohibited in [Rajaraman et al. 1995; Deutsch et al. 2007]. As an evidence of the difference, bounded CQ rewriting using fixed views is C_{2k+1}^p-complete under fixed access constraints (Section 3), as opposed to NP-complete for rewriting using fixed views under access patterns [Deutsch et al. 2007; Li 2003]. (d) To the best of our knowledge, no prior work has studied effective syntax for bounded FO rewriting.

8. CONCLUSION

We have formalized bounded query rewriting using views under access constraints, studied the bounded rewriting problem $\mathsf{VBRP}(\mathcal{L})$ when $\mathcal{L}$ is ACQ, CQ, UCQ, $\exists\mathsf{FO}^+$or FO, and established their upper and lower bounds, all matching, when $M, \mathcal{R}, \mathcal{A}$ and $\mathcal{V}$ are fixed or not. The main complexity results are summarized in Table I, annotated with their corresponding theorems. We have also provided an effective syntax for FO queries with a bounded rewriting, along with an effective syntax for FO queries with bounded output. Moreover, we have shown that bounded query rewriting does not get simpler when we allow a query in $\mathcal{L}$ to be rewritten into a query in another language $\mathcal{L}'$.

One topic for future work is to study bounded rewriting when we allow the amount of data accessed from the underlying dataset $\mathcal{D}$ to be an α-fraction of $\mathcal{D}$, for a small "resource ratio" α in the range of $[0, 1]$, rather than to be bounded by a constant. Intuitively, α indicates the amount of data we can afford to access under our resource budget. Similarly, we may allow M to be a function of resources and workload, instead

Table I. Complexity of $\mathsf{VBRP}(\mathcal{L})$

Queries	Complexity	Condition
FO	undecidable (Th 3.1)	
CQ, UCQ, $\exists$FO$^+$	Σ_3^p-complete (Th 3.1)	
CQ, UCQ, $\exists$FO$^+$	Σ_3^p-complete (Cor 3.10)	fixed $\mathcal{R}, \mathcal{A}, M$
CQ	NP-complete (Prop 4.5)	only FDs in $\mathcal{A}$
Fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$ for the following		
FO	undecidable (Cor 3.9)	
CQ, UCQ, $\exists$FO$^+$	C_{2k+1}^p-complete (Th 3.11)	
CQ	NP-complete (Prop 4.5)	only FDs in $\mathcal{A}$
ACQ	coNP (Th 4.2)	
ACQ	coNP-complete (Th 4.1)	restricted $\mathcal{A}$
ACQ	PTIME (Cor 4.4)	only FDs in $\mathcal{A}$

of a constant. Another topic is to study bounded view maintenance, to incrementally maintain $\mathcal{V}(\mathcal{D})$ by accessing a bounded amount of data in $\mathcal{D}$, in response to changes to $\mathcal{D}$. The third topic is to study top-k (diversified) query rewriting using views, which is to find top-k answers that differ sufficiently from each other [Deng and Fan 2014], by accessing cached views and a bounded amount of underlying data.

A fourth topic is to study approximate query answering. Given a possibly big dataset $\mathcal{D}$, a query Q and a resource ratio $\alpha \in [0, 1]$, it is to compute approximate answers $Q(D_Q)$ to Q in $\mathcal{D}$ by (a) accessing a bounded D_Q such that $|D_Q| \leqslant \alpha|\mathcal{D}|$, and (b) with accuracy above a deterministic bound η, *i.e.,* for any approximate answer $s \in Q(D_Q)$, there exists an exact answer $t \in Q(\mathcal{D})$ such that the distance between s and t is at most η, and conversely, for any $t \in Q(\mathcal{D})$, there exists $s \in Q(D_Q)$ such that s "covers" t with distance at most η. Preliminary work in this direction has been reported in [Cao and Fan 2017]. We aim to extend the approximation framework by incorporating bounded query rewriting.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

AFRATI, F. N. 2011. Determinacy and query rewriting for conjunctive queries and views. *TCS 412,* 11, 1005–1021.

AFRATI, F. N., LI, C., AND ULLMAN, J. D. 2007. Using views to generate efficient evaluation plans for queries. *JCSS 73,* 5, 703–724.

AHO, A. V., SAGIV, Y., AND ULLMAN, J. D. 1979. Equivalences among relational expressions. *SIAM Journal on Computing 8,* 2, 218–246.

ANONYMOUS A. details omitted due to double-blind reviewing.

ANONYMOUS B. details omitted due to double-blind reviewing.

ARMBRUST, M., CURTIS, K., KRASKA, T., FOX, A., FRANKLIN, M. J., AND PATTERSON, D. A. 2011. PIQL: Success-tolerant query processing in the cloud. *PVLDB 5,* 3.

ARMBRUST, M., FOX, A., PATTERSON, D. A., LANHAM, N., TRUSHKOWSKY, B., TRUTNA, J., AND OH, H. 2009. SCADS: Scale-independent storage for social computing applications. In *CIDR*.

ARMBRUST, M., LIANG, E., KRASKA, T., FOX, A., FRANKLIN, M. J., AND PATTERSON, D. 2013. Generalized scale independence through incremental precomputation. In *SIGMOD*.

BENEDIKT, M., LEBLAY, J., TEN CATE, B., AND TSAMOURA, E. 2016a. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

BENEDIKT, M., TEN CATE, B., AND TSAMOURA, E. 2016b. Generating plans from proofs. *ACM Trans. on Database Systems 40,* 4.

CALÌ, A. AND MARTINENGHI, D. 2008. Querying data under access limitations. In *ICDE*.

CAO, Y. AND FAN, W. 2016. An effective syntax for bounded relational queries. In *SIGMOD*.

CAO, Y. AND FAN, W. 2017. Data driven approximation with bounded resources. *PVLDB 10,* 9, 973–984.

CAO, Y., FAN, W., HUAI, J., AND HUANG, R. 2015. Making pattern queries bounded in big graphs. In *ICDE*.

CAO, Y., FAN, W., AND YU, W. 2014. Bounded conjunctive queries. *PVLDB*.

CHANDRA, A. K. AND MERLIN, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *STOC*.

CHEKURI, C. AND RAJARAMAN, A. 2000. Conjunctive query containment revisited. *TCS 239,* 2, 211–229.

COHEN, S., NUTT, W., AND SEREBRENIK, A. 1999. Rewriting aggregate queries using views. In *PODS*.

DENG, T. AND FAN, W. 2014. On the complexity of query result diversification. *ACM Trans. Database Syst. 39,* 2, 15:1–15:46.

DEUTSCH, A., LUDÄSCHER, B., AND NASH, A. 2007. Rewriting queries using views with access patterns under integrity constraints. *TCS 371,* 3, 200–226.

DI PAOLA, R. A. 1969. The recursive unsolvability of the decision problem for the class of definite formulas. *J. ACM 16,* 2, 324–327.

FACEBOOK. 2013. Introducing Graph Search. *https://en-gb.facebook.com/about/graphsearch*.

FACEBOOK. 2014. Constraints on the number of photos, friends and tags. *https://www.facebook.com/help*.

FAN, W., GEERTS, F., CAO, Y., DENG, T., AND LU, P. 2015. Querying big data by accessing small data. In *PODS*.

FAN, W., GEERTS, F., AND LIBKIN, L. 2014. On scale independence for querying big data. In *PODS*.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

GELDER, A. V. AND TOPOR, R. W. 1991. Safety and translation of relational calculus queries. *ACM Trans. on Database Systems 16,* 2, 235–278.

GOGACZ, T. AND MARCINKOWSKI, J. 2016. Red spider meets a rainworm: Conjunctive query finite determinacy is undecidable. In *PODS*.

GOTTLOB, G., LEONE, N., AND SCARCELLO, F. 1999. Hypertree decompositions and tractable queries. In *PODS*.

GRAHAM, M. H. 1979. On the universal relation. Tech. rep., University of Toronto.

GRUJIC, I., BOGDANOVIC-DINIC, S., AND STOIMENOV, L. 2014. Collecting and analyzing data from e-government Facebook pages. In *ICT Innovations*.

HALEVY, A. Y. 2001. Answering queries using views: A survey. *VLDB J. 10,* 4.

HUHTALA, Y., KÄRKK AINEN, J., PORKKA, P., AND TOIVONEN, H. 1999. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J. 42,* 2, 100–111.

KRATOCHVÍL, J. 1993. Precoloring extension with fixed color bound. *Acta Math. Univ. Comen 62*, 139–153.

LENZERINI, M. 2002. Data integration: A theoretical perspective. In *PODS*.

LEVY, A. Y., MENDELZON, A. O., SAGIV, Y., AND SRIVASTAVA, D. 1995. Answering queries using views. In *PODS*.

LI, C. 2003. Computing complete answers to queries in the presence of limited access patterns. *VLDB J. 12,* 3, 211–227.

MAIER, D., MENDELZON, A. O., AND SAGIV, Y. 1979. Testing implications of data dependencies. *ACM Transactions on Database Systems (TODS) 4,* 4, 455–469.

NASH, A. AND LUDÄSCHER, B. 2004. Processing first-order queries under limited access patterns. In *PODS*.

NASH, A., SEGOUFIN, L., AND VIANU, V. 2010. Views and queries: Determinacy and rewriting. *ACM Trans. on Database Systems 35,* 3.

RAJARAMAN, A., SAGIV, Y., AND ULLMAN, J. D. 1995. Answering queries using templates with binding patterns. In *PODS*.

RAMAKRISHNAN, R. AND GEHRKE, J. 2000. *Database management systems*. McGraw Hill.

SAGIV, Y. AND YANNAKAKIS, M. 1980. Equivalences among relational expressions with the union and difference operators. *J. ACM 27,* 4, 633–655.

STOCKMEYER, L. J. 1976. The polynomial-time hierarchy. *TCS 3,* 1, 1–22.

STOLBOUSHKIN, A. P. AND TAITSLIN, M. A. 1995. Finite queries do not have effective syntax. In *PODS*.

ULLMAN, J. D. 1982. *Principles of Database Systems, 2nd Edition*. Computer Science Press.

WAGNER, K. W. 1987. More complicated questions about maxima and minima, and some closures of NP. *TCS 51,* 1-2, 53–80.

YU, C. T. AND ÖZSOYOĞLU, M. Z. 1979. An algorithm for tree-query membership of a distributed query. In *COMPSAC*. 306–312.

Overview of used notations

Table II. Notations

symbols	notations
$\mathcal{R}$, R	database schema $\mathcal{R}$ and relation schema $R \in \mathcal{R}$
$\mathcal{A}$	access schema
$\mathcal{D} \models \mathcal{A}$	an instance $\mathcal{D}$ of $\mathcal{R}$ satisfies access schema $\mathcal{A}$
$Q \in \mathcal{L}$	query Q in a query language $\mathcal{L}$
$\mathcal{V}$, V	a set $\mathcal{V}$ of views and a view $V \in \mathcal{V}$
$\xi(Q, \mathcal{V}, \mathcal{R})$	a query plan ξ for Q using $\mathcal{V}$ over instances of $\mathcal{R}$
T_ξ	query plan ξ represented as a query tree
$\xi(\mathcal{D})$	the result of applying ξ to $\mathcal{D}$
VBRP($\mathcal{L}$)	the bounded rewriting problem for queries in $\mathcal{L}$
VBRP$^+(\mathcal{L}_1, \mathcal{L}_2)$	the problem of $\mathcal{L}_1$-to-$\mathcal{L}_2$ bounded rewriting using views
$Q \equiv_\mathcal{A} Q'$	$\mathcal{A}$-equivalence
$Q \sqsubseteq_\mathcal{A} Q'$	$\mathcal{A}$-containment
QP$_Q$	the set of all possible query plans of a bounded size
$\xi \sqsubseteq_\mathcal{A} Q$	$Q_\xi \sqsubseteq_\mathcal{A} Q$, query Q_ξ expressed by ξ

Appendix: Proofs

Proof of Theorem 4.1

We prove that VBRP(ACQ) is coNP-hard under fixed $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$, and when $\mathcal{A}$ has the forms specified cases (1), (2) and (3).

