Bounded Query Rewriting Using Views

Yang Cao[1,3] Wenfei Fan[1,3] Floris Geerts[2] Ping Lu[3]
[1]University of Edinburgh [2]University of Antwerp [3]Beihang University
{Y.Cao-17@sms, wenfei@inf}.ed.ac.uk, floris.geerts@uantwerpen.be, luping@buaa.edu.cn

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 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 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 $\mathcal{A}$.

PODS'16, June 26-July 01, 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2902251.2902294

Bounded evaluation has proven useful [10–12]. Experimenting 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 evaluability, 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 $\mathcal{R}_0$ 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 $\mathcal{R}_0$, 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 (≤ 100) is obtained by aggregating $\mathcal{R}_0$ instances; an index is built on movie such that given any (studio, release) value, it returns

(at most N_0) corresponding mids; 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: an instance D_0 of $\mathcal{R}_0$ may have billions of person and like tuples [22], 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 follows:

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

As will be seen later, Q_0 can be rewritten into a conjunctive query Q_ξ using V_1, such that for all instances D_0 of $\mathcal{R}$ that satisfy $\mathcal{A}_0$, $Q_0(D_0)$ can be computed by Q_ξ that accesses only $V_1(D_0)$ and an additional $2N_0$ tuples from D_0, no matter how big D_0 is. Here $V_1(D_0)$ is a small set, much smaller than D_0 although its size is dependent on 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 [6]. 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 effective 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 D of $\mathcal{R}$ that satisfies $\mathcal{A}$, there exists a fraction D_Q of D such that

- $Q(D) = Q'(D_Q, \mathcal{V}(D))$, and
- the time for identifying D_Q and hence the size $|D_Q|$ of D_Q are independent of the size $|D|$ of D.

That is, we compute the exact answers $Q(D)$ via Q' by accessing cached $\mathcal{V}(D)$ and a *bounded* fraction D_Q of D. While $\mathcal{V}(D)$ may not be bounded, we can select small views following the methods of [6], 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 [33], which have a bounded size M determined by our available resources such as processors and time constraint.

(2) Complexity. We study *the bounded rewriting problem* (Section 3), referred to as $\mathsf{VBRP}(\mathcal{L})$ 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, $\mathsf{VBRP}(\mathcal{L})$ is to decide whether under $\mathcal{A}$, Q has a bounded rewriting in $\mathcal{L}$ using $\mathcal{V}$ with a query plan no larger than M, referred to as *an M-bounded query plan*. The need for studying $\mathsf{VBRP}(\mathcal{L})$ is evident: if Q has a bounded rewriting, then we can find efficient query plans to answer Q on possibly big datasets D.

We investigate $\mathsf{VBRP}(\mathcal{L})$ when $\mathcal{L}$ ranges over conjunctive queries (CQ, *i.e.,* SPC), unions of conjunctive queries (UCQ, *i.e.,*SPCU), positive FO queries ($\exists\mathsf{FO}^+$, 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 $\exists\mathsf{FO}^+$; 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 yet, we show that the intractability of VBRP is quite robust (Section 4). It remains intractable for acyclic CQ (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 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 $\mathcal{V}$; 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 *core* subclass of FO queries with a bounded rewriting using $\mathcal{V}$, without sacrificing their expressive power, along the same lines as rang-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 $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in 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 $\mathcal{V}$ have *bounded output* $\mathcal{V}(D)$ over all datasets D that satisfy $\mathcal{A}$, *i.e.,* the size $|\mathcal{V}(D)|$ is bounded by a constant. This is to ensure bounded accesses 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 FO (Section 3). In light of this, we develop 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 iff it is equivalent to a size-bounded FO, 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.

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 Θ 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 Θ tuples, *in the absence of* access constraints. 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 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(CQ) is Σ_3^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 CQ, UCQ and $\exists$FO$^+$, and is undecidable for FO [17]. A notion of effective boundedness was studied for CQ [12], based on a *restricted form* of query plans that conduct all data fetching before any relational operations start. It was shown [12] that effective boundedness is in PTIME for CQ. 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 $|\mathcal{V}(D)|$. (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 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 [17] that VBRP is Σ_3^p-complete for CQ, 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 CQ [17], and another one for FO [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 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 [10]. As remarked earlier, we have to reason about the size $|\mathcal{V}(D)|$ 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 CQ is necessarily equivalent to an effectively bounded CQ.

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 $\mathcal{V}(D)$. 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 [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 CQ and FO. Query rewriting using views under access patterns has been studied for CQ [32], and for UCQ and UCQ$^\neg$ (with negated relation 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 $R(X \to Y, N)$, $X \cup Y$ 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 evaluability. (c) Bounded rewriting allows access to the underlying data with controlled I/O, which is prohibited in [14,32]. 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 [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 $\mathbf{U}$ 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* $\mathcal{A}$ 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. An instance D of R *satisfies* φ if

- 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
- there exists a function (index) that given an X-value $\bar{a}$, returns $D_Y(X = \bar{a})$ 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 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. 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 $\mathcal{A}$, denoted by $D \models \mathcal{A}$, if D satisfies all the constraints in $\mathcal{A}$.

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

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

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

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

(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 in Q, 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) 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;

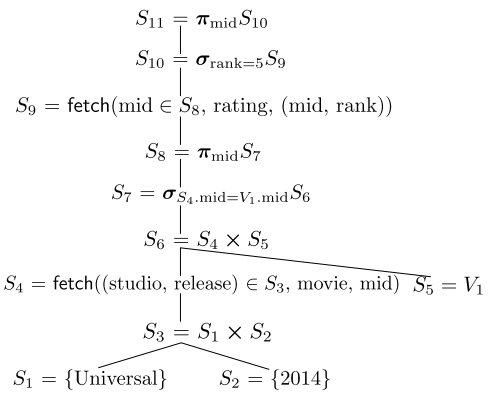

Figure 1: A query plan ξ_0 for Q_0 using view V_1

(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 D of $\mathcal{R}$, relations S_i's are computed by δ_i, bottom up in T_ξ as usual [33]. More specifically, δ_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_n associated with the root of T_ξ is the result of the computation.

