CERTAIN ANSWERS OF EXTENSIONS OF CONJUNCTIVE QUERIES BY DATALOG AND FIRST-ORDER REWRITING

Amélie Gheerbrant[1], Leonid Libkin[2,3,4], Alexandra Rogova[1,5] and Cristina Sirangelo[1,4]

[1]*Université Paris Cité, CNRS, IRIF, F-75013, Paris, France*
[2]*School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK*
[3]*RelationalAI*
[4]*DI ENS, ENS, PSL University, CNRS, Inria Paris, France*
[5]*Data Intelligence Institute of Paris (diiP) , Inria*

Abstract

To answer database queries over incomplete data the gold standard is finding certain answers: those that are true regardless of how incomplete data is interpreted. Such answers can be found efficiently for conjunctive queries and their unions, even in the presence of constraints such as keys or functional dependencies. With negation added, the complexity of finding certain answers becomes intractable however.

In this paper we exhibit a well-behaved class of queries that extends unions of conjunctive queries with a limited form of negation and that permits efficient computation of certain answers even in the presence of constraints by means of rewriting into Datalog with negation. The class consists of queries that are the closure of conjunctive queries under Boolean operations of union, intersection and difference. We show that for these queries, certain answers can be expressed in Datalog with negation, even in the presence of functional dependencies, thus making them tractable in data complexity. We show that in general Datalog cannot be replaced by first-order logic, but without constraints such a rewriting can be done in first-order.

Keywords

Incomplete information, Certain answers, Datalog rewritings, First-order rewritings, Functional dependencies, Chase

1. Introduction

We study the classical problem of answering queries over databases with incomplete information where incompleteness is represented by means of nulls, as in the most common practice in relational databases. Such databases are required to satisfy integrity constraints, most commonly keys. This addition of constraints makes query answering more complex, even for fairly simple queries.

We consider the setting of databases with *marked nulls*, as is often required in applications such as data integration, data exchange, ontology-based data access, and others. This is markers for missing information, but the same marker may appear in multiple places. These generalize

Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

nulls in SQL and relational database management systems where repetition is not allowed; our results will thus apply to SQL nulls as well. We use the standard model of query answering, namely finding *certain answers* which are guaranteed to be true regardless of the interpretation of nulls. Over the years, we have learned that query answering is generally easy for conjunctive queries (CQ) and closely related classes, while becoming computationally infeasible for more general queries. For example, in the absence of constraints such as keys, and under the prevalent closed-world semantics used in the case of database incompleteness [1, 2], we know that:

- Certain answers to conjunctive queries and their unions can be found by naïve evaluation, i.e., the standard evaluation of queries in which nulls are treated as new distinct constants.
- This could be extended with a limited form of guarded negation [3]; in fact the limits of such naive evaluation are dictated by the notion of query preservation under homomorphisms.

Under constraints, even such simple ones as keys, the picture is less complete. We know the following:

- Certain answers to a conjunctive query Q (or a union of CQs) on a database D under key constraints Σ can be found by naïve evaluation of Q on the result of the *chase* of D with Σ. Mathematically, $\mathsf{cert}_\Sigma(Q, D) = Q(\mathsf{chase}_\Sigma(D))$, where on the left-hand side we have certain answers under constraints, and on the right hand side the naïve evaluation of Q over the result of the chase. Here chase_Σ refers to the classical textbook chase procedure with keys, or more generally functional dependencies. In fact the above result applies when Σ is a set of functional dependencies, not just keys.

Unfortunately the above result does not work when we move outside the class of select-project-join-union queries, or unions of CQs. In fact even without constraints, certain answers to a query of the form $Q_1 - Q_2$, where both Q_1 and Q_2 are CQs, are not necessarily produced by naïve evaluation. To see why, take a database containing one fact $R(1, \bot)$ where $\bot$ is a null and Q_1 returning R while Q_2 is given by a formula $R(x, y) \wedge x = y$. Here naïve evaluation of $Q_1 - Q_2$ returns R while certain answers is empty.

This motivates our question whether we can extend the class of CQs and their unions to obtain tractable evaluation of certain answers under constraints such as keys and functional dependencies. The answer is positive; in fact the query of the form $Q_1 - Q_2$ above will be an example of a query in this class. To start with, the class must be such that finding certain answers for its queries without constraints is already tractable. We know one such class: it consists of arbitrary Boolean combinations of CQs, not just their union. We shall denote it by BCCQ. It was proved in [4] that certain answers for it can be found in polynomial time, though the procedure was a tableau-based and not particularly suitable for implementation in a database system. To be implementable, certain answers should ideally be expressible in a database query language: in an ideal world, in FO (and thus basic SQL), or at least in Datalog (and thus recursive SQL).

This is precisely what we do in this paper. We establish three main results:

1. For an arbitrary BCCQ Q and a set of functional dependencies Σ one can construct a Datalog (with negation) query Q' whose naive evaluation computes $\mathsf{cert}_\Sigma(Q, D)$, thereby ensuring its polynomial-time data complexity.

2. There are however simple BCCQs, in fact even CQs, Q and keys Σ such that $\mathsf{cert}_\Sigma(Q, D)$ cannot be expressed in FO.
3. Without constraints present, certain answers to BCCQs are not only polynomial-time computable as had been shown previously, but also can be expressed in FO and thus efficiently implemented in SQL databases.

After giving preliminaries in the next Section, the following three sections address these items, respectively.