(1) When $\mathcal{A}$ consists of a single $R(A \to B, N)$ **and** $N \geqslant 2$. Consider a database schema $\mathcal{R}$ with a single binary relation $R(A, B)$. We assume M to be any predefined constant, and $\mathcal{V}$ to be any fixed set of ACQ queries. We start with $N = 2$, and will show that VBRP remains coNP hard when $N > 2$. We show that VBRP(ACQ) is coNP-hard in this setting by reduction from the complement of the precoloring extension problem, which is NP-complete [Kratochvíl 1993]. Given an undirected graph $G = (V_G, E)$, a precoloring μ_0 is a coloring of a *subset* W of the nodes of V_G with colors in $\{r, g, b\}$. The precoloring extension problem is to decide whether μ_0 can be extended to a coloring μ of the *entire* set of nodes in V_G with colors in $\{r, g, b\}$. That is, whether there exists a coloring μ of all nodes in V_G such that $\mu(v) = \mu_0(v)$ for each $v \in W$ and $\mu(v) \neq \mu(w)$ whenever $(v, w) \in E$. From the proof in [Kratochvíl 1993], we know that the problem remains NP-complete when each connected component in G has at least one leaf (degree-one node) and the precoloring μ_0 only assigns colors to the leaves of G.

Given a graph $G = (V_G, E)$ and a 3-coloring μ_0 of the leaves V_1 of G, where $V_G = \{v_1, \ldots, v_n\}$, $V_1 \subseteq V_G$, and $E = \{e_1, \ldots, e_m\}$, we define an ACQ Q, such that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if the precoloring μ_0 cannot be extended to a valid coloring of G. The query Q is constructed as follows:

$$Q() = \exists \bar{x}_1, \bar{x}_2, \bar{v} \left(Q_E(\bar{x}_1, \bar{x}_2) \wedge \bigwedge_{v_i \in V_G} Q^1_V(v_i, \bar{x}_1) \wedge \bigwedge_{v_i \in V_G} Q^2_V(v_i, \bar{x}_2) \wedge \bigwedge_{v_i \in V_1} Q_L(v_i) \wedge Q_1() \wedge Q_f() \right).$$

Here $\bar{x}_1$ and $\bar{x}_2$ consist of variables $x^1_{(v_i,v_j)}$ and $x^2_{(v_i,v_j)}$, respectively, for $v_i, v_j \in V_G$.

Intuitively, Q_E, Q^1_V and Q^2_V encode graph G, $Q_L(v_i)$ enforces the precoloring, Q_1 checks whether G is 3-colored, and Q_f controls M, as will be elaborated shortly. The challenge arises from encoding a cyclic graph G in ACQ. We approach this as follows. We first replace two vertices of any edge in G with two distinct new variables, and then use the fixed access constraint to recover the original G. For example, a cycle $(v_1, v_2), (v_2, v_3), (v_3, v_1)$ in G is represented by the following set of atoms in Q_E:

$R(x^1_{(v_1,v_2)}, x^2_{(v_1,v_2)})$, $R(x^1_{(v_2,v_3)}, x^2_{(v_2,v_3)})$ and $R(x^1_{(v_3,v_1)}, x^2_{(v_3,v_1)})$, where $x^1_{(v_1,v_2)} \in \bar{x}_1$ and $x^2_{(v_1,v_2)} \in \bar{x}_2$; similarly for the other variables. Although these seem like disconnected edges, we will ensure that $v_1 = x^1_{(v_1,v_2)} = x^2_{(v_3,v_1)}$, $v_2 = x^2_{(v_1,v_2)} = x^1_{(v_2,v_3)}$ and $v_3 = x^2_{(v_2,v_3)} = x^1_{(v_3,v_1)}$ by means of the access constraint and queries Q^1_V and Q^2_V. This allows us to recover the cycle in G, and encode cyclic graphs in an acyclic CQ query.

We will use relation R to encode edges in E as well as coloring of vertices in V_G.

Queries $Q_E, Q^1_V, Q^2_V, Q_L, Q_1$ and Q_f are in ACQ, and are given as follows.

– $Q_E(\bar{x}_1, \bar{x}_2) = \bigwedge_{(v_i,v_j)\in E} \left(R(x^1_{(v_i,v_j)}, x^2_{(v_i,v_j)}) \wedge R(x^2_{(v_i,v_j)}, x^1_{(v_i,v_j)}) \right)$. This sub-query renames the nodes of each edge in G. We use two directed edges to encode one edge in G so that the access constraint $R(A \to B, 2)$ can be used to recover variables $x^1_{(v_i,v_j)}$ and $x^2_{(v_i,v_j)}$.

– $$Q^1_V(v_i, \bar{x}_1) = \bigwedge_{e_j=(v_i,v_j)\in E} \Big(\underbrace{(R(i,1) \wedge R(i,v_i) \wedge R(i,x^1_{e_j}))}_{Q_{e^1_j}} \wedge \underbrace{(R(i+n,2) \wedge R(i+n,v_i) \wedge R(i+n,x^1_{e_j}))}_{Q_{e^2_j}} \wedge \underbrace{(R(i+2*n,3) \wedge R(i+2*n,v_i) \wedge R(i+2*n,x^1_{e_j}))}_{Q_{e^3_j}} \Big).$$

It ensures that each variable $x^1_{e_j} = x^1_{(v_i,v_j)}$ in $\bar{x}_1$ denotes node v_i by enforcing that $x^1_{e_j} = v_i$. Suppose that $x^1_{e_j} \neq v_i$. Then by the access constraint $R(A \to B, 2) \in \mathcal{A}$, from $Q_{e^1_j}$ (marked underlying Q^1_V) we know that $v_i = 1 \vee x^1_{e_j} = 1$. Similarly, $v_i = 2 \vee x^1_{e_j} = 2$ or $v_i = 3 \vee x^1_{e_j} = 3$ by $Q_{e^2_j}$ and $Q_{e^3_j}$. That is, $\{v_i, x^1_{e_j}\} = \{1,2,3\}$, a contradiction. Hence $x^1_{e_j} = v_i$.

– $$Q^2_V(v_i, \bar{x}_2) = \bigwedge_{e_j=(v_j,v_i)\in E} \Big(\underbrace{(R(i,1) \wedge R(i,v_i) \wedge R(i,x^2_{e_j}))}_{Q_{e^1_j}} \wedge \underbrace{(R(i+n,2) \wedge R(i+n,v_i) \wedge R(i+n,x^2_{e_j}))}_{Q_{e^2_j}} \wedge \underbrace{(R(i+2*n,3) \wedge R(i+2*n,v_i) \wedge R(i+2*n,x^2_{e_j}))}_{Q_{e^3_j}} \Big).$$

This ensures that each variable $x^2_{e_j}$ in $\bar{x}_2$ corresponding to v_i satisfies $x^2_{e_j} = v_i$.

– $$Q_L(v_i) = \underbrace{(R(i,1) \wedge R(i,v_i) \wedge R(i,\mu_0(v_i)))}_{Q^1_{v_i}} \wedge \underbrace{(R(i+n,2) \wedge R(i+n,v_i) \wedge R(i+n,\mu_0(v_i)))}_{Q^2_{v_i}} \wedge \underbrace{(R(i+2*n,3) \wedge R(i+2*n,v_i) \wedge R(i+2*n,\mu_0(v_i)))}_{Q^3_{v_i}}.$$

This is to ensure that for each vertex $v \in V_1$, $v = \mu_0(v)$, *i.e.,* the coloring preserves the precoloring μ_0 of the leaves, making use of R to encode coloring.

– $Q_1() = R(r,g) \wedge R(r,b) \wedge R(g,r) \wedge R(g,b) \wedge R(b,r) \wedge R(b,g)$. It is to ensure that graph G is colored with $\{r,g,b\}$. More specifically, consider any instance $D \models \mathcal{A}$ of R such that $Q(D) \neq \emptyset$. Suppose that ν is a valuation of the variables of Q to vertices in D. We next show that for each vertex $v \in V_G$, $\nu(v) \in \{r,g,b\}$, *i.e.,* G is colored with $\{r,g,b\}$, and for any edge $(v,v') \in E$, $\mu(v) \neq \mu(v')$. Let v be any vertex in V_G. Since we assume that each connected component of G has at least one leaf and each edge of G is represented by two directed edges in G_E, there exist a leaf $v_1 \in V_1$ and a path $v_1, v'_1, v'_2, \ldots, v'_t, v$ from v_1 to v. Hence, there exist tuples $R(\nu(v_1), \nu(v'_1)), R(\nu(v'_1), \nu(v'_2)), \ldots, R(\nu(v'_t), \nu(v))$ in D. Since v_1 is a leaf, we know that $v_1 = \mu_0(v_1)$ and $\nu(v_1) \in \{r,g,b\}$. Suppose *w.l.o.g.* that $\nu(v_1) = r$. Since $Q_1(D) \neq \emptyset$, there exist two tuples $R(r,g)$ and $R(r,b)$ in D. By the access constraint $R(A \to B, 2)$, we have that $\nu(v'_1) \in \{g,b\}$. Similarly, we can show that $\nu(v) \in \{r,g,b\}$ and hence the coloring is valid (see details below).

– $Q_f() = \exists Y \left(\bigwedge_{i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)} R(y_i, i) \right)$. It is to fill Q with $M \times |\mathcal{V}| \times |\mathcal{R}|$ constants. Since there are already another three constants r, g, and b in Q, if Q is satisfiable, then each element query of Q contains at least $M \times |\mathcal{V}| \times |\mathcal{R}| + 3$ constants. However, each M-bounded rewriting can only have at most $M \times |\mathcal{V}| \times |\mathcal{R}|$ constants. Indeed, such an M-bounded rewriting can have at most $M \times |\mathcal{V}|$ atoms, and thus has at most $M \times |\mathcal{V}| \times |\mathcal{R}|$ distinct constants. Therefore, Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if $Q \equiv_{\mathcal{A}} \emptyset$.

Obviously, Q is an ACQ. By the definition of Q_f, we can see that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if $Q \equiv_{\mathcal{A}} \emptyset$. Thus it suffices to show that $Q \equiv_{\mathcal{A}} \emptyset$ if and only if the precoloring μ_0 cannot be extended to a valid coloring of G.

