Bounded Query Rewriting Using Views

Yang Cao 1,3
Wenfei Fan 1,3
Floris Geerts 2
Ping Lu 1
1University of Edinburgh, 2University of Antwerp, 3Beihang University

ABSTRACT

A query \( Q \) has a bounded rewriting using a set of views if there exists a query \( Q' \) expressed in the same language as \( Q \), such that given a dataset \( D \), \( Q(D) \) can be computed by \( Q' \) that accesses only cached views and a small fraction \( D_Q \) of \( D \). We consider datasets \( D \) that satisfy a set of access constraints, a combination of cardinality constraints and associated indices, such that the size \( |D_Q| \) of \( D_Q \) and the time to identify \( D_Q \) are independent of \( |D| \), no matter how big \( D \) 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, from \( \Sigma_2 \)-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 core subclass of such queries without sacrificing the expressive power, and can be checked in PTIME.

Keywords: Bounded rewriting; big data; complexity

1. INTRODUCTION

To make query answering feasible in big datasets, practitioners have been studying scale independence [4–6]. The idea is to compute the answers \( Q(D) \) to a query \( Q \) in a dataset \( D \) by accessing a bounded amount of data in \( D \), no matter how big the underlying \( D \) is.

This idea was formalized in [17, 18]. As suggested in [18], 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 [17] if for all datasets \( D \) that satisfy \( \mathcal{A} \), \( Q(D) \) can be computed from a fraction \( D_Q \) of \( D \), and the time for identifying and fetching \( D_Q \), and hence the size \( |D_Q| \) of \( D_Q \) are independent of \( |D| \). We identify \( D_Q \) by reasoning about the cardinality constraints in \( \mathcal{A} \), and fetch \( D_Q \) by using the indices of \( \mathcal{A} \).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

PODS’16, June 26–July 01, 2016, San Francisco, CA, USA
© 2016 ACM. ISBN 978-1-4503-4191-2/16/06...$15.00
DOI: http://dx.doi.org/10.1145/2802251.2802294

Bounded evaluation has proven useful [10–12]. Experi-menting with real-life data, we find that under a couple of hundreds of access constraints, 77% of randomly generated conjunctive queries [12], 67% of relational algebra queries [10], and 60% of graph pattern queries [11] are boundedly evaluable. Query plans for such queries outperform commercial query engines by 3 orders of magnitude, and the gap gets larger on bigger data. The results on CDR (call detail record) queries of one of our industry collaborators are even better: bounded evaluation improves 90% of their queries from 25 times to 5 orders of magnitude.

As an example of bounded evaluable, consider a Graph Search query of Facebook [15]: 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 [16]. Given this, we can answer the query by accessing 470000 tuples [10], as opposed to billions of user tuples and trillions of friend tuples in the Facebook dataset [22].

Still, many queries are not boundedly evaluable. Can we do better for such queries \( Q \)? An approach that has proven effective by practitioners is by making use of views [6]. The idea is to select and materialize a set \( \mathcal{V} \) of small views, and answer \( Q \) in a dataset \( D \) by using views \( \mathcal{V}(D) \) and an additional small fraction of \( D \). That is, we cache \( \mathcal{V}(D) \) with fast access, and compute \( Q(D) \) by using \( \mathcal{V}(D) \) and by restricting costly I/O operations to (possibly big) \( D \). Many queries that are not boundedly evaluable can be efficiently answered using small views and bounded access to \( D \) [6].

Example 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 \( R_0 \) consisting of four relations:

- person(pid, name, affiliation),
- movie(mid, mname, studio, release),
- rating(mid, rank) for ranks of movies, and
- like(pid, id, type), indicating that person pid likes item id of type, including but not limited to movies.

Over \( R_0 \), \( Q_0 \) is written as a conjunctive query:

\[
Q_0(mid) = \exists x_p, x_p', y_m ( \text{person}(x_p, x_p', \text{"NASA"}) \land \text{movie}(mid, y_m, \text{"Universal", "2014"}) \land \text{like}(x_p, mid, \text{"(movie") \land rating(mid, 5)}).)
\]

Consider a set \( \mathcal{A}_0 \) of two access constraints: (a) \( \varphi_1 = \text{movie}(\text{studio, release}) \rightarrow \text{mid, } N_0 \), stating that each studio releases at most \( N_0 \) movies each year, where \( N_0 \leq 100 \) is obtained by aggregating \( \mathcal{A}_0 \) instances; an index is built on movie such that given any (studio, release) value, it returns
Bounded queries con Acyclic rewritings. We investigate VBRP(L) when L ranges over conjunctive queries (CQ), i.e., SPJ, unions of conjunctive queries (UCQ, i.e., SPJ), positive FO queries \( \exists \text{FO}^+ \), select-project-join-union queries) and first-order logic queries (FO, the full relational algebra). We show that VBRP is \( \Sigma_3^P \)-complete for CQ, UCQ and \( \exists \text{FO}^+ \); but it becomes undecidable for FO. In addition, we explore the impact of various parameters \((\mathcal{R}, \mathcal{M}, \mathcal{A} \text{ and } \mathcal{V})\) of VBRP on its complexity.

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

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

(a) every \( \text{FO} \) query that has an \( \mathcal{M} \)-bounded rewriting using \( \mathcal{V} \) is equivalent to a query topped by \((\mathcal{R}, \mathcal{V}, \mathcal{A}, \mathcal{M})\);

(b) every query topped by \((\mathcal{R}, \mathcal{V}, \mathcal{A}, \mathcal{M})\) has an \( \mathcal{M} \)-bounded rewriting in \( \text{FO} \) using \( \mathcal{V} \); and

(c) \( \text{PTIME} \) in \( \mathcal{M}, |\mathcal{V}|, |\mathcal{R}|, |\mathcal{A}| \) to check whether \( Q \) is topped by \((\mathcal{R}, \mathcal{V}, \mathcal{A}, \mathcal{M})\), using an oracle to check whether views in \( V \) have bounded output (see below).

That is, topped queries make a core subclass of \( \mathcal{FO} \) queries with a bounded rewriting using \( \mathcal{V} \), without sacrificing their expressive power, along the same lines as range-safe queries for safe relational calculus (see, e.g., [1]). 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 \((\mathcal{R}, \mathcal{V}, \mathcal{A}, \mathcal{M})\) in \( \text{PTIME} \), by condition (c) above; if so, we can find a bounded rewriting 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 \( V \) have bounded output \( \mathcal{V}(D) \) over all datasets \( D \) that satisfy \( \mathcal{A} \), i.e., the size of \( |\mathcal{V}(D)| \) is bounded by a constant. This is to ensure bounded access to \( D \), since a query plan may filter and fetch data from \( D \) by using values from some views in \( \mathcal{V}(D) \). This problem is, not surprisingly, undecidable for \( \text{FO} \) (Section 3). In light of this, we develop effective syntax for \( \text{FO} \) queries with bounded output. That is, given \( \mathcal{A} \) and \( \mathcal{R} \), we identify a class of \( \mathcal{FO} \) queries, referred to as size-bounded queries, such that under \( \mathcal{A} \), an \( \mathcal{FO} \) view (query) over \( \mathcal{R} \) has bounded output if it is equivalent to a size-bounded \( \mathcal{FO} \), and it is in \( \text{PTIME} \) to check whether a query is size-bounded. We use this as a \( \text{PTIME} \) oracle when checking topped queries (condition (c)) above.

This work is an effort to give a formal treatment of scale independence with views, an approach that has already 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 effective use of bounded rewriting. A variety of techniques are used in the proofs, including characterizations, algorithms and a wide range of reductions.

Related work. We classify related work as follows.

Scale independence. The idea of scale independence originated from [5], 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 [4], an extension of SQL by allowing users to specify bounds on the amount of data accessed. As pointed out by [6], 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 [18]. A query $Q$ is defined to be scale independent in a dataset $D$ w.r.t. a bound $\Theta$ if there exists $D_Q \subseteq D$ such that $Q(D) = Q(D_Q)$ and $|D_Q| \leq \Theta$. Access constraints, a notion of $\bar{x}$-controllability and a set of rules were also introduced in [18], to deduce dependencies on attributes needed for computing $Q(D)$; these yield a sufficient condition to determine the scale independence of FO queries when variables $\bar{x}$ are instantiated. In addition, it studied the problem to decide whether for all instances $D$ of a relational schema, we can compute $Q(D)$ by accessing cached views and at most $\Theta$ tuples, in the absence of access constraints. The problem is NP-complete for FO and undecidable for Q. The notion of $\bar{x}$-controllability was extended to views, giving two simple sufficient conditions to decide the scale independence of query rewriting using views 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 is quite different from the notion of $\bar{x}$-controllability [18]. (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($Q$) is $\Sigma_2^p$-complete, in contrast to NP-complete [18]. (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 [18] via $\bar{x}$-controllability are sufficient but not necessary. Moreover, the rules of [18] 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 and VBRP were not studied in [4–6].

Bounded evaluability. The notion of bounded evaluability was proposed in [17], 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 FO, UCQ and $\exists FO^*$, and is undecidable for FO [17]. A notion of effective boundedness was studied for FO [12], based on a restricted form of query plans that can fetch all data before any relational operators start. It was shown [12] that effective boundedness is in PTIME for FO. It was also studied for graph pattern queries via simulation and subgraph isomorphism [11], 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 $|\nu(D)|$. (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 (see Section 2). In contrast, [17] 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 [17] are not very realistic, and worse yet, their FO plans may actually encode non-recursive datalog queries without union, which yield exponential-size queries when expressed in FO. It is because of the different notions of query plans adopted in this work and [17] that VBRP is $\Sigma_2^p$-complete for FO, while bounded evaluability is EXPSPACE-hard [17].