2. Preliminaries

Incomplete databases and constraints

We represent missing information in relational databases in the standard way using nulls [5, 1, 6]. Incomplete databases are populated by *constants* and *nulls*, coming respectively from two countably infinite sets Const and Null. We denote nulls by $\perp$, sometimes with sub- or superscript. We also allow them to repeat, thus adopting the model of *marked* nulls, as customary in the context of applications such as OBDA or data integration and exchange. A relational schema, or vocabulary σ, is a set of relation names with associated arities. A database D over σ associates to each relation name of arity k in σ, a k-ary relation which is a finite subset of $(\mathsf{Const} \cup \mathsf{Null})^k$. Sets of constants and nulls occurring in D are denoted by $\mathsf{Const}(D)$ and $\mathsf{Null}(D)$. A database is complete if it contains no nulls, i.e. $\mathsf{Null}(D) = \emptyset$. The *active domain* of D is the set of all values appearing in D, i.e. $\mathrm{adom}(D) = \mathsf{Const}(D) \cup \mathsf{Null}(D)$.

A *valuation* $v : \mathsf{Null}(D) \to \mathsf{Const}$ on a database D is a map that assigns constant values to nulls occurring in D. By $v(D)$ and $v(\bar{a})$ we denote the result of replacing each null $\perp$ by $v(\perp)$ in a database D or in a tuple $\bar{a}$. The semantics $[\![D]\!]$ of an incomplete database D is the set $\{v(D) \mid v \text{ is a valuation on } D\}$ of all complete databases it can represent. Here as is common in research on incomplete data, we use closed world assumption [1, 7] (i.e., everything we don't know to be true is automatically assumed to be false and no new tuple can be added).

A *functional dependency* over a relation name R is a first order sentence of the form $\forall \bar{x}, \bar{y} z \;\; (R(\bar{x}, \bar{y}, z) \wedge R(\bar{x}, \bar{y}', z') \to z = z')$.

Throughout this paper we will assume that a set of functional dependencies Σ is associated with the database schema σ.

A valuation v is *consistent* with Σ (or just *consistent*, when Σ is clear from the context) if $v(D) \models \Sigma$. We denote by $\mathsf{V}(D)$ the set of all consistent valuations defined on D.

Query answering

An m-ary *query* Q of active domain $C \subseteq \mathsf{Const}$ is a map that associates with a database D a subset of $(\mathrm{adom}(D) \cup C)^m$. To answer an m-ary query Q over an incomplete database D we follow [8] and adopt a slight generalisation of the usual intersection based certain answers notion, defined as $\cap_v Q(v(D))$. The set of *certain answers* to Q over D is

$$\mathsf{cert}_\Sigma(Q, D) = \{\bar{a} \in \mathrm{adom}(D)^m \;\mid\; v(\bar{a}) \in Q(v(D)) \text{ for all consistent } v\}\,.$$

For queries that explicitly use constants, we shall expand this to allow $\bar{a}$ range over $\text{adom}(D)$ and those constants. The only difference with the usual notion is that we allow answers to contain nulls, to avoid pathological situations when answers known with certainty are not returned (e.g., in a query returning a relation R one would expect R to be returned while the intersection-based certain answer will only return null-free tuples).

We study the certain answers problem from the data complexity perspective, fixing the query:

PROBLEM:	$\text{CERTAINANSWER}_\Sigma(Q)$
INPUT:	A database D and a tuple $\bar{a}$
QUESTION:	Is $\bar{a} \in \mathsf{cert}_\Sigma(Q, D)$?

For arbitrary FO queries and set of FDs, under the closed world semantics, the data complexity of finding certain answers is coNP-complete (to show $\bar{a} \notin \mathsf{cert}_\Sigma(Q, D)$ it is enough to guess a valuation v with $v(D) \models \Sigma$ and $v(D) \not\models Q(v(\bar{a}))$); the problem is coNP-hard even when Σ is empty [9].

Query languages

Here we shall study certain answers to *first-order* (FO) queries by means of their rewriting in *Datalog*. FO queries of vocabulary σ use atomic relational and equality formulae and are closed under Boolean connectives $\wedge, \vee, \neg$ and quantifiers $\exists, \forall$. We write $\varphi(\bar{x})$ for an FO-formula φ with free variables $\bar{x}$. With slight abuse of notation, $\bar{x}$ will denote both a tuple of variables and the set of variables occurring in it. The set of constants used by φ is denoted by $\text{adom}(\varphi)$. We interpret FO-formulas under active domain semantics, i.e. quantified variable"s range over $\text{adom}(D) \cup \text{adom}(\varphi)$. Thus, an FO formula $\varphi(\bar{x})$ represents a query (of active domain $\text{adom}(\varphi)$) mapping each database D into the set of tuples $\{\bar{t} \text{ over } \text{adom}(D) \cup \text{adom}(\varphi) \mid D \models \varphi(\bar{t})\}$.

A *Datalog rule* [5] is an expression of the form $R_1(u_1) \leftarrow R_2(u_2), \ldots, R_n(u_n)$ where $n \geq 1$, $R_1, \ldots, R_n$ are relation names and $u_1, \ldots, u_n$ are free tuples of appropriate arities. Each variable occurring in u_1 must occur in at least one of $u_2, \ldots, u_n$. A *Datalog program* is a finite set of Datalog rules. The *head* of the rule is the expression $R_1(u_1)$; and $R_2(u_2), \ldots, R_n(u_n)$ forms the body. The semantics is the standard fixed-point semantics.

As the language of our rewritings, we shall be using a fragment of *stratified Datalog with negation* in bodies that can be seen in two different ways.

1. A program is evaluated in two steps. First, we can have a Datalog program P defining new idb predicates $S_1, \ldots, S_\ell$. Then we ask an FO query over the schema extended with these predicates $S_1, \ldots, S_\ell$.
2. We evaluate a stratified Datalog with negation program in which the first stratum has no negation (but may have recursion) and the second stratum has no recursion (but may have negation).

From the rewritings we produce it will be clear that they fall in these classes. The key point about them is that they can be implemented in recursive SQL, and that they both have PTIME data complexity, making them feasible.