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

The *size* of plan ξ is the number of nodes in T_ξ. 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 fetch$(X \in S_j, R, Y)$ when applying ξ to D. Intuitively, it measures the amount of I/O operations to access D.

Example 2: A plan $\xi_0(Q_0, V_1, \mathcal{R}_0)$ for Q_0 using view V_1 given in Example 1 is depicted in Fig. 1. Given an instance D of $\mathcal{R}_0$, (a) it fetches the set S_4 of mids of all movies released by Universal Studios in 2014, using constants in Q_0; (b) filters S_4 with mids in $V_1(D)$ 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 finally, (d) finds the set S_{11} of mids, which is precisely the answer $Q_0(D)$ to Q_0 in D. □

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 $\mathcal{A}$ defined over $\mathcal{R}$.

A query plan $\xi(Q, \mathcal{V}, \mathcal{R})$ is said to *conform to* $\mathcal{A}$ if

(a) for each 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_Q such that for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $|D_Q| \leq N_Q$.

That is, while ξ 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 $\mathcal{A}$. Plan ξ tells us how to retrieve D_Q from D such that $Q(D)$ is computed by using the data in D_Q and $\mathcal{V}(D)$ 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 $R(X \to Y, N)$).

Given a natural number M, we say that $\xi(Q, \mathcal{V}, \mathcal{R})$ is an *M-bounded plan for Q using $\mathcal{V}$ under $\mathcal{A}$* if (a) ξ conforms to $\mathcal{A}$, and (b) the size of ξ 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 $\xi(Q, \mathcal{V}, \mathcal{R})$ is M-bounded under $\mathcal{A}$, then for all datasets D that satisfy $\mathcal{A}$, we can efficiently answer Q in D by following ξ and accessing a bounded amount of data from D.

Example 3: 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 D of $\mathcal{R}_0$, ξ_0 accesses at most $2N_0$ tuples from D, where N_0 is the constant in φ_1 of $\mathcal{A}_0$, since $|S_4| \leq N_0$ by φ_1, and $|S_9| \leq N_0$ by $S_8 \subseteq S_4$ and constraint φ_2 on rating in $\mathcal{A}_0$; and (c) eleven operations are conducted in total. 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 evaluable under $\mathcal{A}_0$, 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 $\mathcal{L}$, a set $\mathcal{V}$ of $\mathcal{L}$-definable views, and an access schema $\mathcal{A}$, all over the same database schema $\mathcal{R}$. For a bound 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 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 moreover, for any node labeled $\cup$, all its ancestors in the tree T_ξ of ξ are also labeled with $\cup$; that is, $\cup$ is conducted 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 D of $\mathcal{R}$, $\xi(D) = Q_\xi(D)$ and moreover, 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.

Example 4: The CQ Q_0 of Example 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

$$Q_\xi(\mathsf{mid}) = \exists y_m \big(\mathsf{movie}(\mathsf{mid}, y_m, \text{"Universal"}, \text{"2014"}) \wedge V_1(\mathsf{mid}) \wedge \mathsf{rating}(\mathsf{mid}, 5)\big).$$

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

Notations used in this paper are summarized in Table 1.

symbols	notations
$\mathcal{R}$, R	database schema $\mathcal{R}$ and relation schema $R \in \mathcal{R}$
$\mathcal{A}$	access schema
$D \models \mathcal{A}$	an instance 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	plan ξ represented as a query tree
$\xi(D)$	the result of applying ξ to D
VBRP($\mathcal{L}$)	the bounded rewriting problem for queries in $\mathcal{L}$
$Q \equiv_\mathcal{A} Q'$	$\mathcal{A}$-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

Table 1: Notations

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 $\mathcal{V}$?

While VBRP($\mathcal{L}$) is important, it is nontrivial.

Theorem 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 investigating problems embedded in it, and outline a proof of Theorem 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 examples.

$\mathcal{A}$-equivalence. Consider a database schema $\mathcal{R}$. Under an access schema $\mathcal{A}$ over $\mathcal{R}$, we say that two queries Q_1 and Q_2 defined over $\mathcal{R}$ are *$\mathcal{A}$-equivalent*, denoted by $Q_1 \equiv_\mathcal{A} Q_2$, if for all instances D of $\mathcal{R}$ that satisfy $\mathcal{A}$, $Q_1(D) = Q_2(D)$. 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 $\mathcal{R}$, $Q_1(D) = Q_2(D)$, regardless of whether $D \models \mathcal{A}$. 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 (cf. [1]). In contrast, it has been shown that $\mathcal{A}$-equivalence is harder (unless P = NP).

Lemma 2 [17]: Given access schema $\mathcal{A}$ and two queries Q_1 and Q_2, it is Π_2^p-complete to decide whether $Q_1 \equiv_\mathcal{A} Q_2$, for Q_1 and Q_2 in CQ, UCQ or $\exists$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. It is easy to show that the problem is undecidable when it comes to FO.

Bounded output. Another complication is introduced by views in $\mathcal{V}$. 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 us to check whether a view $V \in \mathcal{V}$ has "bounded output".

Example 5: Consider database schema $\mathcal{R}_0$, query Q_0, and access schema $\mathcal{A}_0$ defined in Example 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 D of $\mathcal{R}_0$, $V_2(D)$ 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:

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

One can verify that Q_2 is a bounded rewriting of Q_0 using V_2 under $\mathcal{A}_1$ iff there exists a constant N_1 such that for all instances D of $\mathcal{R}$, if $D \models \mathcal{A}_1$, then $|V_2(D)| \leq N_1$; 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 leveraging constraint φ_1 of $\mathcal{A}_1$ on movie, and select pairs $(\mathsf{pid}, \mathsf{mid})$ from $V_2(D) \times S$ that are in a tuple $(\mathsf{pid}, \mathsf{mid}, \text{“movie”})$ in the like relation, by making use of 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 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 $\mathcal{A}_1$.