Effective syntax. There has been a host of work on effective syntax (e.g., [20, 35, 36]), which started decades ago to characterize safe relational queries up to equivalence. For bounded query evaluation, an effective syntax was proposed for FO [17], and another one for UCQ [10].

This work develops an effective syntax for bounded rewriting of FO queries using views under access constraints. 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 [17] is in PTIME to check for FO, but $\Pi_2^p$-complete for UCQ, and is not defined for UCQ. (b) Effective syntax for query rewriting is more intriguing than its counterpart for bounded evaluability [10]. As remarked earlier, we have to reason about the size $|\nu(D)|$ of cached views. It is further complicated by user-imposed bound on the size of query plans, which was not considered in [10]. (c) The class of effectively bounded queries of [12] does not make an effective syntax: not every boundedly evaluable FO is necessarily equivalent to an effectively bounded FO.

Query rewriting using views. Query rewriting has been extensively studied (e.g., [2, 3, 13, 28, 31, 32]; see [23, 27] for surveys). In contrast to conventional query rewriting using views, bounded rewriting requires controlled access to the underlying dataset $D$ under access constraints, in addition to cached $\nu(D)$. This makes the analysis more challenging. For instance, it is $\Sigma_2^p$-complete to decide whether there exists a bounded rewriting for FO with query views, as opposed to NP-complete in the conventional setting [28].

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 [8, 9, 14, 29, 30, 32] (see [7] for a survey). Proof-based methods for generating low-cost query plans under access patterns and integrity constraints are studied for FO and UCQ. Query rewriting using views under access patterns has been studied for FO and UCQ and UCQ with negated atoms under fixed views and integrity constraints [14].

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 $R(X \rightarrow Y, N)$, $X \cup Y$ 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 evaluability. (c) Bounded rewriting allows access to the underlying data with controlled I/O, which is prohibited in [14, 32]. An evidence of the difference, bounded FO rewriting using fixed views is $\Sigma_{2k+1}^p$-complete under fixed access constraints (Section 3), as opposed to NP-complete for rewriting using fixed views under access patterns [14]. (d) To the best of our knowledge, no prior work has studied effective syntax for bounded FO rewriting.
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 notations.

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 $U$ of data values, on which instances $D$ of $\mathcal{R}$ are defined. We use $|D|$ to denote the size of $D$, measured as the total number of tuples in $D$.

Access schema. Following [17], we define an access schema $A$ over a database schema $\mathcal{R}$ as a set of access constraints $\varphi = R(X \rightarrow Y, N)$, where $R$ is a relation schema in $\mathcal{R}$, $X$ and $Y$ are sets of attributes of $R$, and $N$ is a natural number. An instance $D$ of $\mathcal{R}$ satisfies $\varphi$ if

\begin{itemize}
  \item for any $X$-value $\bar{a}$ in $D$, $|D_Y(X = \bar{a})| \leq N$, where $D_Y(X = \bar{a})$ is the set $\{t[Y] \mid t \in D, t[X] = \bar{a}\}$; and
  \item there exists a function (index) that given an $X$-value $\bar{a}$, returns $D_Y(X = \bar{a})$ from $D$ in $O(N)$ time.
\end{itemize}

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, $A_0$ described in Example 1 is an access schema.

Note that functional dependencies (FDs) are a special case $R(X \rightarrow Y, 1)$ of access constraints, i.e., when bound $N = 1$, provided that an index is built from $X$ to $Y$. As shown in [10, 12], access constraints can be discovered from instances of $\mathcal{R}$, by extending mining tools for FDs with aggregates.

An instance $D$ of $\mathcal{R}$ satisfies access schema $A$, denoted by $D \models A$, if $D$ satisfies all the constraints in $A$.

Query classes. We express queries and views in the same language $L$, which is one of the following [1]:

\begin{itemize}
  \item conjunctive queries (CQ), built up from relation atoms $R(x)$ for $R \in \mathcal{R}$, and equality atoms $x = y$ or $x = c$ (for constant $c$), by closing them under conjunction $\land$ and existential quantification $\exists$;
  \item unions of conjunctive queries (UCQ) of the form $Q = Q_1 \cup \cdots \cup Q_k$, where $Q_i$ is a CQ for $i \in [1, k]$;
  \item positive existential FO queries (3FO$^+$), built from relation atoms and equality atoms by closing under $\land$, disjunction $\lor$ and existential quantification $\exists$; and
  \item first-order logic queries (FO), built from atomic formulas by closing them under $\land$, $\lor$, negation $\neg$, $\exists$ and universal quantification $\forall$.
\end{itemize}

Query plans. Following [33], we define evaluation plans for a query $Q$ using a set $V$ of views, both defined over a database schema $\mathcal{R}$. To simplify the definition, we write $Q$ in the relational algebra with projection $\pi$, selection $\sigma$, Cartesian product $\times$, union $\cup$, set difference $\setminus$ and renaming $\rho$.

A query plan for $Q$ using $V$, denoted by $\xi(Q, V, \mathcal{R})$, is a tree $T_\xi$ that satisfies the two conditions below:

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

\begin{itemize}
  \item $\{c\}$ for a constant in $Q$, if $u$ is a leaf of $T_\xi$;
  \item a view $V$ for $V \in V$, if $u$ is a leaf of $T_\xi$;
  \item $\text{fetch}(X \in S_j, R, Y)$, if $u$ has a single child $v$ labeled with $S_j = \delta_j$, and $S_j$ has attributes $X$;
\end{itemize}

(2) For each instance $D$ of $\mathcal{R}$, relations $S_i$’s are computed by $\delta_i$, bottom up in $T_\xi$ as usual [33]. More specifically, $\delta_i$ may (a) extract constants from $Q$, (b) access cached views $V(D)$, and (c) access $D$ via a fetch operation, which, for each $\bar{a} \in S_j$, retrieves $D_{XY}(X = \bar{a})$ from $D$; it may also be a relational operation ((d) and (e) above). Relation $S_i$ associated with the root of $T_\xi$ is the result of the computation.

\begin{itemize}
  \item (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
  \item (e) $S_j \times S_i$, $S_j \cup S_i$ or $S_j \setminus S_i$, if $u$ has two children $v$ and $v'$ labeled with $S_j = \delta_j$ and $S_i = \delta_i$, respectively.
\end{itemize}

Intuitively, given an instance $D$ of $\mathcal{R}$, relations $S_i$’s are computed by $\delta_i$, bottom up in $T_\xi$ as usual [33]. More specifically, $\delta_i$ may (a) extract constants from $Q$, (b) access cached views $V(D)$, and (c) access $D$ via a fetch operation, which, for each $\bar{a} \in S_j$, retrieves $D_{XY}(X = \bar{a})$ from $D$; it may also be a relational operation ((d) and (e) above). Relation $S_i$ associated with the root of $T_\xi$ is the result of the computation.

(2) For each instance $D$ of $\mathcal{R}$, the result $\xi(D)$ of applying $\xi(Q, V, \mathcal{R})$ to $D$ is the relation $S_i$ at root of $T_\xi$ computed as above. We require that $\xi(D) \subseteq Q(D)$.

The size of plan $\xi$ is the number of nodes in $T_\xi$. We use $D_Q$ to denote the bag of all tuples fetched for computing $\xi(D)$, i.e., the multiset that collects tuples in $D_{XY}(X = \bar{a})$ for all $\text{fetch}(X \in S_j, R, Y)$ when applying $\xi$ to $D$. Intuitively, it measures the amount of I/O operations to access $D$.

Example 2: A plan $\xi_0(Q_0, V_1, R_0)$ for $Q_0$ using view $V_1$ given in Example 1 is depicted in Fig. 1. Given an instance $D$ of $R_0$, (a) it fetches the set $S_1$ of mids of all movies released by Universal Studios in 2014, using constants in $Q_0$; (b) filters $S_1$ with mids in $V_1(D)$ via join, to get a subset $S_8$ of $S_1$ of movies liked by NASA folks; (c) fetches rating tuples using the mids of $S_8$; and finally, (d) finds the set $S_{11}$ of mids, which is precisely the answer $Q_0(D)$ to $Q_0$ in $D$.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{query_plan.png}
\caption{A query plan $\xi_0$ for $Q_0$ using view $V_1$}
\end{figure}

Bounded plans. To formalize bounded query rewriting, we bring into play access schema and a bound on the size of query plans. Consider an access schema $A$ defined over $\mathcal{R}$. A query plan $\xi(Q, V, \mathcal{R})$ is said to conform to $A$ if

\begin{itemize}
  \item (a) for each $\text{fetch}(X \in S_j, R, Y)$ operation in $\xi$, there exists an access constraint $R(X \rightarrow Y', N)$ in $A$ such that $Y \subseteq X \cup Y'$, and
  \item (b) there exists a constant $N_Q$ such that for all instances $D$ of $\mathcal{R}$ that satisfy $A$, $|D_Q| \leq N_Q$.
\end{itemize}

That is, while $\xi$ can access entire cached views, its access to the underlying $D$ must be via fetch operations only, by making use of the indices in the access constraints of $A$. Plan $\xi$ tells us how to retrieve $D_Q$ from $D$ such that $Q(D)$ is computed by using the data in $D_Q$ and $V(D)$ only. Better still, $D_Q$ is bounded: $|D_Q|$ is independent of possibly big $|D|$. The time for identifying and fetching $D_Q$ is also independent of $|D|$ (assuming that given an $X$-value $\bar{a}$, it takes $O(N)$ time to fetch $D_{XY}(X = \bar{a})$ from $D$ via the index for $R(X \rightarrow Y, N)$).
Given a natural number $M$, we say that $\xi(Q, V, R)$ is an $M$-bounded plan for $Q$ using $V$ under $A$ if (a) $\xi$ conforms to $A$, and (b) the size of $\xi$ is at most $M$.