Naïve evaluation and certain answers

For a query Q written in FO or Datalog, we write $Q(D)$ to mean that such a query is evaluated naïvely. That is, if D contains nulls, nulls of D are treated as new constants in the domain of D, distinct from each other, and distinct from all the other constants in D and φ. For example the query $\varphi(x, y) = \exists z\,(R(x, z) \wedge R(z, y))$, on the database $D = \{R(1, \bot_1), R(\bot_1, \bot_2), R(\bot_3, 2)\}$ selects only the tuple $(1, \bot_2)$.

There are known connections between naïve evaluation and certain answers. If Σ is empty and Q is a union of conjunctive queries, then $\mathsf{cert}_\Sigma(Q, D) = Q(D)$, see [1]. If Σ contains a set of FDs, then $\mathsf{cert}_\Sigma(Q, D) = Q\big(\mathsf{chase}_\Sigma(D)\big)$; cf. [10]. Here chase_Σ refers to the standard chase procedure with a set of FDs [5].

3. Datalog Rewriting

Recall that *conjunctive queries* (CQs) are given by the $\exists, \wedge$-fragment of FO, and their unions (UCQs) by the $\exists, \wedge, \vee$-fragment of FO; these are also captured by the positive fragment of relational algebra (select-project-union-join queries).

To extend tractability results for certain answers to CQs and UCQs, we extend them with a mild form of negation (since adding negation leads to coNP-hardness of certain answers). This mild form comes in the shape of *Boolean combination of conjunctive queries (BCCQs)*, i.e., the closure of conjunctive queries under operations $q \cap q'$, $q \cup q'$, and $q - q'$.

If there are no constraints in Σ, finding certain answers to BCCQs is known to be tractable [4], though by tableau-based techniques that are hard to implement in a database system. We now extend this in two ways. First, we show that tractability is preserved even in the presence of functional dependencies (and thus keys). Second, we show that certain answers can be obtained by rewriting into a fragment of Datalog as described in Section 2. In particular, it means that certain answers can be found by a query expressible in recursive SQL.

Building on constraint-free rewriting techniques from [11], we start by putting each conjunctive query in a normal form which eliminates repetition of variables, by introducing new equality atoms.

Definition 3.1 (NRV normal form)**.** *A conjunctive query Q is in* non-repeating variable normal form *(NRV normal form) whenever it is of the form $Q(\bar{x}) = \exists \bar{w}\,(q(\bar{w}) \wedge e(\bar{x}, \bar{w}))$ where variables in $\bar{x}\bar{w}$ are pairwise distinct, and:*

- *$q(\bar{w})$ is a conjunction of relational atoms without constants, where each free variable in $\bar{w}$ has at most one occurrence in q,*
- *$e(\bar{x}, \bar{w})$ is a conjunction of equality atoms, possibly using constants, where each variable of $\bar{x}$ is involved in at least one equality.*

We say that $q(\bar{w})$ is the relational subquery *of Q, and $e(\bar{x}, \bar{w})$ is the* equality subquery *of Q.*

A BCCQ is in NRV normal form if it is a Boolean combination of CQs in NRV normal form.

Clearly every CQ Q is equivalent to a query in NRV normal form; moreover Q can be easily rewritten in NRV normal form (in linear time in the size of the query). Thus, in what follows,

we assume w.l.o.g. that CQs are given in NRV normal form. Intuitively the NRV normal form allows us to separate the two ingredients of a CQ : the existence of facts in some relations of the database on the one side, and a set of equality conditions on data values occurring in these facts, on the other side. The existence of facts does not depend on the valuation of nulls, and thus can be directly tested on the incomplete database. Instead, equality atoms in an NRV normal form imply conditions that valuations need to satisfy in order for the query to hold.

Given a query Q, a database D, and a tuple $\bar{a}$ over $\mathrm{adom}(D) \cup \mathrm{adom}(Q)$ we let the support of $\bar{a}$ be the set of all valuations that witness it :

$$\mathsf{Supp}(Q, D, \bar{a}) = \{v \in \mathsf{V}(D) \mid v(\bar{a}) \in Q(v(D))\}$$

In order to look for rewritings of BCCQs, a key observation is that $\bar{a}$ is a certain answer to Q iff $\mathsf{Supp}(\neg Q, D, \bar{a}) = \emptyset$. When Q is a BCCQ, so is $\neg Q$, thus we look for ways of expressing (non-)emptiness of the support for BCCQs.

We start by concentrating on the support of equality subqueries. This will be encoded in Datalog and then integrated, as a key ingredient, in the rewriting of the whole query. We let $\gamma(\bar{y})$ be an arbitrary set of equality atoms among variables $\bar{y}$ and possibly constants. Intuitively we will be interested in the case that $\gamma(\bar{y})$ is the equality subquery $e(\bar{x}, \bar{w})$ of a CQ in NRV normal form (thus notice that in the Datalog program below $\bar{y}$ encompasses variables $\bar{x}\bar{w}$ of an equality subquery).

Membership in the set $\mathrm{adom}(D) \cup \mathrm{adom}(\gamma)$ can be expressed by a UCQ formula that we call $Dom(x)$. We encode equivalence of database elements in $\mathrm{adom}(D) \cup \mathrm{adom}(\gamma)$ w.r.t. a set of equalities $\gamma(\bar{y})$ using the following Datalog program [1]:

$$\begin{aligned}
&equiv_\gamma(\bar{y}, z, z) \leftarrow \wedge_i\, Dom(y_i), Dom(z)\\
&equiv_\gamma(\bar{y}, z, z') \leftarrow z = y_k, z' = y_l, \wedge_i\, Dom(y_i) \ \text{ for each } (y_k = y_l) \in \gamma\\
&equiv_\gamma(\bar{y}, z, z') \leftarrow equiv_\gamma(\bar{y}, z, u), equiv_\gamma(\bar{y}, u, z')\\
&equiv_\gamma(\bar{y}, z, z') \leftarrow equiv_\gamma(\bar{y}, z', z)\\
&equiv_\gamma(\bar{y}, z, z') \leftarrow R(\bar{u}, \bar{v}, z),\ R(\bar{u}', \bar{v}', z'),\ \ \wedge_i\, equiv_\gamma(\bar{y}, u_i, u_i')\\
&\qquad \text{for each FD } (R(\bar{u}, \bar{v}, z),\ R(\bar{u}, \bar{v}', z') \rightarrow z = z') \in \Sigma
\end{aligned}$$