(b) In contrast, for rewriting some queries, we *do not* have to check whether a view has bounded output. Consider a rewriting $Q(x) = Q_3(x) \wedge V_3(x)$ 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 $|V_3(D)|$ is bounded or not for instances 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 $a \in V(D)$. This involves only cached $V_3(D)$, without accessing D, and hence, $|V_3(D)|$ 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 defined over $\mathcal{R}$.
- Question: Is there a constant N such that for all instances D of $\mathcal{R}$, if $D \models \mathcal{A}$ then $|V(D)| \leq N$?

The analysis of bounded output is also nontrivial.

Theorem 3: 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 sketch: (1) We show that BOP is in coNP for $\exists\mathsf{FO}^+$and is coNP-hard for CQ. The upper bound proof is a little involved, and demands a characterization of $\exists\mathsf{FO}^+$queries with bounded output, in terms of notions of element queries and covered variables. Informally, (a) an *element query* Q_e of a query Q is a CQ obtained from Q and an access schema $\mathcal{A}$ such that its tableau representation satisfies $\mathcal{A}$. Intuitively, we do not have to worry about $\mathcal{A}$ when dealing with Q_e. We show that under $\mathcal{A}$, each query Q in $\exists\mathsf{FO}^+$is $\mathcal{A}$-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 $\mathcal{A}$ if its use is controlled by $\mathcal{A}$ and hence, the number of its possible valuations is bounded by the cardinality constraints of $\mathcal{A}$. Based on these, we give the characterization as follows:

Lemma 4: For an $\exists\mathsf{FO}^+$query $Q(\bar{x})$ and an access schema $\mathcal{A}$, $Q(\bar{x})$ has bounded output iff for every element query $Q_e(\bar{x}')$ of $Q(\bar{x})$, all variables in $\bar{x}'$ are in $\mathsf{cov}(Q_e, \mathcal{A})$, which is the set of variables in Q_e covered by $\mathcal{A}$. □

Capitalizing on the characterization, we can readily develop a coNP algorithm to check whether an $\exists\mathsf{FO}^+$query has bounded output under an access schema $\mathcal{A}$.

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

(2) We show that $\mathsf{BOP}(\mathsf{FO})$ 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 $\mathcal{R}$, whether there exists an instance D of $\mathcal{R}$ such that $Q(D) \neq \emptyset$. It is undecidable even when $\mathcal{R}$ is fixed (cf. [1]). The reduction uses $\mathcal{A} = \emptyset$ and a fixed relational schema $\mathcal{R}'$. Given Q, we define an FO query Q' such that Q' has bounded output over all instances of $\mathcal{R}'$ 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 VBRP is in Σ_3^p for $\exists\mathsf{FO}^+$, by developing an algorithm for checking whether an $\exists\mathsf{FO}^+$query Q has a bounded rewriting. The algorithm works as follows: (a) guess a query plan ξ in $\exists\mathsf{FO}^+$of size at most M; (b) check whether ξ conforms to $\mathcal{A}$; if so, continue; otherwise reject the current guess; (c) check whether $Q_\xi \equiv_{\mathcal{A}} Q$ for the $\exists\mathsf{FO}^+$query Q_ξ expressed by ξ; if so, return “yes”. The algorithm is in Σ_3^p since step (b) is in Δ_2^p ($\mathsf{P}^{\mathsf{coNP}}$) as it has to check whether the polynomially many fetch operations in ξ have bounded output, and checking each fetch is in coNP by Theorem 3(1); moreover, step (c) is in Π_2^p by Lemma 2.

(2) The lower bound proof is more involved. We show that VBRP is Σ_3^p-hard for CQ, by reduction from the $\exists^*\forall^*\exists^*\mathsf{3CNF}$ problem, which is Σ_3^p-complete [34]. The latter is to decide, given a sentence $\varphi = \exists X \forall Y \exists Z\, \psi$, whether φ is true, where ψ is an instance of 3SAT defined over variables in $X \cup Y \cup Z$.

Given such a φ, the reduction employs fixed database schema $\mathcal{R}$ and access schema $\mathcal{A}$ that do not vary for different instances φ, and a fixed constant $M = 6$. We define a CQ Q and a set $\mathcal{V}$ 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 ξ in the form of $\sigma_{X=\mu_X}(V)$, where μ_X denotes a truth-assignment of X. When V is used in any other form in a query plan ξ', it is shown that either ξ' does not conform to $\mathcal{A}$, or it does not help us in answering Q, and thus can be dispelled. Moreover, the access constraints in $\mathcal{A}$ and query Q are such defined that from $D \models \mathcal{A}$ and $Q(D) \neq \emptyset$, we can derive that D encodes a truth assignment μ_Y of Y. Hence, Q has a bounded rewriting using V iff ξ “corresponds” to a truth assignment μ_X for X, and when ranging over all instances D of $\mathcal{R}$ for which $D \models \mathcal{A}$ and $Q(D) \neq \emptyset$ (to simulate all truth assignments μ_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 μ_X is fixed in ξ, and all μ_Y are considered (since $Q(D) = \xi(D)$ must hold for all $D \models \mathcal{A}$), this implies that $Q \equiv_{\mathcal{A}} \xi$ iff $\forall Y \exists Z\, \psi(\mu_X, Y, Z)$ is true, where μ_X is the chosen truth-assignment for V.

(3) We show that $\mathsf{VBRP}(\mathsf{FO})$ 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 $\mathcal{A}$, $\mathcal{V}$ 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 $\mathcal{A}$, bound M and views $\mathcal{V}$, while queries Q and instances D of $\mathcal{R}$ vary.

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

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

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 $\mathcal{A} = \mathcal{V} = \emptyset$ and $M = 1$. □

We now study the impact of parameters on VBRP for CQ, UCQ and $\exists\mathsf{FO}^+$. 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 becomes simpler for these classes of positive queries, but only 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 1). From this the corollary below follows.