Intuitively, $M$ 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 rewriting plans are 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 [17]. Hence we opt to let users specify $M$ based on their resources.

If $\xi(Q, V, R)$ is $M$-bounded under $A$, then for all datasets $D$ that satisfy $A$, we can efficiently answer $Q$ in $D$ by following $\xi$ and accessing a bounded amount of data from $D$.

**Example 3:** Plan $\xi_0$ shown in Fig. 1 is 11-bounded for $Q_0$ using $V_1$ under $A_0$. Indeed, (a) both fetch operations $(S_4$ and $S_5$) are controlled by the access constraints of $A_0$, and (b) for any instance $D$ of $R$, $\xi_0$ accesses at most $2N$ tuples from $D$, where $N_0$ is the constant in $\varphi_1$ of $A_0$, since $|S_4| \leq N_0$ by $\varphi_1$, and $|S_5| \leq N_0$ by $S_2 \subseteq S_5$ and constraint $\varphi_2$ on rating in $A_0$; and (c) eleven operations are conducted in total. Note that rating tuples in $D$ are fetched by using $S_8$, which is obtained by relational operations on $V_1(D)$ and $S_4$. While $V_1$ is not boundedly evaluability under $A_0$, the amount of data fetched from $D$ is independent of $|D|$.

**Bounded query rewriting.** We now formalize this notion. Consider a query $Q$ in a language $L$, a set $V$ of $L$-definable views, and an access schema $A$, all over the same database schema $R$. For a bound $M$, we say that $Q$ has an $M$-bounded rewriting in $L$ using $V$ under $A$, or simply a bounded rewriting using $V$ when $M$ and $A$ are clear from the context, if it has an $M$-bounded query plan $\xi(Q, V, R)$ under $A$ such that $\xi$ is a query plan in $L$, i.e., in each label $S_i = \delta_i$ of $\xi$, 
- if $L$ is CQ, then $\delta_i$ is a fetch, $\pi$, $\sigma$, $\times$, or $\rho$ operation;
- if $L$ is UCQ, $\delta_i$ can be fetch, $\pi$, $\sigma$, $\times$, $\rho$, or $\cup$, and moreover, for any node labeled $\xi$, all its ancestors in the tree $T_\xi$ of $\xi$ are also labeled with $\cup$; that is, $\cup$ is conducted at “the top level” only;
- if $L$ is $\exists FO^+$, then $\delta_i$ is fetch, $\pi$, $\sigma$, $\times$, $\cup$ or $\rho$; and
- if $L$ is FO, $\delta_i$ can be fetch, $\pi$, $\sigma$, $\times$, $\cup$, $\|$, or $\rho$.

One can verify that if $\xi$ is a plan in $L$, then there exists a query $Q_\xi$ in $L$ such that for all instances $D$ of $R$, $\xi(D) = Q_\xi(D)$ and moreover, the size $|Q_\xi|$ of $Q_\xi$ is linear in the size of $\xi$. Such query $Q_\xi$ is unique up to equivalence. We refer to $Q_\xi$ as the query expressed by $\xi$.

**Example 4:** The CQ $Q_0$ of Example 1 has a 11-bounded rewriting in CQ using $V_1$ under $A_0$. Indeed, $\xi_0$ of Fig. 1 is such a bounded plan, which expresses $Q_0(mid) = \exists y_m \left( \text{movie(mid, y_m, "Universal", "2014"} \right) \land V_1(mid) \land \text{rating(mid, 5)}$. It is a rewriting of $Q_0$ using $V_1$ in CQ.

Notations used in this paper are summarized in Table 1.

### 3. DECIDING BOUNDED REWRITING

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

**INPUT:** A database schema $R$, a natural number $M$ (in unary), an access schema $A$, a query $Q \in L$ and a set $V$ of $L$-definable views all defined on $R$.

**QUESTION:** Under $A$, does $Q$ have an $M$-bounded rewriting in $L$ using $V$?

While VBRP($L$) is important, it is nontrivial.

**Theorem 1:** Problem VBRP($L$) is

1. $\Pi^P_3$-complete when $L$ is CQ, UCQ or $\exists FO^+$; and
2. undecidable when $L$ is FO.

Below we first reveal the inherent complexity of VBRP($L$) by investigating problems embedded in it, and outline a proof of Theorem 1 for various $L$ (Section 3.1). We then investigate the impact of parameters $R$, $A$, $V$ and $M$ on the complexity of VBRP($L$) (Section 3.2).

### 3.1 The Bounded Rewriting Problem

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

**A-equivalence.** Consider a database schema $R$. Under an access schema $A$ over $R$, we say that two queries $Q_1$ and $Q_2$ defined over $R$ are $A$-equivalent, denoted by $Q_1 \equiv_A Q_2$, if for all instances $D$ of $R$ that satisfy $A$, $Q_1(D) = Q_2(D)$. 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 $D$ of $R$, $Q_1(D) = Q_2(D)$, regardless of whether $D \models A$. Indeed, if $Q_1 \equiv Q_2$ then $Q_1 \equiv_A Q_2$, but the converse does not hold. It is known that query equivalence for CQ is $\Pi^P_3$-complete (cf. [1]). In contrast, it has been shown that $A$-equivalence is harder (unless $P = NP$).

**Lemma 2** [17]: Given access schema $A$ and two queries $Q_1$ and $Q_2$, it is $\Pi^P_3$-complete to decide whether $Q_1 \equiv_A Q_2$, for $Q_1$ and $Q_2$ in CQ, UCQ or $\exists FO^+$.

Coming back to VBRP, for a query plan $\xi$ and a query $Q$, we need to check whether $\xi$ is a query plan for $Q$, i.e., whether $Q_\xi \equiv_A Q$, where $Q_\xi$ is the query expressed by $\xi$. This step is $\Pi^P_3$-hard for CQ. It is easy to show that the problem is undecidable when it comes to FO.

**Bounded output.** Another complication is introduced by views in $V$. To decide whether a query plan $\xi$ is bounded for a query $Q$ using $V$ under $A$, we need to verify that $\xi$ conforms to $A$. This may require us to check whether a view $V \in V$ has “bounded output.”
Example 5: Consider database schema $R_0$, query $Q_0$, and access schema $A_0$ defined in Example 1.

(a) Suppose that instead of $V_1$, a CQ view $V_2$ is given:

\[ V_2(pid) = \exists x_p \text{person}(pid, x_p, \text{"NASA"}). \]

Given an instance $D$ of $R_0$, $V_2(D)$ consists of people who work at NASA. Extend $A_0$ to $A_1$ by including $\varphi_2 = \text{like}(\text{pid}, \text{id}) \rightarrow (\text{pid, id, type})$, i.e., $\varphi_2$ is a key of relation like. Then $Q_0$ has a rewriting $Q_2$ using $V_2$:

\[ Q_2(\text{mid}) = \exists x_p, y_v (V_2(x_p) \land \text{like}(x_p, y_v, \text{"movies"}) \land \text{movie}(y_v, \text{mid}, \text{"Universal"}, \text{"2014"}) \land \text{rating}(\text{mid}, 5)). \]

One can verify that $Q_2$ is a bounded rewriting of $Q_0$ using $V_2$ under $A_1$ iff there exists a constant $N_1$ such that for all instances $D$ of $R$, if $D \models A_1$, then $[V_2(D)] \subseteq N_1$; that is, NASA has at most $N_1$ employees. For if it holds, then we can extract a set $S$ of at most $N_0$ mid tuples by leveraging constraint $\varphi_2$ of $A_1$ on movie, and select pairs $(pid, mid)$ from $V_2(D) \times S$ that are in a tuple $(pid, mid, \text{"movies"})$ in the like relation, by making use of constraint $\varphi_2$ given above. For each mid that passes the test, we check its rating via the index in $\varphi_2$, by accessing at most $N_0$ tuples in rating. Putting these together, we access at most $N_1 \cdot N_0 + N_0$ tuples from $D$. Conversely, if the output of $V_2(D)$ is not bounded, then $Q$ has no bounded rewriting using $V_2$ under $A_1$.

(b) In contrast, for rewriting some queries, we do not have to check whether a view has bounded output. Consider a rewriting $Q(x) = Q_2(x) \land V_2(x)$ of query $Q$ over a database schema $R$, where $V_2$ is a view, and $Q_2$ has a bounded query plan under an access schema $A$ and does not use any view. Then $Q$ has a bounded rewriting under $A$ no matter whether $[V_2(D)]$ is bounded or not for instances $D$ of $R$. Indeed, all fetching operations are conducted by $Q_2$; for each $x$-value a computed by $Q_2(x)$, we only need to validate whether $a \in V(D)$. This involves only cached $V_2(D)$, without accessing $D$, and hence, $[V_2(D)]$ does not need to be bounded.

The analysis of bounded output is also nontrivial.

**Theorem 3:** Problem $\text{BOP}(L)$ is

1. coNP-complete when $L$ is CQ, UCQ or $\exists \forall^+$; and
2. undecidable when $L$ is FO.