Intuitively, if $\bar{t}$ is a tuple of database elements assigned to $\bar{y}$, equivalent elements of D are the ones which should be collapsed into a single value in order for a valuation of D to satisfy all the equalities $\gamma(\bar{t})$ and the FDs. For fixed γ and $\bar{t}$, the relation $\{(s, s') \mid D \models equiv_\gamma(\bar{t}, s, s')\}$ is an equivalence relation over $\mathrm{adom}(D) \cup \mathrm{adom}(\gamma)$ where each element of $\mathrm{adom}(D)$ neither in $\bar{t}$ nor in $\mathrm{adom}(\gamma)$ forms a singleton equivalence class.

The formula $equiv_\gamma$ is a key ingredient in our rewriting; as formalized in the following lemma, it selects precisely the pairs of elements that a consistent valuation needs to collapse to satisfy a set of equalities. In Lemmas 3.2, 3.5 and Propositions 3.3, 3.4 we use some of the machinery developed in [11] and thus the proofs of those statements, which are adaptations of proofs in [11] are omitted.

[1] Queries we write hereafter can be domain dependent. So it is important to recall that we always use active domain semantics.

Lemma 3.2. *Let $\gamma(\bar{y})$ be a conjunction of equality atoms, D a database, and $\nu(\bar{y}) = \bar{t}$ an assignment over adom$(D) \cup$ adom(γ). Assume v is a consistent valuation of nulls, then $v(D) \models \gamma(v(\bar{t}))$ if and only if $v(s) = v(s')$ for all s, s' such that $D \models equiv_\gamma(\bar{t}, s, s')$.*

Formulas we write in the remainder are over signature $\sigma \cup Null$, where σ is the database schema. In any incomplete database D over $\sigma \cup Null$, $Null$ is always interpreted by the set of nulls occurring in D (in accordance with the semantics of the SQL construct `IS NULL`). I.e. we allow rewritings to test whether a database element is null or not.

For $\gamma(\bar{y})$ a conjunction of equality atoms, using $equiv_\gamma$ we define a new formula $comp_\gamma(\bar{y})$ stating the existence of a consistent valuation that collapses all equivalent elements of a tuple:

$$comp_\gamma(\bar{y}) \;:=\; \forall z z' (equiv_\gamma(\bar{y}, z, z') \wedge \neg Null(z) \wedge \neg Null(z') \rightarrow z = z')$$

Proposition 3.3. *Let $\gamma(\bar{y})$ be a conjunction of equality atoms, D a database, and $\nu(\bar{y}) = \bar{t}$ an assignment over adom$(D) \cup$ adom(γ), then $D \models comp_\gamma(\bar{t})$ if and only if there exists a consistent valuation v of nulls such that $v(D) \models \gamma(v(\bar{t}))$.*

We are now ready to define a formula capturing the inclusion of supports between two conjunctions of equality atoms, which will be a crucial ingredient in our rewriting. Let $\gamma(\bar{x})$ and $\gamma'(\bar{y})$ be conjunctions of equality atoms with adom$(\gamma) =$ adom(γ'). We define :

$$imply_{\gamma,\gamma'}(\bar{x}, \bar{y}) \;:=\; \forall z z' \; (equiv_{\gamma'}(\bar{y}, z, z') \rightarrow equiv_\gamma(\bar{x}, z, z'))$$

Using Proposition 3.3 and Lemma 3.2 we obtain :

Proposition 3.4. *Let $\gamma(\bar{x})$, $\gamma'(\bar{y})$ be conjunctions of equality atoms with adom$(\gamma) =$ adom(γ'), D a database and $\nu(\bar{y}) = \bar{t}$, $\nu'(\bar{y}) = \bar{t}'$ assignments over adom$(D) \cup$ adom(γ). Then $D \models imply_{\gamma,\gamma'}(\bar{t}, \bar{t}') \vee \neg comp_\gamma(\bar{t})$ iff for all consistent valuations v, one has $v(D) \models \gamma(v(\bar{t}))$ implies $v(D) \models \gamma'(v(\bar{t}'))$.*

So far, we have dealt with equality subqueries and we have characterized the emptiness and inclusion of their supports (cf. Proposition 3.3 and Proposition 3.4, respectively). We can now use this machinery to characterize the support of a BCCQ. We start by expressing membership in the support of an individual CQ :

Lemma 3.5. *Let D be a database, v a consistent valuation of D and $Q(\bar{x})$ a conjunctive query in NRV-normal form, with relational subquery $q(\bar{w})$ and equality subquery $\gamma(\bar{x}, \bar{w})$. Then $v \in$ Supp$(Q, D, \bar{r})$ if and only there exists $\bar{s}$ such that $D \models q(\bar{s}) \wedge comp_\gamma(\bar{r}\bar{s})$ and $v(D) \models \gamma(v(\bar{r}\bar{s}))$.*

In the remainder we consider BCCQs $Q(\bar{x}) \;:=\; Q_1(\bar{x}) \vee \ldots \vee Q_n(\bar{x})$ in NRV disjunctive normal form (DNF) where for all $1 \leq i \leq n$:

$$Q_i \;:= Q_{i_0}(\bar{x}) \wedge \neg Q_{i_1}(\bar{x}) \wedge \ldots \wedge \neg Q_{i_m}(\bar{x})$$

and for all $1 \leq j \leq m$:

$$Q_{i_j} \;:= \exists \bar{w}_{i_j} q_{i_j}(\bar{w}_{i_j}) \wedge \gamma_{i_j} \text{ with } \gamma_{i_j} := e_{i_j}(\bar{x}\bar{w}_{i_j})$$

For convenience, we assume w.l.o.g every conjunction of literals to be of the same length m. We can also assume without loss of generality that for each i we have $\text{adom}(\gamma_{i_j}) = \text{adom}(\gamma_{i_0})$ for all j. In fact we can always pad any γ_{i_j} with dummy equalities $c = c$ to extend its active domain.