($\Leftarrow$) Suppose that μ_0 cannot be extended to a valid coloring of G. We show that $Q \equiv_{\mathcal{A}} \emptyset$ by contradiction. Suppose that $Q \not\equiv_{\mathcal{A}} \emptyset$. Then there exists $D \models \mathcal{A}$ such that $Q(D) \neq \emptyset$. Let ν be a valuation of variables in Q. Clearly, $Q_E(D) \neq \emptyset$, $Q_L(D) \neq \emptyset$, and $Q_1(D) \neq \emptyset$. From $Q_L(D) \neq \emptyset$, we know that for each leaf $v \in V_1$, $\nu(v) = \mu_0(v)$. Since $Q_1(D) \neq \emptyset$, there are tuples $R(r,g), R(r,b), R(g,r), R(g,b), R(b,r), R(b,g)$ in D. Putting these together, by the argument about Q_1 we can conclude that G is colored with $\{r,g,b\}$. Since $Q_E(D) \neq \emptyset$ and $R(A \to B, 2)$, for every edge $(v_i, v_j) \in E$, $\nu(v_i) \neq \nu(v_j)$. Indeed, suppose otherwise that $\nu(v_i) = \nu(v_j) = r$, then there will be a tuple $R(r,r)$ in D. Now there are three tuples $R(r,g)$, $R(r,b)$ and $R(r,r)$ in D, contradicting that $D \models \mathcal{A}$. Therefore, there exists a valid 3-coloring of G extending μ_0, a contradiction. Hence $Q \equiv_{\mathcal{A}} \emptyset$.

($\Rightarrow$) Suppose that $Q \equiv_{\mathcal{A}} \emptyset$. We show that μ_0 cannot be extended to a valid coloring of G. Suppose by contradiction that μ is a valid 3-coloring of G that extends the precoloring μ_0. We construct below an instance D of R such that $D \models \mathcal{A}$ and $Q(D) \neq \emptyset$, which will contradict to that $Q \equiv_{\mathcal{A}} \emptyset$. The database D consists of the following tuples.

(1) The 6 tuples demanded by Q_1: $R(r,g), R(r,b), R(g,r), R(g,b), R(b,r), R(b,g)$.
(2) For each vertex $v_i \in V_G$, 6 tuples corresponding to the queries Q_V^1 and Q_V^2:
$$R(i,1), R(i+n,2), R(i+2n,3), R(i,\mu(v_i)), R(i+n,\mu(v_i)), R(i+2n,\mu(v_i)).$$
(3) For each natural number $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$, one tuple $R(c_i, i)$, where c_i's are distinct new constants.

It is easy to verify that $D \models \mathcal{A}$. We next show that $Q(D) \neq \emptyset$. Since Q_V^1 and Q_V^2 ensure that each variable in $\bar{x}_1$ or $\bar{x}_2$ equals the corresponding variable in V_G, we know that Q can be simplified to the following query:

$$Q_2() = \exists \bar{v} \left(Q'_E(\bar{v}) \wedge \bigwedge_{v_i \in V_G} Q'_V(v_i) \wedge \bigwedge_{v_i \in V_G} Q''_V(v_i) \wedge \bigwedge_{v_i \in V_1} Q_L(v_i) \wedge Q_1() \wedge Q_f() \right).$$

Here $Q_L, Q_1()$ and Q_f are sub-queries Q, and Q'_E, Q'_V, and Q''_V are defined as follows:

– $Q'_E(\bar{v}) = \bigwedge_{(v_i,v_j)\in E} \Big(R(v_i,v_j) \wedge R(v_j,v_i)\Big)$;

– $Q'_V(v_i) = \bigwedge_{e_j=(v_i,v_j)\in E} \Big(\underbrace{(R(i,1) \wedge R(i,v_i))}_{Q_{e_j^1}} \wedge \underbrace{(R(i+n,2) \wedge R(i+n,v_i))}_{Q_{e_j^2}} \wedge \underbrace{(R(i+2*n,3) \wedge R(i+2*n,v_i))}_{Q_{e_j^3}} \Big)$;

– $Q''_V(v_i) = \bigwedge_{e_j=(v_j,v_i)\in E} \Big(\underbrace{(R(i,1) \wedge R(i,v_i))}_{Q_{e_j^1}} \wedge \underbrace{(R(i+n,2) \wedge R(i+n,v_i))}_{Q_{e_j^2}} \wedge \underbrace{(R(i+2*n,3) \wedge R(i+2*n,v_i))}_{Q_{e_j^3}} \Big)$.

Since Q_2 is obtained from Q by simply replacing equivalent variables in Q, $Q_2 \equiv_{\mathcal{A}} Q$. Since the variables in Q_2 occur in either $\bar{v}$ or Y (Y is used in Q_f), we can construct a valuation ν of variables of Q_2 as follows: for each $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$, $\nu(y_i) = c_i$; and for each vertex $v \in V_G$, $\nu(v) = \mu(v)$. One can verify that ν satisfies $\nu(Q_2) \subseteq D$ and hence $Q \not\equiv_{\mathcal{A}} \emptyset$, contradicting $Q \equiv_{\mathcal{A}} \emptyset$. Thus μ_0 cannot be extended to a valid coloring of G.

When $N > 2$, we only need to fill the relation R with some constants and use the same reduction. For example, when $N = 3$, let c_1, c_2 and c_3 be distinct new constants. Then Q_1 can be rewritten as $Q_1() = R(r,g) \wedge R(r,b) \wedge R(g,r) \wedge R(g,b) \wedge R(b,r) \wedge R(b,g) \wedge R(r,c_1) \wedge R(g,c_2) \wedge R(b,c_3)$. This revised Q_1 also ensures that G is colored by $\{r,g,b\}$. Indeed, consider any $D \models \mathcal{A}$ such that $Q(D) \neq \emptyset$. Suppose that ν is a valuation of variables of Q. We show that for each vertex $v \in V_G$, $\nu(v) \in \{r,g,b\}$. Let v be any vertex in V_G. Similar to the argument above, we can show that there exist a leaf $v_1 \in V_1$ and a path $v_1, v'_1, v'_2, \ldots, v'_t, v$ from v_1 to v. Hence, there exist tuples $R(\nu(v_1), \nu(v'_1)), R(\nu(v'_1), \nu(v'_2)), \ldots, R(\nu(v'_t), \nu(v))$ in D. Because v_1 is a leaf, we know that $v_1 = \mu_0(v_1)$ and $\nu(v_1) \in \{r,g,b\}$. Suppose that $\nu(v_1) = r$. Since $Q_1(D) \neq \emptyset$, there are three tuples $R(r,g)$, $R(r,b)$, and $R(r,c_1)$ in D. By constraint $R(A \to B, 3)$, we have that $\nu(v'_1) \in \{g,b,c_1\}$. However, by the construction of Q_E, tuple $R(\nu(v'_1), \nu(v_1))$ is also in D. Hence, $\nu(v'_1)$ also appears in the first column of R. But c_1 only appears in the second column of R. Hence $\nu(v'_1) \neq c_1$ and $\nu(v'_1) \in \{g,b\}$. Using the same argument, we can show that $\nu(v) \in \{r,g,b\}$. The other queries can be revised similarly. For other values of N, we can verify the coNP hardness along the same lines.

(2) When $\mathcal{A}$ consists of two access constraints $R(A \to B, 1)$ and $R'(\emptyset \to (E,F), N)$, and $N \geqslant 6$. We will only use the binary relations $R(A,B)$ and $R'(E,F)$ in our proof. We take any predefined constant as M, and $\mathcal{V}$ to be any fixed set of ACQ queries.

We start with $N = 6$ and then extend the proof to $N > 6$. We show that VBRP(ACQ) is coNP-hard in this setting by reduction from the complement of the 3-Colorability problem, which is NP-complete (cf. [Garey and Johnson 1979]). The 3-Colorability problem is to decide, given an undirected graph $G = (V_G, E)$, whether there exists a 3-coloring $\mu : V_G \to \{r,g,b\}$ such that for every edge $(v_i,v_j) \in E$, $\mu(v_i) \neq \mu(v_j)$.

Given $G = (V_G, E)$ with $V_G = \{v_1, \ldots, v_n\}$, we define an ACQ Q such that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if G is not 3-colorable:

$$Q() = \exists \bar{v}, \bar{x}_1, \bar{x}_2 \big(Q_E(\bar{x}_1, \bar{x}_2) \wedge Q_V(\bar{v}, \bar{x}_1, \bar{x}_2) \wedge Q_1() \wedge Q_f()\big).$$

Here Q_E, Q_V, Q_1, and Q_f are similar to their counterparts in case (1), as follows:

– $Q_E(\bar{x}_1, \bar{x}_2) = \bigwedge_{(v_i,v_j)\in E} \Big(R'(x^1_{(v_i,v_j)}, x^2_{(v_i,v_j)}) \wedge R'(x^2_{(v_i,v_j)}, x^1_{(v_i,v_j)})\Big)$, where $x^1_{(v_i,v_j)} \in \bar{x}_1$ and $x^2_{(v_i,v_j)} \in \bar{x}_2$. It renames the nodes of each edge in G as in case (1).
– $Q_V(\bar{v}, \bar{x}_1, \bar{x}_2) = \bigwedge_{v_i\in V_G} \Big(R(i, v_i) \wedge \bigwedge_{(v_i,v_2)\in E} R(i, x^1_{(v_i,v_2)}) \wedge \bigwedge_{(v_1,v_i)\in E} R(i, x^2_{(v_1,v_i)})\Big)$. It is to recover the original G. Indeed, constraint $R(A \to B, 1)$ ensures that the variables v_i, $x^1_{(v_i,v_2)}$ and $x^2_{(v_2,v_i)}$ are "equivalent", *i.e.,* they always take the same value.
– We define $Q_1() = R'(r, g) \wedge R'(r, b) \wedge R'(g, r) \wedge R'(g, b) \wedge R'(b, r) \wedge R'(b, g)$. Since $R'(\emptyset \to (E, F), 6) \in \mathcal{A}$, Q_1 ensures that if $Q()$ is satisfiable, then there exists a valid 3-coloring of G. Consider any $\mathcal{D} \models \mathcal{A}$ such that $Q(\mathcal{D}) \neq \emptyset$, and let ν be a valuation of variables of Q in $\mathcal{D}$. Then $Q_E(\mathcal{D}) \neq \emptyset$, $Q_V(\mathcal{D}) \neq \emptyset$, and $Q_1(\mathcal{D}) \neq \emptyset$. From $Q_E(\mathcal{D}) \neq \emptyset$, we know that for each edge $(v_i, v_j) \in E$, there exists a tuple $R'(\nu(x^1_{(v_i,v_j)}), \nu(x^2_{(v_i,v_j)})$ in $\mathcal{D}$. By $Q_V(\mathcal{D}) \neq \emptyset$ and constraint $R(A \to B, 1)$, we have that $\nu(v_i) = \nu(x^1_{(v_i,v_j)})$ and $\nu(v_j) = \nu(x^2_{(v_i,v_j)})$. Hence, there exists a tuple $R'(\nu(v_i), \nu(v_j))$ in $\mathcal{D}$. On the other hand, since $Q_1(\mathcal{D}) \neq \emptyset$, there are six tuples $R'(r, g), R'(r, b), R'(g, r), R'(g, b), R'(b, r), R'(b, g)$ in $\mathcal{D}$. By $R'(\emptyset \to (E, F), 6) \in \mathcal{A}$, for each vertex $v \in V_G$, $\nu(v) \in \{r, g, b\}$, and for each edge $(v_i, v_j) \in E$, $\nu(v_i) \neq \nu(v_j)$. Indeed, the relation R' must otherwise have more than six tuples and $\mathcal{D} \models \mathcal{A}$ does not hold. Therefore, if $Q()$ is satisfiable, then there exists a correct 3-coloring of G.
– $Q_f() = \exists Y \bigwedge_{i\leqslant(M\times|\mathcal{V}|\times|\mathcal{R}|)} R(y_i, i)$. It fills Q with sufficiently many constants such that if Q is satisfiable, then Q does not have an M-bounded rewriting in ACQ.