Corollary 6: 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 $\mathcal{A}$ $\mathcal{L}$ is CQ, UCQ or $\exists\mathsf{FO}^+$. □

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 predefined. 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 $\mathsf{coNP} \vee \bigvee_{i=1}^{k}(\mathsf{NP} \wedge \mathsf{coNP})$ [38]. Here $\mathsf{NP} \wedge \mathsf{coNP}$ is also known as D^P, where a languages L' is in D^P iff 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 $\Delta_2^p = P^{\mathsf{NP}}$. More specifically, it is known that a language L is in $\mathsf{C}^\mathsf{P}_{2k+1}$ iff there exist $2k+1$ languages $L_0, L_1, \ldots, L_{2k}$, such that each L_i is in NP, $L_0 \supseteq L_1 \supseteq L_2 \supseteq \cdots \supseteq L_{2k}$ and $L = \bar{L}_0 \cup \bigcup_{i=1}^{k}(L_{2i-1} \cap \bar{L}_{2i})$ [38].

Theorem 7: 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 $\mathcal{A}$, for $\mathcal{L}$ as CQ, UCQ or $\exists\mathsf{FO}^+$. □

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

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

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

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

In contrast, if M is fixed while $\mathcal{R}, \mathcal{A}$ or $\mathcal{V}$ is not, there are possibly exponentially many query plans in QP_Q. As an example, when $\mathcal{V}$ is not fixed, consider a view $V(\bar{x})$ with $|\bar{x}| = m$. Then, depending on how the variables in V are instantiated in a query plan ξ with constants, *i.e.,* how V is used in ξ, there are at least 2^m different plans in QP_Q. That is why $\mathsf{VBRP}(\mathsf{CQ})$ remains Σ_3^p-hard when only $M, \mathcal{R}$ and $\mathcal{A}$ are fixed but $\mathcal{V}$ is not (Corollary 6).

(2) Lower bound. For any k, take any language $L = \bar{L}_0 \cup \bigcup_{i=1}^{k}(L_{2i-1} \cap \bar{L}_{2i})$ in $\mathsf{C}^\mathsf{P}_{2k+1}$. We build fixed $\mathcal{R}$, $\mathcal{A}$, $\mathcal{V}$ 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 $\mathcal{V}$ under $\mathcal{A}$.

The reduction uses the following properties. (a) Since each L_i is in NP, we have reductions f_i from L_i to $\mathsf{3SAT}$ such that for each string $\bar{\sigma}$, $\bar{\sigma} \in L_i$ iff $f_i(\bar{\sigma})$ is a satisfiable $\mathsf{3SAT}$ 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 $\mathcal{V}$ under $\mathcal{A}$. In the reduction, we use Q to encode the $2k+1$ $\mathsf{3SAT}$ instances $f_i(\bar{\sigma})$, and employ k unary fixed views and the empty view corresponding to the $k+1$ even number $0, 2, 4, \ldots, 2k$ 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 $\mathcal{R}, M, \mathcal{A}$ 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 8: A query Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$ iff there exists a unique maximum plan $\xi \in \mathsf{QP}_Q$ up to $\mathcal{A}$-equivalence such that $Q \sqsubseteq_{\mathcal{A}} \xi$. □

Proof sketch: Query Q has an M-bounded rewriting under $\mathcal{A}$ using $\mathcal{V}$ iff there exists a query plan $\xi \in \mathsf{QP}_Q$ such that $Q \equiv_{\mathcal{A}} \xi$. We show that when such a plan ξ exists, it is maximum and unique up to $\mathcal{A}$-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_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. As an example, query Q_0 of Example 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 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 $\mathcal{R}$, $\mathcal{A}$, 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 (Section 2) such that the query Q_ξ expressed by ξ is an ACQ. Our main conclusion is that the intractability of VBRP is robust, even for ACQ under fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$. 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 $\mathcal{R}$, $\mathcal{A}, M$ and $\mathcal{V}$. Unfortunately, VBRP remains intractable in this setting, even under quite restrictive access constraints in a fixed $\mathcal{A}$.

Theorem 9: 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) when $\mathcal{A}$ consists of a single access constraint of the form $R(A \to B, N)$ and $N \geq 2$;

(2) when $\mathcal{A}$ consists of two constraints $R(A \to B, 1)$ and $R'(\emptyset \to (E, F), N)$, and $N \geq 6$; or

(3) when $\mathcal{A}$ consists of two constraints $R((A, B) \to C, 1)$ and $R'(\emptyset \to E, N)$, 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 $\mathcal{A}$ and query Q with care such that they do not depend on the instances ϕ of L, and ϕ is a "yes" instance of K iff $Q \equiv_{\mathcal{A}} \emptyset$, a query over $\mathcal{R}$ that returns empty for any instances $D \models \mathcal{A}$ of R. In light of these, the proofs remain intact when M is any predefined number and $\mathcal{V}$ 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 $\mathcal{A}$ 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 μ of leaves of G, whether μ is extendable to a color c_i ($c_i \in \{r, g, b\}$) for all nodes v_i in V_G such that for each edge $(v_i, v_j) \in E$, $c_i \neq c_j$.

The reduction is a little complicated since we need to encode a cyclic graph with ACQ. It uses an access schema $\mathcal{A}$ consisting of a single $\phi = R(A \to B, 2)$, 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 ϕ to enforce necessary "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_i ($c_i \in \{r, g, b\}$) for each vertex v_i in V_G such that for every edge $(v_i, v_j) \in E$, $c_i \neq c_j$. Again the challenge is to encode cyclic G with ACQ. The reduction uses a set $\mathcal{A}$ with two access constraints $R(A \to B, 1)$ and $R'(\emptyset \to (E, F), 6)$, 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 $\mathcal{A}$ having $R((A, B) \to C, 1)$ and $R'(\emptyset \to E, 2)$, 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 the limited facility of ACQ, and restricted relations of $\mathcal{R}$ and constraints of $\mathcal{A}$. Hence the reduction is quite delicate. Given an instance $\psi(X)$ of 3SAT, we define Q to encode each clause in ψ 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 ψ 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($\mathcal{C}$) is tractable for sub-classes $\mathcal{C}$ of ACQ, and give an upper bound for VBRP(ACQ).