Given a disjunct Q_i in a BCCQ in DNF, we now define $poss_{Q_i}$, encoding the set of possible answers to Q_i, and $cons_{Q_i}$, checking the compatibility of an answer with the negative literals in Q_i.

$$poss_{Q_i}(\bar{x}\bar{w}) := q_{i_0}(\bar{w}) \wedge comp_{\gamma_{i_0}}(\bar{x}\bar{w}) \wedge cons_{Q_i}(\bar{x}\bar{w})$$

$$cons_{Q_i}(\bar{x}\bar{w}) := \bigwedge_{1\leq j\leq m} \forall \bar{w}'((q_{i_j}(\bar{w}') \wedge comp_{\gamma_{i_j}}(\bar{x}\bar{w}')) \rightarrow \neg imply_{\gamma_{i_0},\gamma_{i_j}}(\bar{x}\bar{w}, \bar{x}\bar{w}'))$$

Using these new formulae, we show that the non-emptiness of $\text{Supp}(Q(\bar{x}), D, \bar{r})$ can be expressed as the existence of a possible answer.

Proposition 3.6. *Let D be a database and $Q(\bar{x})$ a DNF BCCQ in NRV normal form, then* $\text{Supp}(Q(\bar{x}), D, \bar{r}) \neq \emptyset$ *if and only if* $D \models \bigvee_{1\leq i\leq n} \exists \bar{w}\ poss_{Q_i}(\bar{r}\bar{w})$.

Proof. $\Leftarrow$ Let $D \models \bigvee_{1\leq i\leq n} \exists \bar{w}\ poss_{Q_i}(\bar{r}\bar{w})$, then there exists $1 \leq i \leq n$ and an assignment ν with $\nu(\bar{w}) = \bar{s}$, $D \models q_{i_0}(\bar{s}) \wedge comp_{\gamma_{i_0}}(\bar{r}\bar{s})$ and for all $1 \leq j \leq m, \bar{s'}$ such that $D \models q_{i_j}(\bar{s'}) \wedge comp_{\gamma_{i_j}}(\bar{r}\bar{s'})$, one has $D \models \neg imply_{\gamma_{i_0},\gamma_{i_j}}(\bar{r}\bar{s}, \bar{r}\bar{s}')$. Since $D \models comp_{\gamma_{i_0}}(\bar{r}\bar{s})$, it's easy to see that for each $s \in adom(D) \cup \text{adom}(\gamma_{i_0})$ there exists at most one constant c such that $D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, s, c)$. In fact if for constants c_1 and c_2, $D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, s, c_1)$ and $D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, s, c_2)$, by transitivity $D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, c_1, c_2)$, implying $c_1 = c_2$.

Using this observation we now build a consistent valuation v^* having the following "tightness" property : for all $s, s' \in \text{adom}(D) \cup \text{adom}(\gamma_{i_0})$, we have $v^*(s) = v^*(s')$ iff $D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, s, s')$. To build v^* we associate to each equivalence class $\mathcal{C}$ of the relation $\{(s, s') \mid D \models equiv_{\gamma_{i_0}}(\bar{r}\bar{s}, s, s')\}$, a new fresh constant $c_{\mathcal{C}}$ outside $\text{adom}(D) \cup \text{adom}(\gamma_{i_0})$. Then v^* can be defined as follows. For $s \in \text{adom}(D)$, if $D \models equiv_{\gamma}(\bar{r}\bar{s}, s, c)$, for some constant c, then $v^*(s) = c$; otherwise $v^*(s) = c_{\mathcal{C}}$ where $\mathcal{C}$ is the equivalence class of s. Consistency of v^* derives from the tightness property, and the fact that $equiv_{\gamma_{i_0}}$ satisfies the last rule of the Datalog program that defines it. Moreover by Lemma 3.2, $v^*(D) \models \gamma_{i_0}(v^*(\bar{r}\bar{s}))$ and we can prove the following claim :

Claim 3.7. *For all conjunction of equalities $\gamma'(\bar{y})$ with $adom(\gamma') = adom(\gamma_{i_0})$ and all $\bar{t}$ over $adom(D) \cup adom(\gamma_{i_0})$, one has $v^*(D) \models \gamma'(v^*(\bar{t}))$ iff for all consistent valuations v, $v(D) \models \gamma_{i_0}(v(\bar{r}\bar{s}))$ implies $v(D) \models \gamma'(v(\bar{t}))$.*

Now fix some arbitrary $j \geq 1$ and $\bar{s'}$ with $D \models q_{i_j}(\bar{s'}) \wedge comp_{\gamma_{i_j}}(\bar{r}\bar{s'})$. By Proposition 3.4, it follows from $D \models \neg imply_{\gamma_{i_0},\gamma_{i_j}}(\bar{r}\bar{s}, \bar{r}\bar{s'}) \wedge comp_{\gamma_{i_0}}(\bar{r}\bar{s})$ that there exists a consistent valuation v' with $v'(D) \models \gamma_{i_0}(v'(\bar{r}\bar{s}))$ but $v'(D) \not\models \gamma_{i_j}(v'(\bar{r}\bar{s'}))$. By the above claim $v^*(D) \not\models \gamma_{i_j}(v^*(\bar{r}\bar{s'}))$. In summary we have :

(i) $D \models q_{i_0}(\bar{s}) \wedge comp_{\gamma_{i_0}}(\bar{r}\bar{s})$ and $v^*(D) \models \gamma_{i_0}(v^*(\bar{r}\bar{s}))$ and so by Lemma 3.5, we have $v^* \in \mathsf{Supp}(Q_{i_0}(\bar{x}), D, \bar{r})$, i.e., $v^*(D) \models Q_{i_0}(v^*(\bar{r}))$.