It can be verified that Q is in ACQ along the same lines as in case (1).

From the definition of Q_f, we can conclude that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if $Q \equiv_{\mathcal{A}} \emptyset$. Indeed, from the argument above, we can see that if $Q \not\equiv_{\mathcal{A}} \emptyset$, then G is 3-colorable. Hence we only need to show that if $Q \equiv_{\mathcal{A}} \emptyset$, then G is not 3-colorable. We show this by contradiction. Let μ be a valid 3-coloring of G. We construct a database $\mathcal{D}$ such that $\mathcal{D} \models \mathcal{A}$ and $Q(\mathcal{D}) \neq \emptyset$, which contradict to the assumption that $Q \equiv_{\mathcal{A}} \emptyset$. The database $\mathcal{D}$ consists of the following tuples:

(1) the 6 tuples in Q_1: $R'(r, g), R'(r, b), R'(g, r), R'(g, b), R'(b, r), R'(b, g)$;
(2) for each vertex $v_i \in V_G$, one tuple encoding the 3-coloring of G: $R(i, \mu(v_i))$; and
(3) for each natural number $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$, one tuple $R(c_i, i)$, where c_i's are distinct new constants.

It is easy to verify that $\mathcal{D} \models \mathcal{A}$. We next show that $Q(\mathcal{D}) \neq \emptyset$. Since Q_V ensures that each variable in X^1 or X^2 is equal to the corresponding variable in V_G, we know that Q can be simplified to the following query:

$$Q_2() = \exists \bar{v}\big(Q'_E(\bar{v}) \wedge Q'_V(\bar{v}) \wedge Q_1() \wedge Q_c()\big).$$

Here $Q_1()$ and Q_c are the queries already defined in Q, and Q'_E, and Q'_V are defined as:

– $Q'_E(\bar{v}) = \bigwedge_{(v_i,v_j)\in E} \Big(R'(v_i, v_j) \wedge R'(v_j, v_i)\Big)$; and
– $Q'_V(\bar{v}) = \bigwedge_{v_i\in V_G} R(i, v_i)$.

Since Q_2 is obtained from Q by replacing equivalent variables in Q with those in V_G, we have that $Q_2 \equiv_{\mathcal{A}} Q$. Moreover, since the only variables in Q_2 occur in $\bar{v}$ or Y (in Q_c), we can construct a valuation ν of variables of Q_2 in $\mathcal{D}$ as follows: for each $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$ (see the proof of case (1) of Theorem 4.1 for an explanation of $M \times |\mathcal{V}| \times |\mathcal{R}|$), $\nu(y_i) = c_i$; and for each vertex $v \in V_G$, $\nu(v) = \mu(v)$. We can verify that ν satisfies $\nu(Q_2) \subseteq \mathcal{D}$. That is $Q \not\equiv_{\mathcal{A}} \emptyset$, which contradicts to the assumption that $Q \equiv_{\mathcal{A}} \emptyset$. Hence G is not 3-colorable.

For $N > 6$, we only need to fill the relation R' with some additional constants such that the same reduction works. For example, suppose that $N = 7$, and let d_1 and d_2 be two distinct new constants. Then we modify Q_1 as $Q_1() = R'(d_1, d_2) \wedge R'(r, g) \wedge R'(r, b) \wedge R'(g, r) \wedge R'(g, b) \wedge R'(b, r) \wedge R'(b, g)$. Similar to the proof of case (1) of Theorem 4.1, this revised query can also ensure that if $Q()$ is satisfiable, then there exists a valid 3-coloring of G. Indeed, consider any $\mathcal{D} \models \mathcal{A}$ such that $Q(\mathcal{D}) \neq \emptyset$, and let ν be a valuation of variables of Q in $\mathcal{D}$. Then $Q_E(\mathcal{D}) \neq \emptyset$, $Q_V(\mathcal{D}) \neq \emptyset$, and $Q_1(\mathcal{D}) \neq \emptyset$. From $Q_E(\mathcal{D}) \neq \emptyset$, we know that for each edge $(v_i, v_j) \in E$, there exists a tuple $R'(\nu(x^1_{(v_i,v_j)}), \nu(x^2_{(v_i,v_j)}))$ in $\mathcal{D}$. By $Q_V(\mathcal{D}) \neq \emptyset$ and $R(A \to B, 1)$, we have that $\nu(v_i) = \nu(x^1_{(v_i,v_j)})$ and $\nu(v_j) = \nu(x^2_{(v_i,v_j)})$. Thus there exists a tuple $R'(\nu(v_i), \nu(v_j))$ in $\mathcal{D}$. On the other hand, since $Q_1(\mathcal{D}) \neq \emptyset$, there are seven tuples $R'(d_1, d_2), R'(r, g), R'(r, b), R'(g, r), R'(g, b), R'(b, r)$ and $R'(b, g)$ in $\mathcal{D}$. By $Q_E(\mathcal{D}) \neq \emptyset$ and access constraint $R'(\emptyset \to (E, F), 7) \in \mathcal{A}$, for each edge $(v_i, v_j) \in E$, there must exist a tuple $R'(\nu(v_i), \nu(v_j))$ in $\mathcal{D}$, $\nu(v_i) \in \{r, g, b, d_1\}$, and $\nu(v_j) \in \{r, g, b, d_2\}$. Since we encode each edge in G by two directed edges, there is also another tuple $R'(\nu(v_j), \nu(v_i))$ in $\mathcal{D}$. However, d_1 can only appear in the first column of R' and d_2 can only appear in the second column of R'. Indeed, otherwise R' would consist of more than 7 tuples, contradicting to that $\mathcal{D} \models \mathcal{A}$. Therefore, $\nu(v_i) \in \{r, g, b\}$ and $\nu(v_j) \in \{r, g, b\}$. Suppose that G is not 3-colorable. Then there exists an edge $(v_i, v_j) \in E$ such that $\nu(v_i) = \nu(v_j)$. Let $\nu(v_j) = r$. Then there exists a tuple $R'(r, r)$ in $\mathcal{D}$, which contradicts to the assumption that $\mathcal{D} \models \mathcal{A}$. Hence G is 3-colorable. Therefore, if $Q()$ is satisfiable, there exists a correct 3-coloring of G.

For other values of N, we can modify Q_1 along the same lines.

(3) When $\mathcal{A}$ consists of $R((A, B) \to C, 1)$ and $R'(\emptyset \to E, N)$, and $N \geqslant 2$. We assume that $\mathcal{R}$ consists of a ternary relation $R(A, B, C)$ and a unary relation $R'(E)$, M is any constant, and $\mathcal{V}$ is any fixed set of ACQ queries.

We start with $N = 2$ and then extend the proof to $N > 2$. We show that VBRP(ACQ) is coNP-hard in this setting by reduction from the complement of the 3SAT problem (see the proof of Theorem 3.4 for 3SAT). Consider an instance ψ of 3SAT, where ψ contains k clauses $C_1, C_2, \ldots, C_k$ defined over variables in $X = \{x_1, \ldots, x_m\}$. We define an ACQ query Q such that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if ψ is false.

The query Q is defined in ACQ and is constructed as follows:

$$Q() = \exists \bar{x}, \bar{x}_1, \bar{x}_2, \bar{x}_3 \Big(Q_{01}() \wedge Q_{\vee}() \wedge Q_{\wedge}() \wedge Q_{\neg}() \wedge Q_X(\bar{x}) \wedge Q_v(\bar{x}, \bar{x}_1, \bar{x}_2, \bar{x}_3) \wedge Q_{\psi}(\bar{x}_1, \bar{x}_2, \bar{x}_3) \wedge Q_f() \Big).$$

Here $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{x}_i = (x^i_1, \ldots, x^i_m)$ for $i = 1, 2, 3$. In order for Q to encode ψ we need Boolean operations. However, in contrast to the proof of Theorem 3.4, here we only have one ternary relation to store instances similar to those shown in Figure 2. Hence, we store all tuples needed in R and use different constants in the A-attribute to extract from R the right set of tuples that encode each of the Boolean operations. The definitions of $Q_\vee$, $Q_\wedge$ and $Q_\neg$ are such defined that tuples with their A-attribute set to 0, 1, $\triangle$, or ∇ encode Boolean disjunction; tuples with their A-attribute set to $\bot$ or $\top$ encode Boolean conjunction; and tuples with their A-attribute set to $\star$ encode Boolean negation. More specifically, $Q_\wedge, Q_\vee$, and $Q_\neg$ are defined as follows:

- $Q_\vee() = \big(R(0,0,\triangle) \wedge R(0,1,\nabla) \wedge R(1,0,\nabla) \wedge R(1,1,\nabla)\big) \wedge \big(R(\triangle,0,\bot) \wedge R(\triangle,1,\top) \wedge R(\nabla,0,\top) \wedge R(\nabla,1,\top)\big)$;
- $Q_\wedge() = R(\bot,\bot,\bot) \wedge R(\bot,\top,\bot) \wedge R(\top,\bot,\bot) \wedge R(\top,\top,\top)$;
- $Q_\neg() = R(\star,0,1) \wedge R(\star,1,0)$; and

where $\nabla, \triangle, \bot, \top$ and $\star$ are new constants. The constants $\triangle, \bot$ and $\nabla, \top$ represent false and true, respectively. Sub-queries Q_{01} and Q_X are defined as follows:

- $Q_{01}() = R'(0) \wedge R'(1) \wedge R(\blacktriangleright,0,0) \wedge R(\blacktriangleright,1,1)$ is used to encode the Boolean values false and true; here $\blacktriangleright$ is a new constant; and
- $Q_X(\bar{x}) = \bigwedge_{1 \leqslant i \leqslant m} (R'(x_i) \wedge R(\blacktriangleright, x_i, x_i))$ ensures that $\bar{x}$ is a truth assignment of X. Indeed, constraint $R'(\emptyset \to E, 2)$ together with Q_{01} ensures that each x_i is mapped to $\{0,1\}$ when Q_X is evaluated on instances $\mathcal{D}$ such that $\mathcal{D} \models \mathcal{A}$ and $Q_{01}(\mathcal{D}) \neq \emptyset$.

It should be remarked that even without using the relation R, $Q_{01}()$ and $Q_X(\bar{x})$ can also ensure that $\bar{x}$ is a truth assignment of X. However, we will use these atoms to handle the cases when $N > 2$, as will become clear shortly.