Theorem 10: 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 $Q \sqsubseteq_{\mathcal{A}} \xi$, 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, when taken together, simplify the analysis of VBRP (unless P = NP), to an extent, as opposed to the Σ_3^p-completeness of Theorem 1 and C_{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 $\mathcal{R}$, $\mathcal{A}$, M and $\mathcal{V}$ are fixed, 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 $\xi' \in \mathsf{QP}_Q$. □

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 $\xi \in \mathsf{QP}_Q$ (up to $\mathcal{A}$-equivalence) if it exists. It then checks whether $Q \sqsubseteq_{\mathcal{A}} \xi$. Its correctness is warranted by Lemma 8. For its complexity, we show that it is in PTIME to check whether QP_Q has a unique maximum plan when $\mathcal{R}, M$, $\mathcal{A}$ and $\mathcal{V}$ are fixed, by Lemma 11 (a), (b) and (d). Hence, the algorithm is in PTIME for any sub-class $\mathcal{C}$ of ACQ as long as it is in PTIME to check whether $Q \sqsubseteq_{\mathcal{A}} \xi$ for all $Q \in \mathcal{C}$. Note that when $\mathcal{C}$ is the entire class of ACQ queries, the algorithm is in coNP since it needs to call an NP oracle to check whether $Q \sqsubseteq_{\mathcal{A}} \xi$, by Lemma 11(c). □

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

Corollary 12: 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 sketch: By Theorem 10, to prove Corollary 12, it suffices to show that $Q \sqsubseteq_{\mathcal{A}} \xi$ is in PTIME. Observe the following. (a) When $\mathcal{A}$ consists of FDs only, we can "chase" the tableau of Q by access constraints of $\mathcal{A}$ in PTIME [1], and get a unique query $Q_{\mathcal{A}}$ such that the tableau of $Q_{\mathcal{A}}$ satisfies $\mathcal{A}$ and $Q \equiv_{\mathcal{A}} Q_{\mathcal{A}}$. (b) Checking $Q \sqsubseteq_{\mathcal{A}} \xi$ is equivalent to checking $Q_{\mathcal{A}} \sqsubseteq \xi$. 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 $Q_{\mathcal{A}} \sqsubseteq Q_\xi$ can be checked in PTIME for fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$. Putting these together, we have that $Q \sqsubseteq_{\mathcal{A}} \xi$ is in PTIME. □

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

Proposition 13: 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 sketch: We show that VBRP is NP-hard by reduction from 3SAT, using fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$, with $\mathcal{A}$ consisting of FDs. In contrast to Corollary 12, for a CQ Q, we can no longer compute its maximum query plan in QP_Q in PTIME; instead, it is an NP process.

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

Lemma 14: If $\mathcal{A}$ consists of FDs only, it is in PTIME to decide whether a plan $\xi \in \mathsf{QP}_Q$ conforms to $\mathcal{A}$. □

We give a PTIME algorithm to verify the lemma. □

Along the same lines as Corollary 12, one can easily 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 [6], when access constraints are not employed at all.

Theorem 10 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 (a) self-join-free CQ, *i.e.,* the class of CQ queries that contain no repeated relation names [25], and (b) 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 [37]. Using Theorem 10 and Lemma 11, one can show that VBRP is also in 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 $\mathcal{R}$, 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 queries defined over $\mathcal{R}$ that have an M-bounded rewriting using $\mathcal{V}$ under $\mathcal{A}$, up to equivalence, and (b) a class of *size-bounded* queries, which "covers" all the views of $\mathcal{V}$ that have bounded output for all instances of $\mathcal{R}$ that satisfy $\mathcal{A}$. The second class helps us effectively check bounded output when deducing topped queries. It is in 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 $\mathcal{R}$, $\mathcal{V}$ 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 $\mathcal{A}$; 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 effective syntax 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 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 $\mathcal{R}$ 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 $\mathcal{A}$;
(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 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 $\mathcal{A}$-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 $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ in PTIME; and if so, (b) to generate a bounded query plan ξ for Q using $\mathcal{V}$. The existence of these algorithms 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 $\mathcal{A}$ of access constraints can be discovered [10]. After these are in place, given an FO query Q posed on a dataset D that satisfies $\mathcal{A}$, 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 $Q(D)$ by executing ξ 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 $R(A, B)$ 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 $V_3(x, y) = \exists R(y, y) \wedge T(x, y)$. Consider FO query:

$$Q_3(z) = Q_4(z) \wedge \neg\exists w\, R(z, w), \text{ where}$$
$$Q_4(z) = \exists y \exists x\, (V_3(x, y) \wedge (x = 1)) \wedge R(y, z)$$

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

$$Q_3'(z) = Q_4(z) \wedge \neg(Q_4(z) \wedge \exists w\, R(z, w)).$$

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.,* $R(y, z)$ and $R(z, w)$. (3) A plan for a sub-query of Q_3 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 $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ to ensure that if $Q_s(\bar{x})$ has a bounded rewriting and $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, then $Q_s(\bar{x}) \wedge Q(\bar{z})$ also has a bounded rewriting.
- Function $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ to compute a natural number, which is an upper bound of the size of minimum sub-plans for sub-query $Q(\bar{z})$ in $Q_s(\bar{x}) \wedge Q(\bar{z})$.

The intuition behind these functions is as follows.