(ii) For all $1 \leq j \leq m$ and assignment ν' with $\nu'(\bar{w}) = \bar{s'}$, if $D \models q_{i_j}(\bar{s'}) \wedge comp_{\gamma_{i_j}}(\bar{r}\bar{s'})$ then $v^*(D) \not\models \gamma_{i_j}(v^*(\bar{r}\bar{s'}))$ and so by Lemma 3.5, we have $v^* \notin \mathsf{Supp}(Q_{i_j}(\bar{x}), D, \bar{r})$, i.e., for all $1 \leq j \leq m$, $v^*(D) \models \neg Q_j(v^*(\bar{r}))$.

This means we have $v^* \in \mathsf{Supp}(Q_{i_0}(\bar{x}) \wedge \neg Q_{i_1}(\bar{x}) \wedge \ldots \wedge \neg Q_{i_m}(\bar{x}), D, \bar{r})$ for all $1 \leq i \leq n$ and so $v^* \in \mathsf{Supp}(Q(\bar{x}), D, \bar{r})$.

$\Rightarrow$ Let $v \in \mathsf{Supp}(Q(\bar{x}), D, \bar{r})$, so v is consistent and there is some $1 \leq i \leq n$ with : (i) $v \in \mathsf{Supp}(Q_{i_0}, D, \bar{r})$, (ii) for all $1 \leq j \leq m$, $v \notin \mathsf{Supp}(Q_{i_j}, D, \bar{r})$. Using Lemma 3.5 (i) implies that there exists $\bar{s}$ such that $D \models q_{i_0}(\bar{s}) \wedge comp_{\gamma_{i_0}}(\bar{r}\bar{s})$ and $v(D) \models \gamma_{i_0}(v(\bar{r}\bar{s}))$. Again by Lemma 3.5, (ii) implies that for all $1 \leq j \leq m$ and $\bar{s'}$, if $D \models q_{i_j}(\bar{s'}) \wedge comp_{\gamma_{i_j}}(\bar{r}\bar{s'})$ then $v(D) \not\models \gamma_{i_j}(v(\bar{r}\bar{s'}))$. This entails by Proposition 3.4 that $D \models comp_{\gamma_{i_0}}(\bar{r}\bar{s}) \wedge \neg imply_{\gamma_{i_0},\gamma_{i_j}}(\bar{r}\bar{s}, \bar{r}\bar{s'})$. This shows $D \models \bigvee_{1 \leq i \leq n} \exists \bar{w}\, poss_{Q_i}(\bar{r}\bar{w})$. □

Now that we have defined the formula expressing for a BCCQ Q non-emptiness of $\mathsf{Supp}(Q(\bar{x}), D, \bar{r})$ (Proposition 3.6), we can easily define a rewriting for the problem $\textsc{CertainAnswer}_\Sigma(Q)$. To do so, we rely on the fact that $\bar{r} \in \mathsf{cert}_\Sigma(Q, D)$ iff $\mathsf{Supp}(\neg Q, D, \bar{r}) = \emptyset$.

Theorem 3.8 (Datalog rewriting)**.** *Let D be a database whose schema contains a set of functional dependencies Σ, and let $Q(\bar{x})$ be a BCCQ in NRV-normal form. Let $Q' = Q'_1(\bar{x}) \vee \ldots \vee Q'_n(\bar{x})$ be $\neg Q$ in DNF normal form. Then $\bar{r} \in \mathsf{cert}_\Sigma(Q, D)$ if and only if $D \models \rho(\bar{r})$ where $\rho(\bar{x}) = \bigwedge_{1 \leq i \leq n} \forall \bar{w}\ \neg poss_{Q'_i}(\bar{x}\bar{w})$.*

Proof. One has that $\bar{r} \in \mathsf{cert}_\Sigma(Q, D)$ iff $Supp(Q', D, \bar{r}) = \emptyset$. Being Q' still a BCCQ, Proposition 3.6 tells us that $Supp(Q', D, \bar{r}) = \emptyset$ iff $D \models \bigwedge_{1 \leq i \leq n} \forall \bar{w}\ \neg poss_{Q'_i}(\bar{r}\bar{w})$. □

Corollary 3.9. *For each fixed BCCQ query Q and a set of FDs Σ, the complexity of $\textsc{CertainAnswer}_\Sigma(Q)$ is in PTIME.*

4. Non-rewritability in FO

The basic starting points for our investigation was the fact that $\mathsf{cert}_\Sigma(Q, D) = Q(\mathsf{chase}_\Sigma(D))$ for a CQ Q and a set Σ of FDs, for every database D. This remained true for unions of CQs, but failed for BCCQs, forcing us to produce a Datalog rewriting to obtain certain answers. But can a first-order rewriting be obtained instead? This would make it possible to produce certain answers using the core of SQL as opposed to its recursive features which do not always perform as well in practice.

In this section we show that the answer, in general, is negative even for CQs (and thus for BCCQs). In the next section however we show that such rewritings can be obtained in FO for BCCQs whenever Σ is empty.

The main result of this section is the following.

Theorem 4.1. *There exists a Boolean CQ Q and single FD Σ over a relational schema of binary and unary relations such that* $\mathsf{cert}_\Sigma(Q, D)$ *is not expressible as an FO query.*

Proof. Consider a schema with one binary relation E and two unary relations A and B. The only FD in Σ is $\forall x \forall y \forall z \left(E(x, y) \wedge E(x, z) \rightarrow y = z\right)$; in other words, the first attribute of E is a key. The query Q is a Boolean CQ $\exists x \, (A(x) \wedge B(x))$.