Furthermore, Q_v, Q_ψ and Q_f are defined as follows:

- $Q_v(\bar{x}, \bar{x}_1, \bar{x}_2, \bar{x}_3) = \left(\bigwedge_{1 \leqslant i \leqslant m} R(i+2, \bullet, x_i)\right) \wedge \left(\bigwedge_{1 \leqslant i \leqslant k} R(f_1(C_i)+2, \bullet, x_i^1) \wedge R(f_2(C_i)+2, \bullet, x_i^2) \wedge R(f_3(C_i)+2, \bullet, x_i^3)\right)$, where $\bullet$ is a new constant and for $j = 1,2,3$, $f_j(C_i) = \ell$ if x_ℓ is the jth variable in clause C_i.
  This query is used to rename the variables in $\bar{x}$ such that each clause has a new copy of the variables in $\bar{x}$, represented by $\bar{x}_1, \bar{x}_2$, and $\bar{x}_3$, one copy for each of the three literals in a clause. Moreover, if $f_j(C_i) = \ell$ then x_i^j and x_ℓ are equivalent due to access constraint $R((A,B) \to C, 1)$, *i.e.,* these variables take the same values in instances $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$. This allows us to encode ψ by using distinct variables to ensure acyclicity, as will be elaborated shortly.
- Query $Q_\psi(\bar{x}_1, \bar{x}_2, \bar{x}_3)$ is to check whether ψ is true given a truth assignment μ_X encoded in $\bar{x}$ (and thus also in $\bar{x}_1, \bar{x}_2, \bar{x}_3$ since they carry the same values as $\bar{x}$) by query Q_v. We first explain how clauses C_j in ψ are encoded. Consider, *e.g.,* $C_j = x_1 \vee \overline{x}_2 \vee x_3$. We construct a query $Q_j(\bar{x}_1, \bar{x}_2, \bar{x}_3, y_j)$ such that y_j holds the truth value of C_j given μ_X. More specifically, $Q_j(\bar{x}_2, \bar{x}_2, \bar{x}_3, y_j) = \exists x_2', y'\, R(x_j^1, x_2', y') \wedge R(y', x_j^3, y_j) \wedge R(\star, x_j^2, x_2')$. Note that x_j^1, x_j^2 and x_j^3 take Boolean values as specified by x_1, x_2 and x_3, respectively, as argued earlier. Hence, by the definitions of $Q_\vee$ and $Q_\neg$, Q_j correctly encodes C_j. Moreover, observe that y_j is either $\bot$ (when C_j is false under μ_X) or $\top$ (when C_j is true under μ_X). In the context of acyclicity, it is also important to observe that the variables x_j^1, x_j^2 and x_j^3 are only used in $Q_j(\bar{x}_1, \bar{x}_2, \bar{x}_3, y_j)$ and in $Q_v(\bar{x}, \bar{x}_1, \bar{x}_2, \bar{x}_3)$, where they occur together with constants. The construction of Q_j is similar for clauses of another form. We now define

$$Q_\psi(\bar{x}_1, \bar{x}_2, \bar{x}_3) = \exists \bar{y} \left(\bigwedge_{1 \leqslant j \leqslant k} Q_j(\bar{x}_1, \bar{x}_2, \bar{x}_3, y_j)\right) \wedge Q'(y_1, y_2, \ldots, y_k),$$

where $Q'(y_1, y_2, \ldots, y_k)$ checks whether $y_1 \wedge y_2 \wedge \cdots \wedge y_k$ evaluates to true, *i.e.,* whether all clauses in ψ are satisfied. In particular, $Q'(\bar{y}) = \exists \bar{v}\, R(y_1, y_2, v_2) \wedge R(v_2, y_3, v_3) \wedge \cdots \wedge R(v_{k-1}, y_k, \top)$. Since the y_i's take values from $\{\bot, \top\}$, by the definition of $Q_\wedge$, Q' encodes the required conjunction and enforces all y_i's to be $\top$ (due to the last atom). The acyclicity of Q_ψ immediately follows from the use of distinct variables for each clause and the fact that these only appear in Q_v together with constants.

– Finally, we define $Q_f() = \left(\exists Y^1, Y^2 \bigwedge_{i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)} R(y_i^1, y_i^2, i)\right)$. It is used to fill Q with sufficiently many constants such that if Q is satisfiable, then Q does not have an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$.

From the definition of Q_f, we can conclude that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if $Q \equiv_{\mathcal{A}} \emptyset$. Thus we only need to verify that $Q \equiv_{\mathcal{A}} \emptyset$ if and only if ψ is false.

($\Leftarrow$) Suppose that ψ is false. We prove $Q \equiv_{\mathcal{A}} \emptyset$, by contradiction. If $Q \not\equiv_{\mathcal{A}} \emptyset$, then there exists $\mathcal{D} \models \mathcal{A}$ such that $Q(\mathcal{D}) \neq \emptyset$. Let ν be a valuation of variables of Q in $\mathcal{D}$. Since $Q_{01}(\mathcal{D}) \neq \emptyset$, $Q_X(\mathcal{D}) \neq \emptyset$, and $\mathcal{D} \models R'(\emptyset \to E, 2)$, for each variable $x \in X$, $\nu(x) \in \{0, 1\}$. We show that $\mu_0 = (\nu(x_1), \ldots, \nu(x_m))$ forms a truth assignment of X that makes ψ true. Indeed, since $Q'(\mathcal{D}) \neq \emptyset$ we know that $\nu(y_i) = \top$ for $1 \leqslant i \leqslant k$. This implies that $Q_j(\mu_0, \top)$ evaluates to true over $\mathcal{D}$ for each $j \in [1, k]$. In other words, each clause C_j is satisfied under μ_0, contradicting the assumption that ψ is false. Hence $Q \equiv_{\mathcal{A}} \emptyset$.

($\Rightarrow$) Suppose that $Q \equiv_{\mathcal{A}} \emptyset$. We show that ψ is false by contradiction. Let μ_0 be a truth assignment of X that makes ψ true. Based on μ_0, we construct an instance $\mathcal{D}$ of $\mathcal{R}$ such that $\mathcal{D} \models \mathcal{A}$ and $Q(\mathcal{D}) \neq \emptyset$. This contradicts to our assumption that $Q \equiv_{\mathcal{A}} \emptyset$. Therefore, ψ must be false. More specifically, database $\mathcal{D}$ consists of the following tuples:

(1) the 18 tuples in $Q_{01}, Q_\wedge, Q_\vee, Q_\neg$;
(2) for each variable $x_i \in X$, one tuple corresponding to $\mu_0(x_i)$: $R(i+2, \bullet, \mu_0(x_i))$; and
(3) for each natural number $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$, one tuple $R(c_i^1, c_i^2, i)$, where c_i^1 and c_i^2 are two distinct new constants.

It is easy to verify that $\mathcal{D} \models \mathcal{A}$. We next show that $Q(\mathcal{D}) \neq \emptyset$. Indeed, we can construct a valuation ν from variables of Q to values of $\mathcal{D}$ as follows: for each number $i \leqslant (M \times |\mathcal{V}| \times |\mathcal{R}|)$, $\nu(y_i^1) = c_i^1$ and $\nu(y_i^2) = c_i^2$; for each variable $x_i \in X$, $\nu(x_i) = \mu_0(x_i)$. Because μ_0 is a truth assignment of X that makes ψ true, we can easily verify that ν satisfies $\nu(Q) \subseteq \mathcal{D}$. Hence, $Q(\mathcal{D}) \neq \emptyset$ and thus $Q \not\equiv_{\mathcal{A}} \emptyset$.

For $N > 2$, we only need to fill the relations R' and R with more constants such that the same reduction as for $N = 2$ works. For example, suppose that $N = 3$, and let e_1 and e_2 be two distinct new constants. Then we modify Q_{01} as $Q_{01}() = R'(0) \wedge R'(1) \wedge R'(e_1) \wedge R(\blacktriangleright, 0, 0) \wedge R(\blacktriangleright, 1, 1) \wedge R(\blacktriangleright, e_1, e_2)$. This revised query can also ensure that $\bar{x}$ is a truth assignment of X. Indeed, consider any instance $\mathcal{D} \models \mathcal{A}$ of $\mathcal{R}$ such that $Q(\mathcal{D}) \neq \emptyset$, and let ν be a valuation of variables in Q. Then $Q_{01}(\mathcal{D}) \neq \emptyset$ and $Q_X(\mathcal{D}) \neq \emptyset$. By $Q_{01}(\mathcal{D}) \neq \emptyset$, there exist tuples $R'(0), R'(1)$, and $R'(e_1)$ in $\mathcal{D}$. By $Q_X(\mathcal{D}) \neq \emptyset$ and $R'(\emptyset \to E, 3)$, for each variable x_i we have that $\nu(x_i) \in \{0, 1, e_1\}$. Suppose that $\nu(x_i) = e_1$. Because $Q_X(\mathcal{D}) \neq \emptyset$, there exists a tuple $R(\blacktriangleright, e_1, e_1)$ in $\mathcal{D}$. However, since $Q_{01}(\mathcal{D}) \neq \emptyset$, there exists also a tuple $R(\blacktriangleright, e_1, e_2)$ in $\mathcal{D}$. From the access constraint $R((A, B) \to C, 1)$, we can conclude that $e_1 = e_2$, which contradicts to our assumption that e_1 and e_2 are two distinct constants. Hence $\nu(x_i) \neq e_1$ and thus $\nu(x_i) \in \{0, 1\}$. Therefore, $\bar{x}$ is a truth assignment of X. Using the same argument for the case $N = 2$, we can show that Q has an M-bounded rewriting in ACQ using $\mathcal{V}$ under $\mathcal{A}$ if and only if ψ is false.

For other values of $N > 2$, we can modify Q_{01} along the same lines. □

Proof of Theorem 5.1

We verify the three conditions of effective syntax one by one as follows.

(1) Each FO query Q with an M-bounded rewriting is $\mathcal{A}$-equivalent to a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. By definition, an FO query Q with an M-bounded rewriting is $\mathcal{A}$-equivalent to an M-bounded query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$. Hence, it suffices to show that for each M-bounded query plan ξ using $\mathcal{V}$ under $\mathcal{A}$, there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $\xi \equiv_\mathcal{A} Q_\xi$. We show this by induction on M. More specifically, we show that for any M-bounded query plan ξ using $\mathcal{V}$ under $\mathcal{A}$, there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $Q_\xi \equiv_\mathcal{A} \xi$.