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

**Proof sketch:** (1) We show that $\text{BOP}$ is in coNP for $\exists \forall^+$ and is coNP-hard for CQ. The upper bound proof is a little involved, and demands a characterization of $\exists \forall^+$ queries with bounded output, in terms of notions of element queries and covered variables. Informally, (a) an element query $Q_i$, of a query $Q$ is a CQ obtained from $Q$ and an access schema $A$ such that its tableau representation satisfies $A$. Intuitively, we do not have to worry about $A$ when dealing with $Q_i$. We show that under $A$, each query $Q$ in $\exists \forall^+$ is $A$-equivalent to $Q_{e_1} \cup \cdots \cup Q_{e_n}$, where each $Q_{e_i}$ is an element query. (b) A variable of $Q$ is covered by $A$ if its use is controlled by $A$ and hence, the number of its possible valuations is bounded by the cardinality constraints of $A$. Based on these, we give the characterization as follows:

**Lemma 4:** For an $\exists \forall^*$ query $Q(x)$ and an access schema $A$, $Q(x)$ has bounded output iff for every element query $Q_i(x)’$ of $Q(x)$, all variables in $x’$ are in $\text{cov}(Q_i, A)$, which is the set of variables in $Q_i$ covered by $A$.

Capitalizing on the characterization, we can readily develop a coNP algorithm to check whether an $\exists \forall^*$ query has bounded output under an access schema $A$.

The lower bound is verified by reduction from the complement of the 3SAT problem to BOP(CQ). Given a propositional formula $\psi$, 3SAT is to decide whether $\psi$ is satisfiable. It is NP-complete (cf. [19]). The reduction uses fixed database schema $R$ and access schema $A$, i.e., they do not depend on 3SAT instance $\psi$. As a result, $\text{BOP}(CQ)$ remains coNP-hard even under fixed $R$ and $A$.

(2) We show that $\text{BOP}(FO)$ is undecidable by reduction from the complement of the satisfiability problem for $FO$. The latter is to decide, given an FO query $Q$ defined over a database schema $R$, whether there exists an instance $D$ of $R$ such that $Q(D) \neq \emptyset$. It is undecidable even when $R$ is fixed (cf. [1]). The reduction uses $A = \emptyset$ and a fixed relational schema $R’$. Given $Q$, we define an FO query $Q’$ such that $Q’$ has bounded output over all instances of $R’$ iff $Q$ is not satisfiable.

Using Lemma 2 and Theorem 3, we prove Theorem 1.

**Proof of Theorem 1.** The proof has three parts.

(1) We show that $\text{VBRP}$ is in $\Sigma_2^P$ for $\exists \forall^+$, by developing an algorithm for checking whether an $\exists \forall^*$ query $Q$ has a bounded rewriting. The algorithm works as follows: (a) guess a query plan $\xi$ in $\exists \forall^*$ of size at most $M$; (b) check whether $\xi$ conforms to $A$; if so, continue; otherwise reject the current guess; (c) check whether $Q_2 \equiv_A Q$ for the $\exists \forall^*$ query $Q_2$ expressed by $\xi$; if so, return “yes”. The algorithm is in $\Sigma_2^P$ since step (b) is in $\Delta_2^p$ (Pspace) as it has to check whether the polynomially many fetch operations in $\xi$ have bounded output, and checking each fetch is in coNP by Theorem 3(1); moreover, step (c) is in $\Pi_2^p$ by Lemma 2.

(2) The lower bound proof is more involved. We show that $\text{VBRP}$ is $\Sigma_2^p$-hard for CQ, by reduction from the $\exists \forall^* \exists \forall^3$ CNF problem, which is $\Sigma_2^p$-complete [34]. The latter is to decide, given a sentence $\varphi = \exists \forall^3 \forall^3 \exists \forall^3 \forall^3 \psi$, whether $\varphi$ is true, where $\psi$ is an instance of 3SAT defined over variables in $X \cup \bar{X} \cup Z$.

Given such a $\varphi$, the reduction employs fixed database schema $R$ and access schema $A$ that do not vary for different instances $\varphi$, and a fixed constant $M = 6$. We define a CQ $Q$ and a set $V$ consisting of a single view $V$ in CQ. The encoding of $V$ is quite intricate; it assures that whenever $V$ is used in a bounded rewriting of $Q$, it must occur in the corresponding query plan $\xi$ in the form of $\sigma_{X - \forall^3 X} (V)$, where $\forall^3 X$ denotes a truth-assignment of $X$. When $V$ is used in any other form in a query plan $\xi’$, it is shown that either $\xi’$ does not conform to $A$, or it does not help us in answering $Q$, and thus can be dispelled. Moreover, the access constraints in $A$ and query $Q$ are such defined that from $D \models A$ and $Q(D) \neq \emptyset$, we can derive that $D$ encodes a truth assignment $\mu_Y$ of $Y$. Hence, $Q$ has a bounded rewriting using $V$ iff $\xi’$ "corresponds" to a truth assignment $\mu_X$ for $X$, and when ranging over all instances $D$ of $R$ for which $D \models A$ and $Q(D) \neq \emptyset$ (to simulate all truth assignments $\mu_Y$ for $Y$), we
have that $Q(D) = \xi(D)$. By ensuring that $\xi(D) = \emptyset$ when $\exists Z \psi(\mu_X, \mu_Y, Z)$ is false, we show that $Q(D) = \xi(D)$ whenever $\exists Z \psi(\mu_X, \mu_Y, Z)$ is true. Since $\mu_X$ is fixed in $\xi$, and all $\mu_Y$ are considered (since $Q(D) = \xi(D)$ must hold for all $D \models A$), this implies that $Q \equiv_A \xi$ iff $\forall Y \forall Z \psi(\mu_X, Y, Z)$ is true, where $\mu_X$ is the chosen truth-assignment for $V$.

(3) We show that VBRP(FO) is undecidable also by reduction from the complement of the satisfiability problem for FO. Given an FO query $Q$, we define an FO query $Q'$ such that $Q'$ has an $M$-bounded rewriting iff $Q$ is not satisfiable. We use fixed $A, V$ and $M$ in the reduction. □

### 3.2 The Impact of Various Parameters

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

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

**Corollary 5:** There exist fixed $\mathcal{R}$, $A$, $M$ and $V$ such that it is undecidable to decide, given an FO query $Q$, whether $Q$ has an $M$-bounded rewriting in FO using $V$ under $A$. □

**Proof sketch:** It is undecidable to decide, given an FO query $Q$ over a fixed database schema $\mathcal{R}$, whether there exists an instance $D$ of $\mathcal{R}$ such that $Q(D) \neq \emptyset$. Indeed, a proof is by reduction from the the Post Correspondence Problem using a fixed $\mathcal{R}$ (see e.g., [1]). Hence the reduction given in the proof of Theorem 3 for FO suffices to show Corollary 5, which employs $A = V = \emptyset$ and $M = 1$. □

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

**Fixing $\mathcal{R}$, $A$ and $M$.** Fixing database schema, access schema and plan size does not help us. Indeed, the $\Sigma_2^n$ lower bound for CQ is verified by using fixed $\mathcal{R}$, $A$ and $M$ (Theorem 1). From this the corollary below follows.

**Corollary 6:** There exist fixed $\mathcal{R}$, $A$ and $M$ such that it is $\Sigma_2^n$-complete to decide, given a query $Q$ in $\mathcal{L}$ and a set $V$ of $\mathcal{L}$-definable views over $\mathcal{R}$, whether $Q$ has an $M$-bounded rewriting in $\mathcal{L}$ using $V$ under $A$ $\mathcal{L}$ is CQ, UCQ or $\exists \forall \mathcal{O}^+$. □

**Fixing $\mathcal{R}$, $A$, $M$ and $V$.** Suppose that besides $\mathcal{R}$, $A$ and $M$, the set $V$ of views is also predefined. This puts VBRP in $\Sigma_{2k+1}^p$ for CQ, UCQ and $\exists \forall \mathcal{O}^+$, where $\Sigma_{2k+1}^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$ where $\Sigma_2^p$ is the complexity class defined as $\exists \forall \mathcal{O}^+$.

### Theorem 7: For each natural number $k$, there exist fixed $\mathcal{R}$, $A$, $M$, $V$ such that it is $\Sigma_{2k+1}^p$-complete to decide, given a query $Q$ in $\mathcal{L}$ over $\mathcal{R}$, whether $Q$ has an $M$-bounded rewriting in $\mathcal{L}$ using $V$ under $A$, for $\mathcal{L}$ as CQ, UCQ or $\exists \forall \mathcal{O}^+$. □

To verify Theorem 7, we need the following notations, which will also be used in Section 4.

(a) A query $Q_1$ is $A$-contained in query $Q_2$, denoted by $Q_1 \sqsubseteq_A Q_2$, if for all instances $D$ of $\mathcal{R}$ that satisfy $A$, $Q_1(D) \subseteq Q_2(D)$, for $Q_1$ and $Q_2$ defined over $\mathcal{R}$.

(b) For a query $Q$, denote by $Q_{\Sigma}$ the set of all candidate query plans using $V$ that are no larger than $M$.

(c) For $\xi \in Q_{\Sigma}$, we write $\xi \sqsubseteq_A Q$ if $Q_\xi \sqsubseteq_A Q$, where $Q_\xi$ denotes the query expressed by $\xi$ (recall $Q_\xi$ from Section 2); similarly we write $Q \sqsubseteq_A \xi$ if $Q \sqsubseteq_A Q_\xi$, and $\xi \sqsubseteq_A \xi'$ for $\xi' \in Q_{\Sigma}$ if $Q_\xi \sqsubseteq_A Q_{\xi'}$. We write $\xi \equiv_A \xi'$ if $\xi \sqsubseteq_A \xi'$ and $\xi' \sqsubseteq_A \xi$, and $\xi \sqsubseteq_A \xi'$ but $\xi \not\equiv_A \xi'$.