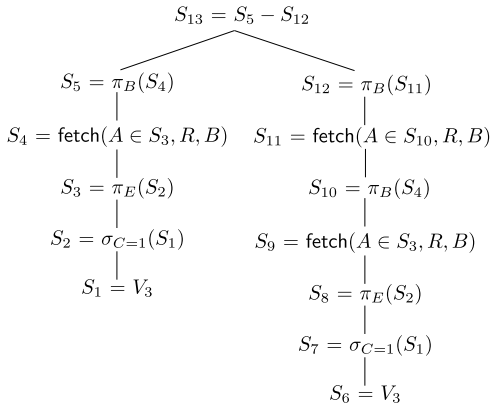

Figure 2: A bounded plan for Q_3 of Example 6

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

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

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

(c) Function $\mathsf{size}(Q_s, Q)$ helps us ensure that our query plans do not exceed a given bound M. In Example 6, $\mathsf{size}(Q_4, \exists w\, R(w, z)) = 7$, 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 $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ if

(1) $\mathsf{cov}(Q_\varepsilon, Q) = \mathsf{true}$; and

(2) $\mathsf{size}(Q_\varepsilon, Q) \le M$.

Here Q_ε is a "tautology query" such that for any query Q, $Q_\varepsilon \wedge Q = Q$, $\mathsf{cov}(Q_\varepsilon, Q_\varepsilon) = \mathsf{true}$ and Q_ε has a 0-bounded rewriting. That is, we compute $\mathsf{cov}(Q_s, Q)$ 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 $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z}))$ *and* $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$. 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 $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ 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.

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 $\forall \bar{x}\ \psi(\bar{x})$ with $\neg\exists \bar{x}(\neg\psi(\bar{x}))$; (c) only variables occur in relation atoms, *e.g.,* $R(x, 1)$ is replaced with $\exists y\, (R(x, y) \wedge y = 1)$; and (d) no relation atoms contain repeated variables, *e.g.,* we substitute $R(x, y) \wedge x = y$ for each $R(x, x)$.

The definition is separated into 7 cases below.

(1) $Q(\bar{z})$ is $z = c$. For equality atom with a constant, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ 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, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = 1$.

That is, constant queries and views have 1-bounded rewriting and hence, are topped queries.

(3) $Q(\bar{z})$ is $Q'(\bar{z}) \wedge C$, where C is $(x = y)$, $(x \neq y)$, $(x = c)$ or $(x \neq c)$. We define $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z}))$, and define $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z}))$ as $\mathsf{size}(Q_s(\bar{x}), Q'(\bar{z})) + 1$ when $\mathsf{cov}(Q_s(\bar{x}), Q'(\bar{z})) = \mathsf{true}$, and as $+\infty$ otherwise.

Note that 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 $\mu_i = \mathsf{cov}(Q_s(\bar{x}), Q_i(\bar{z}_i))$, $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}_i))$, $\mu' = \mathsf{cov}(Q_s(\bar{x}) \wedge Q_1(\bar{z}_1), Q_2(\bar{z}_2))$ and $s' = \mathsf{size}(Q_s(\bar{x}) \wedge Q_1(\bar{z}_2), Q_2(\bar{z}_2))$ $(i \in \{1, 2\})$. Then,

(a) if $\mu_1 = \mathsf{true}$ and $Q_2(\bar{z}_2)$ is of the form $\exists w\, R(\bar{z}_1, \bar{z}'_2, \bar{w})$, $R(Z_1 \to Z'_2, N)$ 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 $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + 1$; otherwise

(b) if $\mu_1 = \mu_2 = \mathsf{true}$, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s_2 + \lambda_{(\bar{z}_1, \bar{z}_2)}$, 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 $\mu_1 \wedge \mu' = \mathsf{true}$ and $|Q_2| \leq K$ for some constant K, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s' + \lambda_{(\bar{z}_1, \bar{z}_2)}$ for the same $\lambda_{(\bar{z}_1, \bar{z}_2)}$ as in (b); otherwise

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

In this case we characterize value propagation shown in Example 6, *e.g.,* y-value from $\exists x, y\,(V(x, y) \wedge (x = 1))$ to $R(y, z)$ in sub-query Q_4 of Q_3. Note that Q_s is expanded in case (c) above to propagate $\bar{z}_1$ from $Q_1 \wedge Q_s$ to Q_2. That is, if sub-query $Q_2(\bar{z}_2)$ of Q does not have a bounded rewriting with $Q_s(\bar{x})$ (*i.e.,* $s_2 = \mathsf{false}$), we may extend $Q_s(\bar{x})$ with $Q_1(\bar{z}_1)$ to make $Q_2(\bar{z}_2)$ bounded when $\mu' = \mathsf{true}$.

When $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, 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 a 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.

(b) If both Q_1 and Q_2 have bounded 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 by 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 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.,* $\bar{z}_1 \cap \bar{z}_2 \neq \emptyset$, 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 $Z_1 \cap Z_2$.

(c) If Q_1 has a bounded plan with Q_s while Q_2 has one 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 $|\xi_1| + |\xi'_2| + \lambda_{(\bar{z}_1, \bar{z}_2)}$ along the same lines as (b) above, 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. One can verify that this expansion policy assures that Q_s always has a bounded plan since we start with a tautology query $Q_s = Q_\varepsilon$.

(5) $Q(\bar{z})$ is $Q_1(\bar{z}) \vee Q_2(\bar{z})$. Let $\mu_i = \mathsf{cov}(Q_s(\bar{x}), Q_i(\bar{z})$ and $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}))$ for $i \in \{1, 2\}$. We define $\mathsf{cov}(Q_s(\bar{z}), Q(\bar{z})) = \mu_1 \wedge \mu_2$, and $\mathsf{size}(Q_s(\bar{z}), Q(\bar{z}))$ as $s_1 + s_2 + 1$ if $\mathsf{cov}(Q_s(\bar{z}), Q(\bar{z})) = \mathsf{true}$, and as $+\infty$ 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 $|\xi_1| + |\xi_2| + 1$.