To prove inexpressibility of $\mathsf{cert}_\Sigma(Q, \cdot)$ in FO, for each $n > 0$ we create two databases D_n and D'_n. In both of them, E is interpreted as a disjoint union $T_1 \cup T_2$ where T_1 and T_2 are balanced binary trees of depth n in which all nodes are distinct nulls. In both A and B are singleton sets. In D_n, the set A contains a leaf of T_1 and B contains a leaf of T_2. In D'_n, both A and B contain leaves of T_1 such that their only common ancestor in the tree is the root (in other words, they are leaves of subtrees rooted at different children of the root of T_1).

Because of the constraint Σ, for every valuation v such that the resulting database satisfies it we have that both $v(T_1)$ and $v(T_2)$ are chains. Indeed, consider any node $\perp$ with children $\perp_1, \perp_2$ in T_i. If $v(\perp_1) \neq v(\perp_2)$ then the resulting tuples $(v(\perp), v(\perp_1))$ and $(v(\perp), v(\perp_2))$ violate the constraint. Thus $v(\perp_1) = v(\perp_2)$ and applying this construction inductively we see that $v(T_i)$ is a chain. Hence, it has a single leaf, and thus $\mathsf{cert}_\Sigma(Q, D'_n)$ is true, since A and B must be interpreted as that leaf. On the other hand, $\mathsf{cert}_\Sigma(Q, D_n)$ is false, since there is a valuation v that sends T_1 and T_2 into two disjoint chains, and thus A and B are interpreted as two distinct elements.

Assume now that $\mathsf{cert}_\Sigma(Q, \cdot)$ is rewritable as an FO sentence ϕ. Then, for every $n > 0$, we have $D'_n \models \phi$ and $D_n \models \neg\phi$. We next show that such a sentence cannot exist, thereby proving non-FO-rewritability.

Recall that in a database (with one binary relation, like considered here) a radius r neighborhood of an element a is its restriction to the set of all elements reachable from a by a path of length at most r, where the path does not take into account the orientation of edges of E (for example, if we have $E(a, b)$ and $E(c, a)$ then both b and c are in the radius 1 neighborhood of a). When two neighborhoods, of elements a and b, are isomorphic, it means that there is an isomorphism between them that sends a to b. In other words, centers of neighborhoods are viewed as distinguished elements when it comes to defining neighborhoods. It is known that each first order sentence ψ is Hanf-local [12]: that is, there exists a number $r > 0$ such that for any two databases D_1 and D_2, if there is a bijection f between D_1 and D_2 such that the radius r neighborhoods of a in D_1 and $f(a)$ in D_2 are isomorphic then D_1 and D_2 agree on ψ, i.e. either both satisfy it or both do not.

Now let r be such a number for the sentence ϕ we assumed exists. Consider D_n and D'_n and let T_{1a}, T_{1*} be the subtrees of the root of T_1 in D_n such that the first contains A while the second contains neither A not B, and let T_{2b}, T_{2*} be defined similarly for subtrees of the root of T_2 with respect to B. In D'_n we define T'_{1a}, T'_{1b} as subtrees of the root of the tree containing A, B such that the first contains the A leaf and the second contains the B leaf, while T'_{2*}, T'_{2**} be the subtrees of the root of the tree having neither A nor B elements. Then it is easy to see that the following pairs of trees are isomorphic: T_{1a} and T'_{1a}, T_{2b} and T'_{1b}, T_{1*} and T'_{2*}, T_{2*} and T'_{2**}.

We now define the bijection f as the union of those isomorphisms plus mapping roots of trees T_i in D into roots of T_i in D'. It is an immediate observation that if $n > r + 1$ (i.e., leaves

are not in the radius r neighborhood of children of roots) then f satisfies the condition that neighborhoods of a and $f(a)$ of radius r are isomorphic. This would tell us that D_n and D'_n agree on ϕ but we know they do not. This contradiction completes the proof. □

As a corollary to the proof, we obtain the following result showing that non-recursive SQL is incapable of computing $\mathsf{cert}_\Sigma(Q, D)$ in the setting of Theorem 4.1.

Corollary 4.2. *There exists a Boolean CQ Q and single FD Σ over a relational schema of binary and unary relations such that $\mathsf{cert}_\Sigma(Q, D)$ is not expressible in the basic* SELECT-FROM-WHERE-GROUP BY-HAVING *fragment of SQL with arbitrary aggregate functions.*

This is due to the fact that queries in this fragment of SQL with grouping and aggregation can be translated into a logic with aggregate functions [13] which itself is known to be Hanf-local [14].

5. An FO rewriting

We now focus on the special case where Σ is empty. First notice that the only Datalog component in our rewriting was the $equiv_\gamma$ formula. Let $\sim_\gamma$ be the reflexive symmetric transitive closure of $\{(x, w) \mid x = w \in \gamma\}$. As shown in [11], as Σ is empty, we can rewrite as follows the $equiv_\gamma$ formula in FO, where m is the number of equivalence classes of $\sim_\gamma$:

$$equiv^{FO}_\gamma(\bar{y}, z, z') \;:=\; z = z' \;\vee \bigvee_{\substack{u_1, v_1 \ldots u_m, v_m \in\, \bar{y} \,\cup\, \mathrm{adom}(\gamma) \;\mid \\ u_i \sim_\gamma v_i \text{ for all } 1 \le i \le m}} (z = u_1 \wedge z' = v_m \wedge \bigwedge_{1 \le i < m} v_i = u_{i+1})$$

Intuitively this holds because each disjunct of $equiv_\gamma(\bar{t}, s, s')$ corresponds to a possible derivation of (s, s') in the reflexive symmetric transitive closure of $\{(\nu(x), \nu(w)) \mid x = w \in \gamma\}$, and one can prove that there is a bound only depending on γ on the number of steps of this derivation.