Base case. We first show that the statement holds when M = 1. In this case, ξ can only be one of the following three forms (see the definition of query plans in Section 2): (i) a constant $\{c\}$; (ii) a view $V(\bar{x})$ in $\mathcal{V}$; or (iii) a fetch operator $\mathsf{fetch}(\emptyset, R, X)$ with access constraint $R(\emptyset \to X, N) \in \mathcal{A}$. Define Q_ξ as $x = c$, $V(\bar{x})$ or $\exists \bar{y}\, R(\bar{y}, \bar{x})$, respectively. Clearly, $Q_\xi \equiv_\mathcal{A} \xi$; so it remains to verify whether Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, *i.e.,* whether $\mathsf{covq}(Q_\varepsilon, Q_\xi)$ is true and $\mathsf{size}(Q_\varepsilon, Q_\xi) = 1$. This is an immediate consequence of the definition of these two functions. Indeed, case (i) corresponds to case (1) of topped queries given in Section 5; case (ii) corresponds to case (2) with $\bar{z} = \bar{x}$; and case (iii) corresponds to case (7a) with $\bar{z} = \bar{x}$ and $\bar{w} = \bar{y}$. Hence, Q_ξ is indeed topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Induction step. Suppose that the statement holds for $(M-1)$-bounded query plans ξ using $\mathcal{V}$ under $\mathcal{A}$. We next show that the statement also holds for M-bounded query plans ξ. By analyzing the structure of ξ we can distinguish the following six cases: (i) $\xi = (\xi', \sigma_{X=c}(\xi'))$ (resp. $(\xi', \sigma_{X=Y}(\xi'))$); (ii) $\xi = (\xi', \pi_Y(\xi'))$; (iii) $\xi = (\xi_1, \xi_2, \xi_1 \times \xi_2)$; (iv) $\xi = (\xi_1, \xi_2, \xi_1 \cup \xi_2)$; (v) $\xi = (\xi_1, \xi_2, \xi_1 - \xi_2)$; and (vi) $\xi = (T = \xi', \mathsf{fetch}(X \in T, R, Y))$. We next show that there exists an FO query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $Q_\xi \equiv_\mathcal{A} \xi$, for each of these six cases.

Case (i). We prove the case when $\xi = (\xi', \sigma_{X=c}(\xi'))$; the case when $\xi = (\xi', \sigma_{X=Y}(\xi'))$ is similar. Clearly, ξ' is an $(M-1)$-bounded query plan under $\mathcal{V}$ using $\mathcal{A}$. Hence, by the induction hypothesis, there exists $Q_{\xi'}$ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $Q_{\xi'} \equiv_\mathcal{A} \xi'$. Let Q_ξ be $Q_{\xi'} \wedge (x = c)$. Since $Q_{\xi'} \equiv_\mathcal{A} \xi'$, we also have that $Q_\xi \equiv_\mathcal{A} \xi$.

We next show that Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. To see this, consider the conjunction case (3) of topped queries given in Section 5. By the induction hypothesis, $\mathsf{covq}(Q_\varepsilon, Q_{\xi'}) =$ true. Therefore, case (3) applies here. That is, $\mathsf{covq}(Q_\varepsilon, Q_\xi) = \mathsf{covq}(Q_\varepsilon, Q_{\xi'}) = \mathsf{true}$ and $\mathsf{size}(Q_\varepsilon, Q_\xi) = \mathsf{size}(Q_\varepsilon, Q_{\xi'}) + 1$. We know by the induction hypothesis that the size is bounded by $(M-1) + 1 = M$. Hence, Q_ξ is indeed topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (ii). When $\xi = (\xi', \pi_Y(\xi'))$, ξ' is an $(M-1)$-bounded query plan under $\mathcal{V}$ using $\mathcal{A}$. Hence, by the induction hypothesis, there exists an FO query $Q_{\xi'}(\bar{z})$ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $\xi' \equiv_\mathcal{A} Q_{\xi'}$. Let Q_ξ be $\exists(\bar{z} \setminus \bar{y})\, Q_{\xi'}(\bar{z})$. From $Q_{\xi'} \equiv_\mathcal{A} \xi'$ it follows that $Q_\xi \equiv_\mathcal{A} \xi$ also holds.

We next verify that query Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. Observe that there are two cases of $\mathsf{covq}(Q_\varepsilon, Q_\xi)$ and $\mathsf{size}(Q_\varepsilon, Q_\xi)$, corresponding to cases (7a) and (7c) given in Section 5, respectively. Note that case (7b) does not apply here as the last operation of ξ is $\pi_Y(\xi')$ instead of a fetch as for case (7b). We show that in both cases, Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. (a) If $Q_{\xi'}$ is $R(\bar{z})$ and $R(\emptyset \to Z, N) \in \mathcal{A}$, then case (7a) of Section 5 applies here. Thus $\mathsf{covq}(Q_\varepsilon, Q_\xi) = \mathsf{true}$ and $\mathsf{size}(Q_\varepsilon, Q_\xi) = 1 \leqslant M$. Hence Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. (b) Otherwise, case (7c) applies here because ξ is an M-bounded query plan. Hence by the induction hypothesis, $\mathsf{covq}(Q_\varepsilon, Q_\xi) = \mathsf{covq}(Q_\varepsilon, Q_{\xi'}) = \mathsf{true}$ and $\mathsf{size}(Q_\varepsilon, Q_\xi) = \mathsf{size}(Q_\varepsilon, Q_{\xi'}) + 1 \leqslant M$. Thus Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (iii). When $\xi = (\xi_1, \xi_2, \xi_1 \times \xi_2)$, then ξ_1 is an M_1-bounded query plan and ξ_2 is an M_2-bounded query plan such that $M_1 + M_2 \leqslant M - 1$. Let $Q_{\xi_1}(\bar{x}_1) \equiv_{\mathcal{A}} \xi_1$ and $Q_{\xi_2}(\bar{x}_2) \equiv_{\mathcal{A}} \xi_2$ be the corresponding queries topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M_1)$ and $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M_2)$, respectively. Note that $\bar{x}_1 \cap \bar{x}_2 = \emptyset$. Consider $Q_\xi = Q_{\xi_1}(\bar{x}_1) \wedge Q_{\xi_2}(\bar{x}_2)$. Clearly, $Q_\xi \equiv_{\mathcal{A}} \xi$.

We show that query Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. Since $\mathsf{covq}(Q_\varepsilon, Q_{\xi_1})$ and $\mathsf{covq}(Q_\varepsilon, Q_{\xi_2})$ are both true, we know from the conjunction case (4b) of topped queries given in Section 5 that $\mathsf{covq}(Q_\varepsilon, Q_\xi) = \mathsf{true}$ as well. Furthermore, since $\bar{x}_1 \cap \bar{x}_2 = \emptyset$, $\mathsf{size}(Q_\varepsilon, Q_\xi)$ is defined in case (4b) as $2 \cdot \mathsf{size}(Q_\varepsilon, Q_\varepsilon) + \mathsf{size}(Q_\varepsilon, Q_{\xi_1}) + \mathsf{size}(Q_\varepsilon, Q_{\xi_2}) + 1$, which is bounded by $M_1 + M_2 + 1 \leqslant M$. Therefore, query Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Case (iv). The case when $\xi = (\xi_1, \xi_2, \xi_1 \cup \xi_2)$ is verified in the same way as the previous case, by using the disjunction case (5) of topped queries specified in Section 5.

Case (v). When $\xi = (\xi_1, \xi_2, \xi_1 \setminus \xi_2)$, the case is handled in the same way as case (iii), by using the negation case (6a) given in Section 5.

Case (vi). When $\xi = (S = \xi', \mathsf{fetch}(X \in S, R, Z))$, since ξ is an M-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$, we know that there exists an access constraint $R(X \to Z', N)$ in $\mathcal{A}$ such that $Z \subseteq X \cup Z'$ and as before, ξ' is an $(M-1)$-bounded query plan using $\mathcal{V}$ under $\mathcal{A}$. In addition, ξ' must have bounded output. Hence, the induction hypothesis applies here. Let $Q_{\xi'}(\bar{x})$ be a query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M-1)$ such that $Q_{\xi'}(\bar{x}) \equiv_{\mathcal{A}} \xi'$ and consider $Q_\xi(\bar{x}, \bar{z}) = Q_{\xi'}(\bar{x}) \wedge \exists \bar{u}\, R(\bar{x}, \bar{z}, \bar{u})$. Clearly, $Q_\xi(\bar{x}, \bar{z}) \equiv_{\mathcal{A}} \xi$.

We next verify that query Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. This follows from the conjunction case (4a) given in Section 5. Indeed, by the induction hypothesis we have that $\mathsf{covq}(Q_\varepsilon, Q_{\xi'}) = \mathsf{true}$. Furthermore, $Q_{\xi'}(\bar{x})$ has bounded output. Thus by the definition of topped queries in case 4(a), we have that $\mathsf{covq}(Q_\varepsilon, Q_\xi) = \mathsf{true}$ and $\mathsf{size}(Q_\varepsilon, Q_\xi) = \mathsf{size}(Q_\varepsilon, Q_{\xi'}) + 1$ bounded by M. Hence Q_ξ is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

(2) Every query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. We show that every query Q topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ indeed has a $\mathsf{size}(Q_\varepsilon, Q)$-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$. The statement we will prove is as follows:

if $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{covq}(Q_s, Q) = \mathsf{true}$ *and* Q_s *has a* $\mathsf{size}(Q_\varepsilon, Q_s)$*-bounded plan,*
then $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{true}$ *and* $Q_s \wedge Q$ *has a* $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$*-bounded plan.*

For if this holds, then Q has an M-bounded plan if it is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. Indeed, when Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$, $\mathsf{covq}(Q_\varepsilon, Q) = \mathsf{true}$ and $\mathsf{size}(Q_\varepsilon, Q) \leqslant M$. Since $\mathsf{covq}(Q_\varepsilon, Q_\varepsilon) = \mathsf{true}$ and Q_ε has a 0-bounded plan, by the statement, $Q_\varepsilon \wedge Q = Q$ has a $\mathsf{size}(Q_\varepsilon, Q_\varepsilon \wedge Q) = \mathsf{size}(Q_\varepsilon, Q)$-bounded plan, *i.e.,* an M-bounded plan. That is, Q has an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$ if Q is topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$.

Below we prove the statement by induction on the structure of Q. In the sequel, for a tuple $\bar{x}$ of variable, we denote by X its corresponding set of attributes, and vice versa.

<u>*Base case*</u>. We first show that the statement holds when Q has one of the following forms: (b1) $z = c$; (b2) a view $V(\bar{z})$ in $\mathcal{V}$; or (b3) a relation $\exists w\, R(\bar{w}, \bar{z})$.

Case (b1). For base case (b1), *i.e.,* when Q is $z = c$, if $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{covq}(Q_s, Q) = \mathsf{true}$ and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan (say ξ_s), then consider plan $\xi = (\xi_s, \sigma_{z=c}(\xi_s))$. Since $Q_s \equiv_{\mathcal{A}} \xi_s$, we have that $Q_s \wedge Q \equiv_{\mathcal{A}} \xi$. Since $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, we know that case (3) of topped queries specified in Section 5 can apply to $Q_s \wedge Q$. Hence $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{true}$ and $Q_s \wedge Q$ has a $(|\xi_s| + 1)$-bounded plan, where $|\xi_s| + 1 \leqslant \mathsf{size}(Q_\varepsilon, Q_s) + 1 = \mathsf{size}(Q_\varepsilon, Q_s \wedge (z = c))$. That is, the statement holds for case (b1).