**Proof sketch:** (1) Upper bound. A query $Q$ in $\exists \forall \mathcal{O}^+$ has an $M$-bounded rewriting using $V$ under $A$ if and only (either (a) $Q$ is not satisfiable by any instance $D \models A$ of $\mathcal{R}$, i.e., $Q \equiv_A Q_\emptyset$, where $Q_\emptyset$ denotes a query that returns $\emptyset$ on all $D$, or (b) $Q$ is satisfiable and $Q \equiv_A \xi$ for a non-empty $\xi \in Q_{\Sigma}$). We show that condition (a) can be checked in coNP. For each non-empty $\xi \in Q_{\Sigma}$, condition (b) is to check whether $\xi \sqsubseteq_A Q$ and $Q \sqsubseteq_A \xi$, which can be decided in NP and coNP, respectively, for fixed $\mathcal{R}$, $A$, $M$ and $V$. Here $k$ denotes the number of non-empty plans in $Q_{\Sigma}$, a constant here.

In contrast, if $M$ is fixed while $\mathcal{R}$, $A$ or $V$ is not, there are possibly exponentially many query plans in $Q_{\Sigma}$. As an example, when $V$ is not fixed, consider a view $V(x)$ with $|x| = m$. Then, depending on how the variables in $V$ are instantiated in a query plan $\xi$ with constants, i.e., how $V$ is used in $\xi$, there are at least $2^m$ different plans in $Q_{\Sigma}$. That is why VBRP(QC) remains $\Sigma_2^p$-hard when only $M$, $\mathcal{R}$ and $A$ are fixed but $V$ is not (Corollary 6).

(2) Lower bound. For any $k$, take any language $L = \bar{L}_0 \cup \bigcup_{i=1}^k (L_{2i-1} \cap L_{2i})$ in $\Sigma_{2k+1}$. We build fixed $\mathcal{R}$, $A$, $V$ and $M = 1$, i.e., they depend on $k$ only, not on $L$. We show that deciding whether a string $\bar{\sigma} \in L$ can be reduced to checking whether a CQ has an 1-bounded rewriting using $V$ under $A$.

The reduction uses the following properties. (a) Since each $L_i$ is in NP, we have reductions $f_i$ from $L_i$ to $\text{3SAT}$ such that for each string $\bar{\sigma}$, $\bar{\sigma} \in L_i$ iff $f_i(\bar{\sigma})$ is a satisfiable 3SAT instance. Hence if $f_{i+1}(\bar{\sigma})$ is satisfiable then so is $f_i(\bar{\sigma})$, by $L_i \supseteq L_{i+1}$. We use this property to ensure that only $k+1$ possible query plans need to be considered. (b) Following [38], it can be verified that $\bar{\sigma} \in L$ iff $\{|i \mid f_i(\bar{\sigma}) \text{ is satisfiable, } i \in [0, 2k]\}$ is an even number. We show that determining the latter condition is equivalent to deciding whether a CQ $Q$ has an 1-bounded rewriting using $V$ under $A$. In the reduction, we use $Q$ to encode the $2k+1$ 3SAT instances $f_i(\bar{\sigma})$, and employ $k$ unary fixed views and the empty view corresponding to the $k+1$ even number 0, 2, 4, . . . , $2k$ above.

**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 $\mathcal{R}$, $M$, $A$ and $V$ do not have to be fixed.

We use the following notations. For candidate plan $\xi \in Q_{\Sigma}$, we say that $\xi$ is a maximum plan with $(A, V)$ if (a)
Lemma 8: A query $Q$ has an $M$-bounded rewriting under $A$ using $V$ iff there exists a unique maximum plan $\xi \in Q_{\mathcal{P}}$ such that $\xi \subseteq A Q$. We say that $\xi$ is unique in $Q_{\mathcal{P}}$ if there exists no another maximum plan $\xi' \in Q_{\mathcal{P}}$ such that $\xi \not= A \xi'$. □

Proof sketch: A query $Q$ has an $M$-bounded rewriting under $A$ using $V$ iff there exists a query plan $\xi \in Q_{\mathcal{P}}$ such that $Q \equiv A \xi$. We show that when such a plan $\xi$ exists, it is maximum and unique up to $A$-equivalence. □

4. BOUNDED REWRITING FOR ACQ

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

(1) Acyclic conjunctive queries, denoted by ACQ. A CQ $Q$ is acyclic if its hypergraph has hypertree-width 1 [21]. The hypergraph of $Q$ is a hypergraph $(V_{\xi}, E_{\xi})$ in which $V_{\xi}$ consists of variables in $Q$ and $E_{\xi}$ has an edge for each subset of variables that occur together in a relation atom in $Q$. Acyclic conjunctive queries are commonly used in practice. As an example, query $Q_0$ of Example 1 is an ACQ.

(2) Fixed $\mathcal{R}, A, M$ and $V$. We consider predefined database schema $\mathcal{R}$, access schema $A$, bound $M$ and views $V$. After all, for an application, $\mathcal{R}$ is designed first, $M$ is determined by our available resources (e.g., processors and time constraints), access constraints are discovered from sample instances of $\mathcal{R}$, and views are selected based on the application [6]. These are determined before we start answering queries. Thus it is practical to assume fixed $\mathcal{R}, A, M$ and $V$.

In this setting, we study bounded rewriting of ACQ. Given an ACQ $Q$, we want to find an $M$-bounded query plan $\xi(Q, V, \mathcal{R})$ under $A$ in CQ (Section 2) such that the query $Q\xi$ expressed by $\xi$ is an ACQ. Our main conclusion is that the intractability of VBRP is robust, even for ACQ under fixed $\mathcal{R}, A, M$ and $V$. However, 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 $\mathcal{R}, A, M$ and $V$. Unfortunately, VBRP remains intractable in this setting, even under quite restrictive access constraints in a fixed $A$.

Theorem 9: Given fixed $\mathcal{R}, A, M$ and $V$, VBRP(ACQ) is coNP-hard when $A$ has one of the following forms:

1. when $A$ consists of a single access constraint of the form $R(A \rightarrow B, N)$ and $N \geq 2$;
2. when $A$ consists of two constraints $R(A \rightarrow B, 1)$ and $R'(\emptyset \rightarrow (E, F), N)$, and $N \geq 6$; or
3. when $A$ consists of two constraints $R((A, B) \rightarrow C, 1)$ and $R'(\emptyset \rightarrow E, N)$, and $N \geq 2$.

Proof sketch: We prove the statements by reductions from coNP-complete problems $L$. In the reductions we construct database schema $\mathcal{R}$, access schema $A$ and query $Q$ with care such that they do not depend on the instances $\phi$ of $L$, and $\phi$ is a “yes” instance of $K$ iff $Q \equiv_A \emptyset$, a query over $\mathcal{R}$ that returns empty for any instances $D \models A$ of $R$. In light of these, the proofs remain intact when $M$ is any predefined number and $V$ is any predefined set of ACQ queries.

The proofs tell us that there is intricate interaction between ACQ and access constraints, and the forms of access constraints in $A$ have impact on the complexity of VBRP.

Case 1) We verify the coNP lower bound by reduction from the complement of the precoloring extension problem, which is NP-complete [26]. The latter is to decide, given an undirected graph $G = (V_G, E)$ and a 3-coloring of leaves of $G$, whether $\mu$ is extendable to a color $c_j (\epsilon_j \in \{r, g, b\})$ for all nodes $v_i$ in $V_G$ such that for each edge $(v_i, v_j) \in E$, $c_i \not= c_j$. The reduction is a little complicated since we need to encode a cyclic graph with ACQ. It uses an access schema $A$ consisting of a single $\phi = R(A \rightarrow B, 2)$, and a database schema $\mathcal{R}$ with a single binary relation $R(A, B)$. To ensure that query $Q$ is acyclic, we represent the two vertices of each edge with distinct variables, and use access constraint $\phi$ to enforce “equality” on the variables and thus, “restore” the original graph $G$. That is, the reduction makes use of the interaction between the access constraint and ACQ.

Case 2) We show this case by reduction from the complement of the 3-Colorability problem, which is NP-complete (cf. [19]). The latter problem is to decide, given an undirected graph $G = (V_G, E)$, whether there exists a color $c_j (\epsilon_j \in \{r, g, b\})$ for each vertex $v_i$ in $V_G$ such that for every edge $(v_i, v_j) \in E$, $c_i \not= c_j$. Again the challenge is to encode cyclic $G$ with ACQ. The reduction uses a set $A$ with two access constraints $R(A \rightarrow B, 1)$ and $R'(\emptyset \rightarrow (E, F), 6)$, and a database schema $\mathcal{R}$ of two relations $R(A, B)$ and $R'(E, F)$.

Case 3) We show this by reduction from the complement of the 3SAT problem. We use an access schema $A$ having $R((A, B) \rightarrow C, 1)$ and $R'(\emptyset \rightarrow E, 2)$, and database schema $\mathcal{R}$ of a ternary relation $R(A, B, C)$ and a unary relation $R'(E)$.

The complication is introduced by encoding Boolean operations disjunction, conjunction and negated variables given limited facility of ACQ, and restricted relations of $\mathcal{R}$ and constraints of $A$. Hence the reduction is quite delicate. Given an instance $\psi(X)$ of 3SAT, we define $Q$ to encode each clause in $\psi$ with distinct copies of variables of $X$, and encode the Boolean operations in terms of distinct constants to represent their truth values. Together with the constraints, these ensure that $Q$ correctly encodes $\psi$ and is an ACQ.

We show that the proofs for cases (1), (2), (3) can be extended to $N > 2, N > 6, N > 2$, respectively. □

Characterization. In light of Theorem 9, we next characterize when VBRP(ACQ) is tractable for sub-classes $C$ of ACQ, and give an upper bound for VBRP(ACQ).