As a consequence, we can rewrite in FO the formula poss_{Q_i} of Section 3 encoding the set of possible answers to Q_i. It is enough to replace each occurrence of the Datalog $equiv_\gamma(\bar{y}, z, z')$ program in it by $equiv^{FO}_\gamma(\bar{y}, z, z')$. We denote by $\mathsf{poss}^{FO}_{Q_i}$ the rewriting so obtained. With this, we obtain an extension to BCCQ of the rewriting techniques proposed in [11] for UCQ.

Theorem 5.1 (FO rewriting)**.** *Let D be a database, $\Sigma = \emptyset$ and let $Q(\bar{x})$ be a BCCQ in NRV-normal form. Let $Q' = Q'_1(\bar{x}) \vee \ldots \vee Q'_n(\bar{x})$ be $\neg Q$ in DNF normal form. Then $\bar{r} \in \mathsf{cert}_\Sigma(Q, D)$ if and only if $D \models \rho(\bar{r})$ where $\rho(\bar{x}) = \bigwedge_{1 \le i \le n} \forall \bar{w}\, \neg poss^{FO}_{Q'_i}(\bar{x}\bar{w})$.*

Note that tractability of BCCQ was already proved in [4] using tableau based methods. We now refine complexity as follows.

Corollary 5.2. *For each fixed BCCQ query Q, the complexity of* CertainAnswer$_\Sigma(Q)$ *is in DLOGSPACE whenever $\Sigma = \emptyset$.*

6. Future work

Our rewriting techniques are closer to a practical implementation than the previous tableau based method from [4]. This is due to their expressibility in recursive SQL (or even non-recursive in the case of Theorem 5.1). However, while theoretically feasible, an actual implementation will need additional techniques to achieve acceptable performance. To see why, notice that the first rule in the definition of $equiv_\gamma$ creates a cross product over the full active domain, i.e., the set of all elements that appeared in the database. This of course will be prohibitively large. While this may appear to be a significant obstacle, a similar situation with computing or approximating certain answers is not new in the literature. For instance, the first approximation scheme for certain answers to SQL queries that appeared in [15] has done exactly the same, and generated very large Cartesian products even for simple queries with negation. Nonetheless, an alternative was found quickly [16] that completely avoided the need for such expensive queries, and it was shown to work well on several TPC-H queries. Thus, looking for a practical and implementable rewriting is one of the possible directions for future work.

As another open problem, we note that the query for which we have shown certain answers to be non-rewritable in FO has DLOGSPACE data complexity. Indeed the problem is essentially reachability over trees, which can be easily encoded using deterministic transitive closure [17]. To express DLOGSPACE problems, we need a language weaker than Datalog with negation. Thus, it is natural to ask whether a low complexity Datalog fragment would be sufficient to express rewritings of BCCQ, or a separating example that is PTIME-complete can be found.

Acknowledgments

We thank the anonymous referees for their useful feedback. This work was supported by ANR grants ANR-18-CE40-0031 (QUID) and ANR-21-CE48-0015 (VeriGraph) as well as EPSRC grant N023056 (MAGIC). We also acknowledge a PGSM Master grant from the FSMP.

References

[1] T. Imieliński, W. Lipski, Incomplete information in relational databases, Journal of the ACM 31 (1984) 761–791.

[2] M. Console, P. Guagliardo, L. Libkin, E. Toussaint, Coping with incomplete data: Recent advances, in: ACM PODS, ACM, 2020, pp. 33–47.

[3] A. Gheerbrant, L. Libkin, C. Sirangelo, Naïve evaluation of queries over incomplete databases, ACM Trans. Database Syst. 39 (2014) 31:1–31:42.

[4] A. Gheerbrant, L. Libkin, Certain answers over incomplete XML documents: Extending tractability boundary, ACM ToCS 57 (2015) 892–926.

[5] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Boston, MA, USA, 1995.

[6] R. van der Meyden, Logical approaches to incomplete information: a survey, in: Logics for databases and information systems, Kluwer Academic Publishers, Norwell, MA, USA, 1998, pp. 307–356.

[7] R. Reiter, On closed world data bases, in: Logic and Data Bases, 1977, pp. 55–76.
[8] W. Lipski, On relational algebra with marked nulls, in: PODS, ACM, Waterloo, Ontario, Canada, 1984, pp. 201–203.
[9] S. Abiteboul, P. C. Kanellakis, G. Grahne, On the representation and querying of sets of possible worlds, TCS 78 (1991) 158–187.
[10] S. Greco, C. Molinaro, F. Spezzano, Incomplete Data and Data Dependencies in Relational Databases, Synthesis Lectures on Data Management, Morgan & Claypool Publishers, 2012.
[11] A. Gheerbrant, C. Sirangelo, Best answers over incomplete data : Complexity and first-order rewritings, in: Proceedings of IJCAI-19, 2019, pp. 1704–1710.
[12] R. Fagin, L. J. Stockmeyer, M. Y. Vardi, On monadic NP vs. monadic co-NP, Inf. Comput. 120 (1995) 78–92.
[13] L. Libkin, Expressive power of SQL, Theor. Comput. Sci. 296 (2003) 379–404.
[14] L. Hella, L. Libkin, J. Nurmonen, L. Wong, Logics with aggregate operators, J. ACM 48 (2001) 880–907.
[15] L. Libkin, SQL's three-valued logic and certain answers, ACM Trans. Database Syst. 41 (2016) 1:1–1:28. URL: https://doi.org/10.1145/2877206. doi:10.1145/2877206.
[16] P. Guagliardo, L. Libkin, Making SQL queries correct on incomplete databases: A feasibility study, in: T. Milo, W. Tan (Eds.), Proceedings of ACM PODS, 2016, ACM, 2016, pp. 211–223.
[17] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16 (1987) 760–778. URL: https://doi.org/10.1137/0216051. doi:10.1137/0216051.