Case (b2). For base case (b2), *i.e.,* when Q is a view $V(\bar{z})$, if $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{covq}(Q_s, Q) = \mathsf{true}$ and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan (say ξ_s), then we have the following plans for $Q_s \wedge Q$. For $Q_s(\bar{x})$ and $V(\bar{z})$, (i) if $\bar{x} \cap \bar{z} = \emptyset$, then let plan ξ be ($T_1 = \xi_s$, $T_2 = V(\bar{z})$, $T_3 = T_1 \times T_2$); and (ii) if $\bar{x} \cap \bar{z} = \bar{w} \neq \emptyset$, then let plan ξ be ($T_1 = \xi_s$, $T_2 = V(\bar{z})$, $T_3 = \rho(T_2)$, $T_4 = T_1 \times T_3$, $T_5 = \sigma_{T_1[W]=T_3[W]}(T_4)$, $T_6 = \pi_{T_1[X],T_3[Z \setminus W]}(T_5)$). In both cases, $\xi \equiv_{\mathcal{A}} Q_s \wedge Q$ when $\xi_s \equiv_{\mathcal{A}} Q_s$. Moreover, note that by case (2) of Section 5 we have that $\mathsf{covq}(Q_\varepsilon, V(\bar{z})) = \mathsf{true}$, since $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{true}$ following case (4b) of Section 5 we also have that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge V(\bar{z}))$ = true and $\mathsf{size}(Q_\varepsilon, Q_s \wedge V(\bar{z})) = \mathsf{size}(Q_\varepsilon, Q_\varepsilon) + \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_\varepsilon, V(\bar{z})) + \lambda_{(\bar{x},\bar{z})} \geqslant |\xi|$ in both cases (i) and (ii) (recall $\lambda_{(\bar{x},\bar{z})}$ from case (4b); note that $\lambda_{(\bar{x},\bar{z})} = 1$ for case (i) and $\lambda_{(\bar{x},\bar{z})} = 4$ for case (ii)). Therefore, the statement holds for case (b2).

Case (b3). For base case (b3), *i.e.,* when Q is $\exists \bar{w} R(\bar{w}, \bar{z})$, if $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{covq}(Q_s, Q) = \mathsf{true}$ and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan ξ_s, observe the following. Given that $\mathsf{covq}(Q_s, Q)$ = true, from cases (7a-7b) of Section 5, we know that either (i) $R(\emptyset \to Z, N) \in \mathcal{A}$ or (ii) $R(X \to Z', N) \in \mathcal{A}$, $X \cup Z' = Z$ and $Q_s(\bar{x})$ has bounded output under $\mathcal{A}$.

First consider case (i). We distinguish two cases: $\bar{x} \cap \bar{z} = \emptyset$, and $\bar{x} \cap \bar{z} = \bar{w}' \neq \emptyset$. When $\bar{x} \cap \bar{z} = \emptyset$, let ξ = ($T_1 = \xi_s$, $T_2 = \mathsf{fetch}(\emptyset, R, Z)$, $T_3 = T_1 \times T_2$). Since $\xi_s \equiv_{\mathcal{A}} Q_s$, we have that $\xi \equiv_{\mathcal{A}} Q$. Observe that by case (7a) of Section 5, $\mathsf{covq}(Q_\varepsilon, Q) = \mathsf{true}$. In addition, $\mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{true}$ by the condition of the statement. Therefore, case (4b) specified in Section 5 applies to $Q_s \wedge Q$. Hence $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{true}$, $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$ = $\mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_\varepsilon, Q) + 1 = \mathsf{size}(Q_\varepsilon, Q_s) + 2 \geqslant |\xi|$. That is, $Q_s \wedge Q$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$-bounded plan. For the case when $\bar{x} \cap \bar{z} = \bar{w}' \neq \emptyset$, one can verify that $Q_s(\bar{x}) \wedge Q(\bar{z})$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$-bounded plan along the same lines as above.

Next consider case (ii). Since Q_s has bounded output under $\mathcal{A}$ and $\mathsf{covq}(Q_\varepsilon, Q_s)$ = true, case (4a) given in Section 5 applies to $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q)$ here. Hence $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{true}$. Consider a plan ξ = ($T_1 = \xi_s$, $T_2 = \mathsf{fetch}(X \in T_1, R, Z')$). Since $\xi_s \equiv_{\mathcal{A}} Q_s$, we have that $\xi \equiv_{\mathcal{A}} Q_s \wedge Q$. Hence $|\xi| = |\xi_s| + 1 \leqslant \mathsf{size}(Q_\varepsilon, Q_s) + 1 = \mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$ by case (4a) of Section 5. That is, $Q_s \wedge Q$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$-bounded plan.

Induction step. Assume that the statement holds for sub-queries of a topped query Q. Below we show that the statement also holds for Q itself, by analyzing the structure of $Q(\bar{z})$ as follows, corresponding to the different cases presented in Section 5. We number the cases accordingly in the proof below.

(3) $Q(\bar{z})$ is $Q'(\bar{z}) \wedge (x = c)$. Since $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, we know that $\mathsf{covq}(Q_s, Q') = \mathsf{true}$ as well. Since $\mathsf{covq}(Q_\varepsilon, Q_s)$ = true and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan, by the induction hypothesis, $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q') = \mathsf{true}$ and $Q_s \wedge Q'$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q')$-bounded plan ξ'. Thus, by the definition of $\mathsf{covq}(\cdot, \cdot)$ in case (3) in Section 5, we know that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge (Q' \wedge (x = c))) = \mathsf{true}$. Moreover, $Q_s \wedge Q$ has a bounded plan ξ = ($T_1 = \xi'$, $T_2 = \sigma_{X=c} T_1$) and $|\xi| = |\xi'| + 1 \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q') + 1 = \mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$. That is, the statement holds for $Q(\bar{z})$. The cases when Q is $Q' \wedge (x = y)$, $Q' \wedge (x \neq y)$ or $Q' \wedge (x \neq c)$ can be verified in the same way.

(4) $Q(\bar{z})$ is $Q_1(\bar{z}_1) \wedge Q_2(\bar{z}_2)$. There are three cases (4a), (4b) and (4c) of topped queries given in Section 5 when $\mathsf{covq}(Q_s, Q(\bar{z})) = \mathsf{true}$. We verify these cases one by one below.

Case (4a). For case (4a) of topped queries specified in Section 5, when $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, we have that $\mathsf{covq}(Q_s, Q_1(\bar{z}_1)) = \mathsf{true}$, $Q_2(\bar{z}_2)$ is a relation $\exists w\, R(\bar{z}_1, \bar{z}'_2, \bar{w})$, $R(Z_1 \to Z'_2, N) \in \mathcal{A}$ with $Z_1 \cup Z'_2 = Z_2$, and $Q_s \wedge Q_1$ has bounded output under $\mathcal{A}$. Now consider $Q_s \wedge Q$ = $Q_s \wedge (Q_1 \wedge Q_2)$. Since $\mathsf{covq}(Q_s, Q_1) = \mathsf{covq}(Q_\varepsilon, Q_s) = \mathsf{true}$ and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan, by the induction hypothesis we know that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_1)$ = true and $Q_s \wedge Q_1$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1)$-bounded plan ξ_{s1}. In addition, ξ_{s1} has bounded output. Now case (4c) in Section 5 applies to $Q_s \wedge (Q_1 \wedge Q_2)$ to handle multiple conjuncts. Thus $\mathsf{covq}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2))$ = $\mathsf{covq}(Q_\varepsilon, Q_s) \wedge \mathsf{covq}(Q_s, Q_1 \wedge Q_2)$ = true.

Consider plan ξ = ($T_1 = \xi_{s1}$, $T_2 = \mathsf{fetch}(T_1, R, Z_2')$). Note that $\xi \equiv_{\mathcal{A}} Q_s \wedge Q$ because $\xi_{s1} \equiv_{\mathcal{A}} Q_s \wedge Q_1$. Since T_{s1} is of bounded size, ξ is a $(|\xi_{s1}| + 1)$-bounded plan, where $(|\xi_{s1}| + 1) \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1) + 1 = \mathsf{size}(Q_\varepsilon, (Q_s \wedge Q_1) \wedge Q_2) \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$. That is, the statement holds for Q when Q falls in case (4a).

Case (4b). For case (4b) of topped queries of Section 5, when $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, we have that $\mathsf{covq}(Q_s, Q_1) = \mathsf{covq}(Q_s, Q_2) = \mathsf{true}$. To be more specific, we distinguish four cases: (i) $\mathsf{covq}(Q_\varepsilon, Q_1)$ = $\mathsf{covq}(Q_\varepsilon, Q_2)$ = true, (ii) $\mathsf{covq}(Q_\varepsilon, Q_1)$ = true and $\mathsf{covq}(Q_\varepsilon, Q_2)$ = false, (iii) $\mathsf{covq}(Q_\varepsilon, Q_1)$ = false and $\mathsf{covq}(Q_\varepsilon, Q_2)$ = true, and (iv) $\mathsf{covq}(Q_\varepsilon, Q_1)$ = $\mathsf{covq}(Q_\varepsilon, Q_2)$ = false. Assume $\bar{z}_1 \wedge \bar{z}_2 = \emptyset$. For case (i), since $\mathsf{covq}(Q_\varepsilon, Q_i) = \mathsf{true}(i \in \{1, 2\})$, by the induction hypothesis, there are $\mathsf{size}(Q_\varepsilon, Q_i)$-bounded plans ξ_i for Q_i. Let ξ_s be the $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan for Q_s Now consider plan $\xi = (T_1 = \xi_1, T_2 = \xi_2, T_3 = T_1 \times T_2, T_4 = \xi_s, T_5 = T_3 \times T_4)$. Then $\xi \equiv_{\mathcal{A}} \xi$. Note that $|\xi| = |\xi_1| + |\xi_2| + |\xi_s| + 2 \leqslant \mathsf{size}(Q_\varepsilon, Q_1) + \mathsf{size}(Q_\varepsilon, Q_2) + \mathsf{size}(Q_\varepsilon, Q_s) + 2$. Hence $\mathsf{size}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2)) = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_\varepsilon, Q_1 \wedge Q_2) + 1 = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_\varepsilon, Q_1) + \mathsf{size}(Q_\varepsilon, Q_2) + 1 + 1 \geqslant |\xi|$. For case (ii), since $\mathsf{covq}(Q_\varepsilon, Q_1) = \mathsf{true}$ and $\mathsf{covq}(Q_s, Q_2) = \mathsf{true}$, by the induction hypothesis, we know that Q_1 has a $\mathsf{size}(Q_\varepsilon, Q_1)$-bounded plan ξ_1 and $Q_s \wedge Q_2$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_2)$-bounded plan ξ_{s2} (note that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_2) = \mathsf{covq}(Q_\varepsilon, Q_s) \wedge \mathsf{covq}(Q_s, Q_2) = \mathsf{true}$). Consider plan $\xi = (T_1 = \xi_1, T_2 = \xi_{s2}, T_3 = T_1 \times T_2)$. Then $\xi \equiv_{\mathcal{A}} Q_1 \wedge (Q_s \wedge Q_2) = Q_s \wedge Q$ and $|\xi| \leqslant \mathsf{size}(Q_\varepsilon, Q_1) + \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_2) + 1 = \mathsf{size}(Q_\varepsilon, Q_1) + \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_2) + 1$. Note that $\mathsf{size}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2)) = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1 \wedge Q_2) = 3 * \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1) + \mathsf{size}(Q_s, Q_2) + 1$. In addition, one can easily verify that, when $\mathsf{covq}(Q_\varepsilon, Q) = \mathsf{covq}(Q_s, Q) = \mathsf{true}$, $\mathsf{size}(Q_\varepsilon, Q) \leqslant \mathsf{size}(Q_s, Q)$, by induction on Q. Hence $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q) \geqslant |\xi|$. Similarly for case (iii). For case (iv), from $\mathsf{covq}(Q_\varepsilon, Q_i) \neq \mathsf{true}$ and $\mathsf{covq}(Q_s, Q_i) = \mathsf{true}$ and the induction hypothesis we know that $Q_s \wedge Q_i$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_i)$-bounded plan ξ_{si}, for $i \in \{1, 2\}$. Consider plan $\xi = (T_1 = \xi_{s1}, T_2 = \xi_{s2}, T_3 = T_1 \times T_2)$. Then $\xi \equiv_{\mathcal{A}} (Q_s \wedge Q_1) \wedge (Q_s \wedge Q_2) = Q_s \wedge Q$. Note that $|\xi| \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1) + \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_2) + 1 = 2 * \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1) + \mathsf{size}(Q_s, Q_2) + 1$. Hence $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1 \wedge Q_2) = \mathsf{size}(Q_\varepsilon, Q_s) + 2\mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1) + \mathsf{size}(Q_s, Q_2) + 1 \geqslant |\xi|$. Similarly, one can verify the case when $\bar{z}_1 \cap \bar{z}_2 \neq \emptyset$. Furthermore,$\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{true}$ since $\mathsf{covq}(Q_\varepsilon, Q_s) \wedge \mathsf{covq}(Q_s, Q) = \mathsf{true}$. Therefore, we have that the statement holds on Q in case (4b).