Theorem 10: When $\mathcal{R}, A, M$ and $V$ are fixed, (1) for any sub-class $C$ of ACQ, VBRP(C) is in PTIME if and only if for each query $Q \in C$, it is in PTIME to check whether $Q \equiv_A \xi$, where $\xi$ 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, when taken together, simplify the analysis of VBRP (unless P = NP), to an extent, as opposed to the $\Sigma_2^P$-completeness of Theorem 1 and $\Sigma_{2k+1}^P$-completeness of Theorem 7. Putting Theorems 9 and 10 together, we can see that the cases of VBRP(ACQ) stated in Theorem 9 are coNP-complete.

The proof of Theorem 10 is based on Lemma 8 and the lemma below, which gives the complexity of basic operations that are needed for computing maximum query plans.
Lemma 11: When \( R, A, M \) and \( V \) are fixed, given a CQ \( Q \) and query plans \( \xi, \xi' \in Q_{PQ} \), it is in
(a) \( \text{PTIME} \) to check whether \( \xi \) conforms to \( A \),
(b) \( \text{PTIME} \) to check whether \( \xi \sqsubseteq_A Q \) if \( Q \) is an ACQ,
(c) \( \text{NP} \) to check whether \( Q \sqsubseteq_A \xi \), and
(d) \( \text{PTIME} \) to check whether \( \xi' \sqsubseteq_A \xi \) for \( \xi' \in Q_{PQ} \). □

Based on the lemmas, we prove Theorem 10.

Proof sketch of Theorem 10. We develop an algorithm that, given an ACQ \( Q \), checks whether \( Q \) has a bounded rewriting. The algorithm first computes the unique maximum plan \( \xi \in Q_{PQ} \) (up to \( A \)-equivalence) if it exists. It then checks whether \( Q \sqsubseteq_A \xi \). Its correctness is warranted by Lemma 8. For its complexity, we show that it is in \( \text{PTIME} \) to check whether \( Q \sqsubseteq_A \xi \). We give a proof sketch.

Theorem 10 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 \( A \) are of the form \( R(X \rightarrow Y, 1) \). As remarked earlier, FDs with associated indices are common access constraints, and can be automatically discovered by using existing tools for mining FDs (e.g., [24]).

Corollary 12: When \( R, A, M \) and \( V \) are fixed, VBRP is in \( \text{PTIME} \) for ACQ if \( A \) consists of FDs only.

Proof sketch: By Theorem 10, to prove Corollary 12, it suffices to show that \( Q \sqsubseteq_A \xi \) is in \( \text{PTIME} \). Observe the following. (a) When \( A \) consists of FDs only, we can “chase” the tableau of \( Q \) by access constraints of \( A \) in \( \text{PTIME} \) [1], and get a unique query \( Q_A \) such that the tableau of \( Q_A \) satisfies \( A \) and \( Q \equiv_A Q_A \). (b) Checking \( Q \sqsubseteq_A \xi \) is equivalent to checking \( Q_A \sqsubseteq \xi \). Hence we can use the conventional notion of query containment and the Homomorphism Theorem for CQ [1]. (c) The proof of Lemma 11 shows that \( Q_A \subseteq Q \) can be checked in \( \text{PTIME} \) for fixed \( R, A, M \) and \( V \). Putting these together, we have that \( Q \sqsubseteq_A \xi \) is in \( \text{PTIME} \). □

In contrast to Corollary 12, VBRP remains intractable for CQ under FDs, although the analysis is simpler compared with Theorem 7 (unless \( P = \text{NP} \)).

Proposition 13: For fixed \( R, A, M \) and \( V \), VBRP(CQ) is \( \text{NP} \)-complete if \( A \) consists of FDs only. It remains \( \text{NP} \)-complete when none of \( R, A, M \) and \( V \) is fixed.

Proof sketch: We show that VBRP is \( \text{NP} \)-hard by reduction from 3SAT, using fixed \( R, A, M \) and \( V \), with consisting of FDs. In contrast to Corollary 12, for a CQ \( Q \), we can no longer compute its maximum query plan in \( Q_{PQ} \) in \( \text{PTIME} \); instead, it is an \( \text{NP} \) process.

For the upper bound, we give an algorithm to check \( \text{VBRP}(CQ) \) under FDs, which is in \( \text{NP} \) even when none of \( R, A, M \) and \( V \) is fixed. It uses the following lemma.

Lemma 14: If \( A \) consists of FDs only, it is in \( \text{PTIME} \) to decide whether a plan \( \xi \in Q_{PQ} \) conforms to \( A \).

We give a \( \text{PTIME} \) algorithm to verify the lemma. □

Along the same lines as Corollary 12, one can easily verify that for fixed \( R, A, M \) and \( V \), VBRP is in \( \text{PTIME} \) for the sub-class of ACQ queries such that their tableau representations satisfy the cardinality constraints in \( A \). A special case of this is when \( A = \emptyset \), e.g., the setting of [6], when access constraints are not employed at all.

Theorem 10 remains intact on any class \( C \) of queries as long as it is in \( \text{PTIME} \) to compute a maximum plan in \( Q_{PQ} \) for all queries in \( C \). Examples include (a) self-join-free CQ, i.e., the class of CQ queries that have at most \( 2 \) free variables [37]. Using Theorem 10 and Lemma 11, one can show that VBRP is also in \( \text{PTIME} \) in these two cases.

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. For any database schema \( R \), views \( V \), access schema \( A \) and bound \( M \), we identify two classes of FO queries, (a) a class of queries \( \text{topped by } (R, V, A, M) \), which “covers” all queries defined over \( R \) that have an \( M \)-bounded rewriting using \( V \) under \( A \), up to equivalence, and (b) a class of \( \text{size-bounded} \) queries, which “covers” all the views of \( V \) that have bounded output for all instances of \( R \) that satisfy \( A \). The second class helps us effectively check bounded output when deducing topped queries. It is in \( \text{PTIME} \) to decide 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 15: For any \( R, V \) and \( M \), and under any \( A \),
(a) each FO query \( Q \) with an \( M \)-bounded rewriting using \( V \) is \( A \)-equivalent to a query topped by \( (R, V, A, M) \);
(b) every FO query topped by \( (R, V, A, M) \) has an \( M \)-bounded rewriting in FO using \( V \) under \( A \); and
(c) it takes \( \text{PTIME} \) in \( M, |R|, |Q|, |V| \) and \( |A| \) to check whether an FO query \( Q \) is topped by \( (R, V, A, M) \), which uses an effective syntax that checks whether FO views in \( V \) have bounded output in \( \text{PTIME} \) in \( |Q| \).

Here \( A, Q \) and \( V \) are all defined over the same \( R \). □

That is, topped queries are a core sub-class of FO queries with a bounded rewriting, and can be effectively checked. To check bounded output of queries, we show the following.

Theorem 16: For any \( R \) and under any \( A \),
(a) each FO query \( Q \) over \( R \) that has bounded output is \( A \)-equivalent to a size-bounded query under \( A \);
(b) each size-bounded query has bounded output under \( A \); and
c) it takes \( \text{PTIME} \) in \( |Q| \) to check whether an FO query \( Q \) is a size-bounded query.

Here \( A \) and \( Q \) are defined over the same \( R \). □

Before we define topped and size-bounded queries, we remark the following. (1) Theorems 15 and 16 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 15 does not contradict to Corollary 5 due to the requirement of $A$-equivalence in its condition (a), which is undecidable for FO.

Practical use. Capitalizing on the effective syntax, we can develop algorithms (a) to check whether a given FO query $Q$ is topped by $(\mathcal{R}, \mathcal{V}, A, M)$ in PTIME, and if so, (b) to generate a bounded query plan $\xi$ for $Q$ using $\mathcal{V}$. The existence of these algorithms are warranted by Theorems 15 and 16. We can then support bounded rewriting using views 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 [6], and a set $A$ of access constraints can be discovered [10]. After these are in place, given an FO query $Q$ posed on a dataset $D$ that satisfies $A$, we check whether $Q$ is topped by $(\mathcal{R}, \mathcal{V}, A, M)$. If so, we generate a bounded query plan $\xi$ for $Q$ using $\mathcal{V}$, by using the algorithms described above. Then we can compute $Q(D)$ by executing $\xi$ with the existing DBMS. Moreover, incremental methods for maintaining the views $[6]$ and (the indices of) access constraints [10] have already been developed, in response to updates to $D$. Putting these together, we can expect to efficiently answer a number of FO queries $Q$ in (possibly big) $D$ by leveraging bounded rewriting.

5.2 Topped Queries for Bounded Rewriting

We next define topped queries and prove Theorem 15. It is nontrivial to define an effective syntax, as shown below.

Example 6: Consider a database schema $\mathcal{R}_1$ with two relations $R(A \rightarrow B, N)$ and $T(C, E)$, an access schema $\mathcal{A}_2$ consisting of $R(A \rightarrow B, N)$ and $T(C \rightarrow E, N)$, and $\mathcal{V}_3$ with a single view $V_3(x, y) = \exists y R(y, y) \land T(x, y)$. Consider FO query:

\[
Q_3(z) = Q_4(z) \land \neg\exists w R(z, w),
\]

where

\[
Q_4(z) = \exists y \exists v (V_3(x, y) \land (x = 1)) \land R(y, z).
\]

Then $Q_3$ has a 13-bounded rewriting as shown in Fig. 2. The plan is actually developed for an equivalent query:

\[
Q'_3(z) = Q'_4(z) \land (Q'_4(z) \land \exists w R(z, w)).
\]

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., $R(y, z)$ and $R(z, w)$. (3) A plan for a sub-query of $Q_4$ may have to embed a plan for another sub-query. For instance, (i) $Q_4$ has a 5-bounded rewriting in $Q_3$; (ii) $\exists w R(w, z)$ has a 7-bounded rewriting, 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 $\exists w R(w, z)$. □

Example 6 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 motivates us to define two functions.