(6) $Q(\bar{z})$ is $Q_1(\bar{z}) \wedge \neg Q_2(\bar{z})$. Let $\mu_i = \mathsf{cov}(Q_s(\bar{x}), Q_i(\bar{z}))$, $s_i = \mathsf{size}(Q_s(\bar{x}), Q_i(\bar{z}))$, $\mu_{12} = \mathsf{cov}(Q_s(\bar{x}), Q_1(\bar{z}) \wedge Q_2(\bar{z}))$, and $s_{12} = \mathsf{size}(Q_s(\bar{x}), Q_1(\bar{z}) \wedge Q_2(\bar{z}))$. Then

(a) if $\mu_1 \wedge \mu_2 = \mathsf{true}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ 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 $|Q_2| \leq K$ for some constant K, $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ and $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = s_1 + s_{12} + 1$; otherwise

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

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

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

(b) If $\mu_1 = \mu_{12} = \mathsf{true}$, let ξ_1 and ξ_{12} be the plans to $Q_1(\bar{z})$ 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 $|\xi_1| + |\xi_{12}| + 1$.

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

(7) $Q(\bar{z})$ is $\exists \bar{w}\, Q'(\bar{w}, \bar{z})$ ($\bar{w}$ is possibly empty). Let $\mu' = \mathsf{cov}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z}))$ and $s' = \mathsf{size}(Q_s(\bar{x}), Q'(\bar{w}, \bar{z}))$. Then,

(a) if Q' is $R(\bar{w}, \bar{z})$ and $R(\emptyset \to Z, N) \in \mathcal{A}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$ 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 $Q_s(\bar{x})$ has *bounded output* under $\mathcal{A}$, then $\mathsf{cov}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{true}$, $\mathsf{size}(Q_s(\bar{x}), Q(\bar{z})) = \mathsf{size}(Q_\varepsilon, Q_s(\bar{x})) + 1$;

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

Note 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.

(a) When $Q(\bar{z})$ is a projection of a relation atom, if it is covered by a constraint $R(\emptyset \to Z, N)$ in $\mathcal{A}$, then $\mathsf{fetch}(\emptyset, R, Z)$ is an 1-bounded plan for $Q(\bar{z})$. Otherwise if it is covered by $R(X \to Z', N)$ and $Q_s(\bar{x})$ has bounded output, then it 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. Note that Q_s has a bounded plan by induction.

(b) 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 7: We next show that Q_3 of Example 6 is topped by $(\mathcal{R}_1, \mathcal{A}_2, \mathcal{V}_3, 13)$. Denote the sub-queries of Q_3 as follows:
$q_1 = V_3(x, y) \wedge (x = 1)$, $q_2 = \exists x\, q_1$,
$q_3 = q_2 \wedge R(y, z)$ (thus $Q_4 = \exists y\, q_3$), $q_4 = \exists w\, R(z, w)$.
Then one can easily verify the following:

(a) $\mathsf{cov}(Q_\varepsilon, Q_3) = (\mathsf{cov}(Q_\varepsilon, Q_4) \wedge \mathsf{cov}(Q_\varepsilon, q_4)) \vee (\mathsf{cov}(Q_\varepsilon, Q_4) \wedge \mathsf{cov}(Q_\varepsilon, Q_4 \wedge q_4))$,

(b) $\mathsf{cov}(Q_\varepsilon, Q_4) = \mathsf{cov}(Q_\varepsilon, q_3) = (\mathsf{cov}(Q_\varepsilon, q_2) \wedge \mathsf{cov}(Q_\varepsilon, R(y, z)) \vee (\mathsf{cov}(Q_\varepsilon, q_2) \wedge \mathsf{cov}(q_2, R(y, z))$,

(c) $\mathsf{cov}(Q_\varepsilon, q_2) = \mathsf{cov}(Q_\varepsilon, q_1) = \mathsf{true}$,

(d) $\mathsf{cov}(q_2, R(y, z)) = \mathsf{true}$ (since q_2 has bounded output),

(e) from these it follows that $\mathsf{cov}(Q_\varepsilon, Q_4) = \mathsf{true}$,

(f) $\mathsf{cov}(Q_\varepsilon, Q_4 \wedge q_4) = (\mathsf{cov}(Q_\varepsilon, Q_3) \wedge \mathsf{cov}(Q_\varepsilon, q_4)) \vee (\mathsf{cov}(Q_\varepsilon, Q_4) \wedge \mathsf{cov}(Q_4, q_4)) = \mathsf{true}$.

Thus $\mathsf{cov}(Q_\varepsilon, Q_3) = \mathsf{true}$. One can easily verify that $\mathsf{size}(Q_\varepsilon, Q_3) = 13$. Thus Q_3 is topped by $(\mathcal{R}_1, \mathcal{A}_2, \mathcal{V}_3, 13)$. □

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 $\xi(Q, \mathcal{V}, \mathcal{R})$ under $\mathcal{A}$. We show that there exists a query Q_ξ topped by $(\mathcal{R}, \mathcal{V}, \mathcal{A}, M)$ such that $\xi \equiv_{\mathcal{A}} Q_\xi$. This is verified by induction on M, verifying each case and step of ξ. In this direction, $Q_s = Q_\varepsilon$ suffices; in other words, we do not need to expand Q_s as described in cases (4c) and (6b) above (to be elaborated shortly).

(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 $\mathsf{cov}(Q_s, Q) = \mathsf{true}$, 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 $Q_1(\bar{z}_1) \wedge Q_2(\bar{z}_2)$, $\mathsf{cov}(Q_s, Q(\bar{z}))$ is true and Q_s has a $\mathsf{size}(Q_\varepsilon, Q_s)$-bounded plan, we know that $\mathsf{cov}(Q_s, Q_1(\bar{z}_1))$ 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. Thus either $\mathsf{cov}(Q_s, Q_2(\bar{z}_2)) = \mathsf{true}$ or $\mathsf{cov}(Q_s \wedge Q_1(\bar{z}_1), Q_2(\bar{z}_2)) = \mathsf{true}$. In both cases, again 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, it is easy to see that both $\mathsf{cov}(Q_\varepsilon, Q)$ and $\mathsf{size}(Q_\varepsilon, Q)$ 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 $Q_s = Q_\varepsilon$ 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 evaluability of CQ [17] and for FO [10], 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 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 $\mathcal{A}$ if it is of the following form:

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

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

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

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

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 $D \models \mathcal{A}$, it is easy to see that $Q(D) = Q'(D)$.