Case (4c). When Q falls in case (4c) of topped queries in Section 5, from $\mathsf{covq}(Q_s, Q_1 \wedge Q_2) = \mathsf{true}$ we know that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_1) = \mathsf{true}$ and $\mathsf{covq}(Q_s \wedge Q_1, Q_2) = \mathsf{true}$. By the induction hypothesis, from $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_1) = \mathsf{true}$ we have that $Q_s \wedge Q_1$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1)$-bounded plan ξ_{s1}. Hence, further by the induction hypothesis, from $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_1) = \mathsf{covq}(Q_s \wedge Q_1, Q_2) = \mathsf{true}$ and that $Q_s \wedge Q_1$ has plan ξ_{s1}, we have that $(Q_s \wedge Q_1) \wedge Q_2$ has a $\mathsf{size}(Q_\varepsilon, (Q_s \wedge Q_1) \wedge Q_2)$-bounded plan ξ, by treating $(Q_s \wedge Q_1)$ as a "new Q_s". Note that ξ is also a plan for $Q_s \wedge (Q_1 \wedge Q_2)$. Observe that $|\xi| \leqslant \mathsf{size}(Q_\varepsilon, (Q_s \wedge Q_1) \wedge Q_2)$ = $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1) + \mathsf{size}(Q_s \wedge Q_1, Q_2)$ = $\mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1) + \mathsf{size}(Q_s \wedge Q_1, Q_2)$ (by case (4c)). Therefore, by case (4c) $\mathsf{size}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2)) = \mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1 \wedge Q_2)$ = $\mathsf{size}(Q_\varepsilon, Q_s) + \mathsf{size}(Q_s, Q_1) + \mathsf{size}(Q_s \wedge Q_1, Q_2) \geqslant |\xi|$. Since $\mathsf{covq}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2)) = \mathsf{covq}(Q_\varepsilon, Q_s) \wedge \mathsf{covq}(Q_s, Q_1 \wedge Q_2) = \mathsf{true}$, the statement holds on Q when Q is in case (4c).

A $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$-bounded plan in this case can be constructed along the same lines as its counterpart for (4b) above, distinguishing the case when there exist common variables in $Q_s \wedge Q_1$ and Q_2 from the case when they contain disjoint variables.

Remark. Note that when Q = $Q_1 \wedge Q_2$, to compute $\mathsf{covq}(Q_s, Q)$, we need to compute both $\mathsf{covq}(Q_s, Q_2)$ (case (4b)) and $\mathsf{covq}(Q_s \wedge Q_1, Q_2)$ (case (4c)). When Q_2 is $Q_{21} \wedge Q_{22}$, we need to compute $\mathsf{covq}(Q_s, Q_{22})$, $\mathsf{covq}(Q_s \wedge Q_1, Q_{22})$, $\mathsf{covq}(Q_s \wedge Q_{21}, Q_{22})$ and $\mathsf{covq}(Q_s \wedge Q_1 \wedge Q_{21}, Q_{22})$. In the worst case, we test $2^{|Q_2|}$ many different cases. Hence we restrict the size of Q_2 to bound the number of expansions of Q_s when computing $\mathsf{covq}(Q_\varepsilon, Q)$ to ensure that $\mathsf{covq}(\cdot, \cdot)$ is computable in PTIME (statement (3) of Theorem 5.1), although this has no impact on the statement we are proving now.

(5) $Q(\bar{z})$ is $Q_1(\bar{z}) \vee Q_2(\bar{z})$. The case when Q is $Q_1 \vee Q_2$ is verified in the same way as for case (4b) above.

(6) $Q(\bar{z})$ is $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z})$. When Q is $Q_1 \wedge \neg Q_2$ and $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, there are two cases corresponding to cases (6a) and (6b) given in Section 5, respectively.

Case (6a). The statement can be verified in the same way as case (4b) above.

Case (6b). Since $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, we have that $\mathsf{covq}(Q_s, Q_1) = \mathsf{true}$ and $\mathsf{covq}(Q_s, Q_1 \wedge Q_2) = \mathsf{true}$. By the induction hypothesis, $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q_1) = \mathsf{true}$ and $Q_s \wedge Q_1$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1)$-bounded plan ξ_{s1}; similarly, $Q_s \wedge (Q_1 \wedge Q_2)$ has a $\mathsf{size}(Q_\varepsilon, Q_s \wedge (Q_1 \wedge Q_2))$-bounded plan ξ_{s12}. Thus $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{covq}(Q_\varepsilon, Q_s) \wedge \mathsf{covq}(Q_s, Q) = \mathsf{true}$, and $Q_s \wedge Q$ has a plan $\xi = (\xi_{s1}, \xi_{s12}, \xi_{s1} - \xi_{s12})$. Since $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z}) = Q_1(\bar{z}) \wedge \neg(Q_1(\bar{z}) \wedge Q_2(\bar{z}))$, ξ is a plan of $Q_s \wedge Q$. Moreover, $|\xi| \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1) + \mathsf{size}(Q_\varepsilon, Q_s \wedge Q_1 \wedge Q_2) + 1 = \mathsf{size}(Q_\varepsilon, Q_s \wedge Q)$.

Thus the statement holds for case (6).

(7) $Q(\bar{z})$ is $\exists w\, Q'(\bar{w}, \bar{z})$. When Q falls in case (7) of topped queries in Section 5, *i.e.,* $Q(\bar{z}) = \exists w\, Q'(\bar{w}, \bar{z})$, we only need to consider case (7c) when Q' is not a relation, since cases (7a) and (7b) have already been covered in the base step. In case (7c), when $\mathsf{covq}(Q_s, Q) = \mathsf{true}$, $\mathsf{covq}(Q_s, Q')$ is also true. Thus by the induction hypothesis, $\mathsf{covq}(Q_\varepsilon, Q_s(\bar{x}) \wedge Q'(\bar{w}, \bar{z})) = \mathsf{true}$ and $Q_s(\bar{x}) \wedge Q'(\bar{w}, \bar{z})$ has a $\mathsf{size}(Q_\varepsilon, Q'(\bar{w}, \bar{z}))$-bounded plan ξ'. Consider plan $\xi = (T_1 = \xi', T_2 = \pi_Z(T_1))$. Then $\xi \equiv_{\mathcal{A}} Q$ since $\xi' \equiv_{\mathcal{A}} Q'$. That is, $Q_s \wedge Q$ has a $(|\xi'| + 1)$-bounded plan, where $(|\xi'| + 1) \leqslant \mathsf{size}(Q_\varepsilon, Q_s \wedge Q') + 1 = \mathsf{size}(Q_\varepsilon, Q)$. Observe that $\mathsf{covq}(Q_\varepsilon, Q_s \wedge Q) = \mathsf{covq}(Q_\varepsilon, Q_s \wedge Q') = \mathsf{true}$. Thus the statement holds for case (7).

The proof above gives a construction of bounded rewriting of Q using $\mathcal{V}$ under $\mathcal{A}$, by defining the bounded rewriting for each case of computing $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$. To show that the construction is in PTIME in M, $|Q|$, $|\mathcal{A}|$ and $|\mathcal{V}|$, we only need to show that the computation of $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$ can be done in polynomially many induction steps (*i.e.,* applications of the 7 cases above). This is verified below.

(3) It is in PTIME to check whether FO queries are topped by $(\mathcal{R}, \mathcal{V}, A, M)$ with a PTIME oracle for checking output boundedness. It suffices to show that both functions $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$ are polynomial in $|Q|$, $|Q_s|$, $|\mathcal{A}|$ and $|\mathcal{V}|$. Below we verify this for $\mathsf{covq}(Q_s, Q)$; the proof for $\mathsf{size}(Q_s, Q)$ is similar. Observe the following.

— At most $O(|Q|)$ induction steps are needed for computing $\mathsf{covq}(Q_s, Q)$, where each induction step is an application of one of the seven cases given in the definition of $\mathsf{covq}(Q_s, Q)$ in Section 5. To see this, observe the following. (i) When only cases (1), (2), (3), (5), (4a), 4(b), 6(a) and (7) are involved, $\mathsf{covq}(Q_s, Q)$ can be computed within $|Q|$ induction steps because each application of such cases decreases $|Q|$ by 1 while keeping $|Q_s|$ unchanged. (ii) For case (4c) (when $Q = Q_1 \wedge Q_2$), we need to check $2^{|Q_2|}$ possible expansions of Q_s to compute $\mathsf{covq}(Q_s, Q)$, as remarked in the the proof of statement (2) for case (4c) above. Since $|Q_2|$ is bounded by a predefined constant K (see Section 5), the checking can be done in PTIME. Moreover, Q_s can be expanded at most $O(|Q|)$ times, and each step corresponds to an induction step. That is, the total number of induction steps remains bounded by $O(|Q|)$. This is similar when case (6b) (when $Q = Q_1 \wedge \neg Q_2$) is also concerned.

— Each induction step is in PTIME in $|Q|$, $|\mathcal{V}|$ and $|\mathcal{A}|$ when a PTIME oracle for checking output boundedness is available (Theorem 5.2). This is because (i) $|Q_s|$ can be increased by no larger than $|Q|$ when computing $\mathsf{covq}(Q_s, Q)$ and $\mathsf{size}(Q_s, Q)$; and (ii) each step can be done by syntactically checking Q_s, Q, $\mathcal{A}$ and $\mathcal{V}$, and for output boundedness checking in cases (4a) and (7b).

Thus it is in PTIME in $|Q|$, $|\mathcal{A}|$ and $|\mathcal{V}|$ to decide whether $\mathsf{covq}(Q_\varepsilon, Q) = \mathsf{true}$. Similarly, it takes PTIME in M, $|Q|$, $|\mathcal{A}|$ and $|\mathcal{V}|$ to check whether $\mathsf{size}(Q_\varepsilon, Q) \leqslant M$. Taken together with the constructive proof given in (2) above, these show that it takes PTIME to generate an M-bounded rewriting using $\mathcal{V}$ for each query topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$. □