- Boolean function $\text{cov}(Q_3(x), Q(z))$ to ensure that if $Q_3(x)$ has a bounded rewriting and $\text{cov}(Q_3(x), Q(z)) = \text{true}$, then $Q_3(x) \land Q(z)$ also has a bounded rewriting.

- Function $\text{size}(Q_3(x), Q(z))$ to compute a natural number, which is an upper bound of the size of minimum sub-plans for sub-query $Q(z)$ in $Q_3(x) \land Q(z)$.

The intuition behind these functions is as follows.

\[
S_{12} = S_5 - S_{12}
\]

\[
S_{5} = \pi_B(S_1)
\]

\[
S_{12} = \pi_B(S_{11})
\]

\[
S_4 = \text{fetch}(A \in S_3, R, B)
\]

\[
S_{11} = \text{fetch}(A \in S_3, R, B)
\]

\[
S_1 = \pi_B(S_2)
\]

\[
S_{10} = \pi_B(S_4)
\]

\[
S_2 = \sigma_{<1}(S_3)
\]

\[
S_3 = \text{fetch}(A \in S_3, R, B)
\]

\[
S_1 = V_3
\]

\[
S_9 = \pi_B(S_2)
\]

\[
S_7 = \sigma_{<1}(S_1)
\]

\[
S_6 = V_3
\]

Figure 2: A bounded plan for $Q_3$ of Example 6

(a) Intuitively, $Q(z)$ indicates a (sub-)query we are inspecting, and $Q_3(x)$ keeps track of sub-queries from which values are propagated to $Q(z)$, to make $Q(z)$ bounded.

(b) Function $\text{cov}(Q_3, Q)$ is to check whether we can propagate values from $Q_3$ to $Q$, and get a bounded rewriting of $Q$. In particular, $Q_3$ may include views from $\mathcal{V}$. When computing $\text{cov}(Q_3, Q)$, we distinguish views that need to have bounded output from those that do not have to.

For instance, $\text{cov}(Q_4, \exists w R(w, z)) = \text{true}$ for $Q_3$ in Example 6, in which $Q_4$ is $Q_3$, and $\exists w R(w, z)$ is $Q$. It indicates that by propagating values from bounded $Q_3$, we can have a bounded rewriting for sub-query $\exists w R(w, z)$.

(c) Function $\text{size}(Q_3, Q)$ helps us ensure that our query plans do not exceed a given bound $M$. In Example 6, $\text{size}(Q_4, \exists w R(w, z)) = 7$, which is the size of the plan for evaluating $\exists w R(w, z)$ in $Q_3$ by using values propagated from $Q_4$.

Using the functions, we now define topped queries. An FO query $Q$ over $\mathcal{R}$ is in topped by $(\mathcal{R}, \mathcal{V}, A, M)$ if

\[
(1) \quad \text{cov}(Q_3, Q) = \text{true}; \quad \text{and}
\]

\[
(2) \quad \text{size}(Q_3, Q) \leq M.
\]

Here $Q_3$ is a “tautology query” such that for any query $Q$, $Q_3 \land Q = Q$, $\text{cov}(Q_3, Q_3) = \text{true}$ and $Q_3$ has a 0-bounded rewriting. That is, we compute $\text{cov}(Q_3, Q)$ and $\text{size}(Q_3, Q)$ starting with $Q_3 = Q_3$, and conclude that $Q$ is topped by $(\mathcal{R}, \mathcal{V}, A, M)$ if the two conditions above are satisfied.

Functions $\text{cov}(Q_3(x), Q(z))$ and $\text{size}(Q_3(x), Q(z))$. We next define the two 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 $\text{cov}(Q_3(x), Q(z)) = \text{true}$ and $Q_3(x)$ has a size $(Q_3, Q_3(x))$-bounded rewriting, then $Q_3(x) \land Q(z)$ has a size $(Q_3, Q_3(x) \land Q(z))$-bounded rewriting.

To simplify the discussion, we assume w.l.o.g. the following [1]: (a) no variable $x$ occurs both free and bound in $Q$, and there exists at most one quantifier for each $x$; and (b) there exists no universal quantifier in $Q$, i.e., we replace each $\forall x \psi(x)$ with $\neg\exists x (\neg\psi(x))$; (c) only variables occur in relation atoms, e.g., $R(x, 1)$ is replaced with $\exists y (R(x, y) \land y = 1)$; and (d) no relation atoms contain repeated variables, e.g., we substitute $R(x, y) \land x = y$ for each $R(x, x)$.

The definition is separated into 7 cases below.

\[
(1) \quad Q(z) \text{ is } z = c. \quad \text{For equality atom with a constant,} \quad \text{cov}(Q_3(x), Q(z)) = \text{true} \quad \text{and} \quad \text{size}(Q_3(x), Q(z)) = 1.
\]

\[
(2) \quad Q(z) \text{ is } V_3. \quad \text{We can access cached views; thus,} \quad \text{cov}(Q_3(x), Q(z)) = \text{true} \quad \text{and} \quad \text{size}(Q_3(x), Q(z)) = 1.
\]

That is, constant queries and views have 1-bounded rewriting and hence, are topped queries.
(3) $Q(\bar{z})$ is $Q'(\bar{z}) \land C$, where $C$ is $(x = y), (x \neq y), (x = c)$
or $(x \neq c)$. We define $\text{cov}(Q(\bar{z}), Q(\bar{z}')) = \text{cov}(Q(\bar{z}), Q'(\bar{z}'))$,
and define $\text{size}(Q(\bar{z}), Q(\bar{z}))$ as $\text{size}(Q(\bar{z}), Q'(\bar{z})) + 1$ when
$\text{cov}(Q(\bar{z}), Q'(\bar{z}))$ is true, and as $\infty$ otherwise.

Note that given a bounded plan $\varepsilon'$ for $Q$, a bounded plan for $Q$ is $(T = \varepsilon', \sigma(\varepsilon'))$, increasing the size of $\varepsilon'$ by 1.

(4) $Q(\bar{z})$ is $Q(\bar{z}) \lor Q(\bar{z})$, where $Q$ is not an (in)equality.

Let $\mu_s = \text{cov}(Q(\bar{z}), Q(\bar{z}))$, $s_1 = \text{size}(Q(\bar{z}), Q(\bar{z}))$, $\mu' = \text{cov}(Q(\bar{z}) \land Q(\bar{z}), Q(\bar{z}))$ and $s' = \text{size}(Q(\bar{z}) \land Q(\bar{z}), Q(\bar{z}))$.

Then, if $\mu_s = 1$ and $s_2(\bar{z}_2)$ is of the form $\exists y R(\bar{z}_1, \bar{z}_2, \bar{w})$,
$R(Z_1 \to Z_2, N)$ is in $A$ with $Z_1 \cup Z_2 = Z_2$ and if $Q(\bar{z}) \land Q(\bar{z})$ has bounded output under $A$, then $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$.

In this case we characterize value propagation shown in Example 6, e.g., $y$ from $\exists y \forall(x, y, z)$
and $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$, $\text{size}(Q(\bar{z}), Q(\bar{z})) = s_1 + 1$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = \text{size}(Q(\bar{z}), Q(\bar{z})) = 0$.

In this case we characterize value propagation shown in Example 6, e.g., $y$ from $\exists y \forall(x, y, z)$
and $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$, $\text{size}(Q(\bar{z}), Q(\bar{z})) = s_1 + 1$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = 0$ then $s_2 = 0$; otherwise
if $s_1 = \text{size}(Q(\bar{z}), Q(\bar{z})) = 0$.

Intuitively, if $Q_1$ and $Q_2$ have bounded plans $\varepsilon_1$ and $\varepsilon_2$, respectively, then $Q(\bar{z})$ has a bounded plan $(T_1 = \varepsilon_1, T_2 = \varepsilon_2, T_1 \cup T_2)$, of size $|\varepsilon_1| + |\varepsilon_2| + 1$.

(6) $Q(\bar{z})$ is $Q(\bar{z}) \land Q(\bar{z})$. Let $\mu_s = \text{cov}(Q(\bar{z}, Q(\bar{z}))$, $s_1 = \text{size}(Q(\bar{z}), Q(\bar{z}))$, $\mu_s = \text{cov}(Q(\bar{z}), Q(\bar{z}) \land Q(\bar{z}))$, and $s_1 = \text{size}(Q(\bar{z}), Q(\bar{z}))$.

(a) if $\mu_s = 1$ and $\varepsilon_1 = 1$ then $\text{cov}(Q(\bar{z}), Q(\bar{z}))$ is true and
$\text{size}(Q(\bar{z}), Q(\bar{z})) = s_1 + s_2 + 1$; otherwise
(b) if $\mu_s = 1$ and $\varepsilon_1 \leq K$ for some constant $K$, then $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$.

This is the case that captures how sub-query $Q_4$ of $Q_3$ is propagated to $\exists y R(w, z)$ in Example 6. When $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$, we have one of the following three cases.

(a) When $\mu_s = 1$ and $\varepsilon_1 = 1$, is similar to case (5) above.
(b) When $\mu_s = 0$ and $\varepsilon_1 = 0$, let $\varepsilon_2 = 1$ and $\varepsilon_2 = 0$, then $\text{size}(Q(\bar{z}), Q(\bar{z})) = s_1 + s_2 + 1$.
(c) Otherwise, $\text{cov}(Q(\bar{z}), Q(\bar{z})) = 0$, and $\text{size}(Q(\bar{z}), Q(\bar{z})) = s_1 + s_2 + 1$.

(7) $Q(\bar{z})$ is $\exists y Q(\bar{z}, \bar{w})$ ($\bar{w}$ is possibly empty). Let $\mu' = \text{cov}(Q(\bar{z}, Q(\bar{z}), Q(\bar{z}), \bar{w}))$.