(b) Consider a size-bounded query $Q(\bar{x})$ of the form above. Observe that for any D, if $Q'(D)$ contains more than K answer tuples, then $Q(D) = \emptyset$. Otherwise, $Q(D) = Q'(D)$ and $Q(D)$ includes at most K tuples. Putting these together, we have that $|Q(D)| \leq K$. 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. □

Queries	Complexity	Condition
FO	undecidable (Th 1)	
CQ, UCQ, $\exists$FO$^+$	Σ_3^p-complete (Th 1)	
CQ, UCQ, $\exists$FO$^+$	Σ_3^p-complete (Cor 6)	fixed $\mathcal{R}, \mathcal{A}, M$
CQ	NP-complete (Prop 13)	only FDs in $\mathcal{A}$
Fixed $\mathcal{R}, \mathcal{A}, M$ and $\mathcal{V}$ for the following		
FO	undecidable (Cor 5)	
CQ, UCQ, $\exists$FO$^+$	C^p_{2k+1}-complete (Th 7)	
CQ	NP-complete (Prop 13)	only FDs in $\mathcal{A}$
ACQ	coNP (Th 10)	
ACQ	coNP-complete (Th 9)	restricted $\mathcal{A}$
ACQ	PTIME (Cor 12)	only FDs in $\mathcal{A}$

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 $\mathcal{V}$ 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 $\mathcal{V}$ are defined, *e.g.,* when Q and $\mathcal{V}$ are CQ while Q' is a UCQ. We have only considered the setting when Q, Q' and $\mathcal{V}$ 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 evaluability [10, 17] when we allow the amount of data accessed from the underlying dataset D to be an α-fraction of D, for a small α 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 $\mathcal{V}(D)$ 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 $\mathcal{V}$ and discovering access constraints $\mathcal{A}$, such that under $\mathcal{A}$, the number of queries that have a bounded rewriting using $\mathcal{V}$ is maximized in a given application.

7. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *TCS*, 412(11):1005–1021, 2011.

[3] F. N. Afrati, C. Li, and J. D. Ullman. Using views to generate efficient evaluation plans for queries. *JCSS*, 73(5):703–724, 2007.

[4] M. Armbrust, K. Curtis, T. Kraska, A. Fox, M. J. Franklin, and D. A. Patterson. PIQL: Success-tolerant query processing in the cloud. *PVLDB*, 5(3), 2011.

[5] M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, J. Trutna, and H. Oh. SCADS: Scale-independent storage for social computing applications. In *CIDR*, 2009.

[6] M. Armbrust, E. Liang, T. Kraska, A. Fox, M. J. Franklin, and D. Patterson. Generalized scale independence through incremental precomputation. In *SIGMOD*, 2013.

[7] M. Benedikt, J. Leblay, B. ten Cate, and E. Tsamoura. *Generating Plans from Proofs: The Interpolation-based Approach to Query Reformulation*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2016.

[8] M. Benedikt, B. Ten Cate, and E. Tsamoura. Generating plans from proofs. *TODS*, 40(4), 2016.

[9] A. Calì and D. Martinenghi. Querying data under access limitations. In *ICDE*, 2008.

[10] Y. Cao and W. Fan. An effective syntax for bounded relational queries. In *SIGMOD*, 2016.

[11] Y. Cao, W. Fan, J. Huai, and R. Huang. Making pattern queries bounded in big graphs. In *ICDE*, 2015.

[12] Y. Cao, W. Fan, and W. Yu. Bounded conjunctive queries. *PVLDB*, 2014.

[13] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *PODS*, 1999.

[14] A. Deutsch, B. Ludäscher, and A. Nash. Rewriting queries using views with access patterns under integrity constraints. *TCS*, 371(3):200–226, 2007.

[15] Facebook. Introducing Graph Search. *https://en-gb.facebook.com/about/graphsearch*, 2013.

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

[17] W. Fan, F. Geerts, Y. Cao, T. Deng, and P. Lu. Querying big data by accessing small data. In *PODS*, 2015.

[18] W. Fan, F. Geerts, and L. Libkin. On scale independence for querying big data. In *PODS*, 2014.

[19] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[20] A. V. Gelder and R. W. Topor. Safety and translation of relational calculus queries. *TODS*, 16(2):235–278, 1991.

[21] G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. In *PODS*, 1999.

[22] I. Grujic, S. Bogdanovic-Dinic, and L. Stoimenov. Collecting and analyzing data from e-government facebook pages. In *ICT Innovations*, 2014.

[23] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4), 2001.

[24] Y. Huhtala, J. Kärkk ainen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[25] P. G. Kolaitis and E. Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

[26] J. Kratochvíl. Precoloring extension with fixed color bound. *Acta Math. Univ. Comen*, 62:139–153, 1993.

[27] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[28] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, 1995.

[29] C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB J.*, 12(3):211–227, 2003.

[30] A. Nash and B. Ludäscher. Processing first-order queries under limited access patterns. In *PODS*, 2004.

[31] A. Nash, L. Segoufin, and V. Vianu. Views and queries: Determinacy and rewriting. *TODS*, 35(3), 2010.

[32] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *PODS*, 1995.

[33] R. Ramakrishnan and J. Gehrke. *Database management systems*. McGraw Hill, 2000.

[34] L. J. Stockmeyer. The polynomial-time hierarchy. *TCS*, 3(1):1–22, 1976.

[35] A. P. Stolboushkin and M. A. Taitslin. Finite queries do not have effective syntax. In *PODS*, 1995.

[36] J. D. Ullman. *Principles of Database Systems, 2nd Edition*. Computer Science Press, 1982.

[37] M. Y. Vardi. On the complexity of bounded-variable queries. In *PODS*, 1995.

[38] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *TCS*, 51(1-2):53–80, 1987.

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/M025268/1, NSF III 1302212, Shenzhen Peacock Program 1105100030834361, 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.