Note that $Q(\bar{z})$ may not have bounded output even when it has a bounded rewriting. Therefore, in case (a) above we have to ensure that $Q(\bar{z})$ has bounded output in order to propagate $\exists y$-value from $Q(\bar{z})$ to $R(z)$, for a fetch operation to use the $\varepsilon$-value. Moreover, observe the following.

(a) When $Q(\bar{z})$ is a projection of a relation atom covered by a constraint $R(\bar{z} \to Z, N)$ in $A$, then $\text{size}(Q(\bar{z}), Q(\bar{z})) = 0$.
(b) If $Q(\bar{z})$ is covered by $R(\bar{z} \to Z, N)$ and $Q(\bar{z})$ has bounded output, then it has a plan $(T_1 = \varepsilon_1, T_2 = \varepsilon_2, T_3 = T_1 \times T_2)$, where $\varepsilon_1$ is the plan for $Q(\bar{z})$. Note that $Q(\bar{z})$ has a bounded plan by induction.

(b) Otherwise, $Q(\bar{z})$ has a bounded plan if $Q(\bar{z}, \bar{w})$ has one. Let $\varepsilon'$ be the plan for $Q(\bar{z})$. Then $(T_1 = \varepsilon', T_2 = \varepsilon')$ is a plan for $Q(\bar{z})$ with $Q(\bar{z})$.
Thus $\text{cov}(Q_r, Q_s) = \text{true}$. One can easily verify that $\text{size}(Q_r, Q_s) = 13$. Thus $Q_3$ is topped by $(\mathcal{R}_1, A_2, V_3, 13)$. 

Having defined topped queries, we now prove Theorem 15.

**Proof sketch of Theorem 15.** (a) Suppose that $Q$ is an FO query with an $M$-bounded rewriting, i.e., $Q$ has an $M$-bounded query plan $\xi(Q, V, \mathcal{R})$ under $A$. We show that there exists a query $Q_1$ topped by $(\mathcal{R}, V, \mathcal{A}, M)$ such that $\xi \equiv_{\mathcal{A}} Q_1$. This is verified by induction on $M$, verifying each case and step of $\xi$. In this direction, $Q_2 = Q_3$ suffices; in other words, we do not need to expand $Q_3$ as described in cases (4c) and (6b) above (to be elaborated shortly).

(b) We show that every query $Q$ topped by $(\mathcal{R}, V, \mathcal{A}, M)$ has a $\text{size}(Q_r, Q_s)$-bounded rewriting using $V$ under $A$. The proof needs the following lemma: if $\text{cov}(Q_r, Q_s) = \text{true}$, and if $Q_s$ has a $\text{size}(Q_r, Q_s)$-bounded plan, then $Q_s \wedge Q$ has a $\text{size}(Q_r, Q_s \wedge Q)$-bounded plan. This is verified by induction on the size of $Q$. For instance, when $Q(\bar{x}) = Q_1(\bar{z}_1) \wedge Q_2(\bar{z}_2)$, $\text{cov}(Q_s, Q(\bar{x})) = \text{true}$ and $Q_s$ has a $\text{size}(Q_1, Q_2)$-bounded plan, we know that $\text{cov}(Q_s, Q_1(\bar{z}_1))$ is also true. By the induction hypothesis we have that $Q_1 \wedge Q_1(\bar{z}_1)$ has a $\text{size}(Q_1, Q_1(\bar{z}_1))$-bounded plan. Thus either $\text{cov}(Q_s, Q_2(\bar{z}_2)) = \text{true}$ or $\text{cov}(Q_s, Q_1(\bar{z}_1), Q_2(\bar{z}_2)) = \text{true}$. In both cases, again by the induction hypothesis, $Q_s \wedge Q_1 \wedge Q_2$ has a $\text{size}(Q_s, Q_1 \wedge Q_2)$-bounded plan.

(c) It takes PTIME in $|\mathcal{R}|$, $|Q|$, $|V|$, $|A|$ and $M$ to check whether an FO query is topped by $(\mathcal{R}, V, \mathcal{A}, M)$. Indeed, it is easy to see that both $\text{cov}(Q_s, Q)$ and $\text{size}(Q_s, Q)$ are PTIME functions, which use a PTIME oracle to check bounded output for cases (4a) and (7b) of topped queries. 

**Remark.** Observe the following. (a) As mentioned above, to prove Theorem 15(1), it suffices to use $Q_s = Q_r$ and does not need to expand $Q_s$ (for cases (4c) and (6b)). We allow value propagation in cases (4c) and (6b) to cover queries that are commonly used and have a bounded rewriting. (b) The class of topped queries is quite different from the rules for $\bar{x}$-controllability [18] and the syntactic rules for bounded evaualibility of CQ [17] and for FO [10], particularly in the use of $Q_s$ to check bounded output of views and the function $\text{size}(Q_s, Q(\bar{x}))$ to ensure the bounded size of query plans.

5.3 Size Bounded Queries

We next define size-bounded queries and prove Theorem 16. 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 $A$ if it is of the following form:

$$Q(\bar{x}) = Q'(\bar{x}) \wedge \forall \bar{x}_1, \ldots, \bar{x}_{K+1}. (Q(\bar{x}_1) \wedge \ldots \wedge Q'(\bar{x}_{K+1})$$

$$\rightarrow \bigvee_{i, j \in [1, K+1], i \neq j}(\bar{x}_i = \bar{x}_j).$$

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

Intuitively, an FO query $Q'(\bar{x})$ has bounded output under $A$ if there exists a bound $K$ such that it is $A$-equivalent to $Q(\bar{x})$ of the form above, which explicitly states that $Q'(\bar{x})$ has at most $K$ distinct answers for a constant $K$.

This class of size-bounded queries suffices for Theorem 16.

**Proof of Theorem 16.** (a) Consider an FO query $Q(\bar{x})$ having bounded output under $A$. By the definition of queries with bounded-output (Section 3.1), there exists a natural

<table>
<thead>
<tr>
<th>Queries</th>
<th>Complexity</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>FO</td>
<td>undecidable (Th 1)</td>
<td></td>
</tr>
<tr>
<td>CQ, UCQ, $\exists FO^*$</td>
<td>$\Sigma_2^p$-complete (Th 1)</td>
<td></td>
</tr>
<tr>
<td>CQ, UCQ, $\exists FO^*$</td>
<td>$\Sigma_2^p$-complete (Cor 6)</td>
<td>fixed $\mathcal{R}, \mathcal{A}, \mathcal{M}$</td>
</tr>
<tr>
<td>CQ</td>
<td>NP-complete (Prop 13)</td>
<td>only FDs in $\mathcal{A}$</td>
</tr>
<tr>
<td>Fixed $\mathcal{R}, \mathcal{A}, \mathcal{M}$ and $V$ for the following</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FO</td>
<td>undecidable (Cor 5)</td>
<td></td>
</tr>
<tr>
<td>CQ, UCQ, $\exists FO^*$</td>
<td>$\Sigma_2^p$-complete (Th 7)</td>
<td></td>
</tr>
<tr>
<td>CQ</td>
<td>NP-complete (Prop 13)</td>
<td>only FDs in $\mathcal{A}$</td>
</tr>
<tr>
<td>ACQ</td>
<td>coNP (Th 10)</td>
<td></td>
</tr>
<tr>
<td>ACQ</td>
<td>coNP-complete (Th 9)</td>
<td>restricted $\mathcal{A}$</td>
</tr>
<tr>
<td>ACQ</td>
<td>PTIME (Cor 12)</td>
<td>only FDs in $\mathcal{A}$</td>
</tr>
</tbody>
</table>

Table 2: Complexity of VBRP($\mathcal{L}$)

6. CONCLUSION

We have formalized bounded query rewriting using views under access constraints, studied the bounded rewriting problem VBRP($\mathcal{L}$) when $\mathcal{L}$ is ACQ, CQ, UCQ, $\exists FO^*$ or FO, and established their upper and lower bounds, all matching, when $M, \mathcal{R}, \mathcal{A}$ and $V$ are fixed or not. The main complexity results are summarized in Table 2, 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.

There is much more to be done. One topic for future work is to study bounded rewriting $Q'$ in a query language more powerful than the one in which query $Q$ and views $V$ are defined, e.g., when $Q$ and $V$ are CQ while $Q'$ is a UCQ. We have only considered the setting when $Q, Q'$ and $V$ are in the same language. It is also interesting to study which language is powerful enough to express all bounded rewriting, along the same lines as [31]. Another topic concerns bounded rewriting and bounded evaualibility [10, 17] when we allow the amount of data accessed from the underlying dataset $D$ to be an $\alpha$-fraction of $D$, for a small $\alpha$ in the range of $[0, 1]$, rather than to be bounded by a constant. Similarly, we may allow $M$ to be a function of resources and workload, rather than a constant. The third topic is to study bounded view maintenance, to incrementally maintain $V(D)$ by accessing a bounded amount of data in $D$, in response to changes to $D$. The fourth topic is to develop methods for selecting views $V$ and discovering access constraints $A$, such that under $A$, the number of queries that have a bounded rewriting using $V$ is maximized in a given application.
7. REFERENCES


Acknowledgments. Fan, Cao and Lu are supported in part by ERC 652976, 973 Program 2014CB340302 and 2012CB316200, NSFC 61133002 and 61421003, EPSRC EP/J015377/1 and EP/G025268/1, NSF III 1302212, Shenzhen Peacock Program 110510003843461, Guangdong Innovative Research Team Program 2011D005, Shenzhen Science and Technology Fund JCYJ20150529164656096, Guangdong Applied R&D Program 2015B010131006, Beijing Advanced Innovation Centre for Big Data and Brain Computing, and a research grant from Huawei Technologies.