View determinacy for preserving selected information in data transformations

Wenfei Fan a, Floris Geerts a, Lixiao Zheng b,*

a University of Edinburgh, United Kingdom
b Institute of Software, Chinese Academy of Sciences, P.O. Box 8718, Beijing 100190, China

A R T I C L E   I N F O

Article history:
Received 18 April 2011
Accepted 9 September 2011
Recommended by: J. Van den Bussche
Available online 16 September 2011

Keywords:
Information preservation
Queries
Views
Rewriting
View determinacy

A B S T R A C T

When transforming data one often wants certain information in the data source to be preserved, i.e., we identify parts of the source data and require these parts to be transformed without loss of information. We characterize the preservation of selected information in terms of the notions of invertibility and query preservation, in a setting when transformations are specified as a view $V$ (a set of queries), and source information is selected by a query $Q$. We investigate the problem for determining whether transformations $V$ preserve the information selected by $Q$. (1) We show that the notion of invertibility coincides with view determinacy studied for query rewriting. (2) We establish the undecidability of the problem when either $Q$ or $V$ is in DATALOG or first-order logic, for invertibility and query preservation. (3) When $Q$ and $V$ are conjunctive queries (CQ), the problem is as hard as view determinacy for CQ queries and CQ views, an open problem. Nevertheless, we provide complexity bounds of the problem, either in PTIME or NP-complete, when $V$ ranges over subclasses of CQ (i.e., SP, SC, PC), and when $Q$ is assumed to be a minimal CQ query or not. (4) We show that CQ is complete for $L$-to-CQ rewriting when $L$ is SP, SC or PC, i.e., every CQ query can be rewritten in terms of SP, SC or PC views using a query in CQ.

© 2011 Elsevier Ltd. All rights reserved.

1. Introduction

When transforming data from a data source to a target database in practice, we often want to preserve certain information in the data source. That is, we identify certain parts of the source data and require the parts to be transformed without loss of information. For example, to migrate a customer database $D$ from one platform to another, we may want the transformation to ensure that the entire set of customers in $D$ can be retrieved from the target database. When exchanging the data with a database of domestic customers, on the other hand, we may want the transformation to warrant that all conjunctive queries about domestic customers in $D$ can still be answered by using conjunctive queries posed on the target data.

The practical need gives rise to the following questions. How should we model the preservation of selected information in data transformations? Can we effectively determine whether a given transformation preserves the information selected?

To answer these questions, this paper introduces a characterization of selected information preservation, investigates its fundamental problems and establishes their complexity bounds.

Information preservation: We propose two criteria to specify the preservation of selected information. Consider the setting in which data transformations are specified in terms of a view $V$ (a set of queries) from source to target, and the selected information is identified by a query $Q$ defined on the data source.

We say that $V$ is invertible relative to $Q$ if there exists a query $Q^{-1}$ such that for every source database $D$,
Q(D) = Q^{-1}(V(D)). Intuitively, it says that source data Q(D) selected by Q can be effectively reconstructed from the target data V(D). In other words, when Q(D) is concerned, the transformation V does not lose any information.

Consider a query language \( \mathcal{L}_q \). We say that \( V \) is query preserving relative to \( Q \) and \( \mathcal{L}_q \) if there exists a computable function \( F: \mathcal{L}_q \rightarrow \mathcal{L}_q \) such that for any query \( Q' \in \mathcal{L}_q \) and source database \( D \), \( Q'(Q(D)) = F(Q')V(D) \). Intuitively, for any \( Q' \) in \( \mathcal{L}_q \) that can be answered in \( Q(D) \), the same answer can also be found in the target \( V(D) \) by using a query in the same \( \mathcal{L}_q \); i.e., when queries in someone’s favorite languages are concerned, no information in \( Q(D) \) is lost in the transformation.

Observe that when \( Q \) is the identity query, \( Q(D) \) selects the entire data source \( D \), and invertibility and query preservation aim to preserve the information of the entire \( D \).

We investigate the connection between invertibility and query preservation. These two notions are not equivalent. The former asks for the ability to restore the selected source data \( Q(D) \), while the latter concerns the information in \( Q(D) \) that can be retrieved using queries in a particular language \( \mathcal{L}_q \). We show that when \( \mathcal{L}_q \) contains the identity query as found in most sensible relational query languages, query preservation is a stronger notion. Indeed, if \( V \) is query preserving relative to \( Q \) and \( \mathcal{L}_q \), then \( V \) is invertible relative to \( Q \). In contrast, there exist \( V, Q \) and \( \mathcal{L}_q \) such that \( V \) is invertible relative to \( Q \) but \( V \) is not query preserving relative to \( Q \) and \( \mathcal{L}_q \). We identify sufficient conditions for the two notions to coincide.

Connection with view determinacy: There is also an intimate connection between invertibility and the notion of view determinacy introduced in [1]. A view \( V \) is said to determine a query \( Q \) iff for all databases \( D_1 \) and \( D_2 \), if \( V(D_1) = V(D_2) \) then \( Q(D_1) = Q(D_2) \). That is, \( V \) provides enough information to uniquely determine the answer to \( Q \). The notion of view determinacy has proved useful in a variety of applications such as query rewriting using views, semantic caching, security and privacy [1–7].

We show that invertibility and view determinacy coincide: for any view \( V \) and query \( Q \), \( V \) is invertible relative to \( Q \) iff \( V \) determines \( Q \). Among other things, this tells us that the study of view determinacy also finds applications in preserving selected information in data transformations, and vice versa.

Complexity results: We study two problems for determining whether a transformation preserves selected information.

The invertability problem is to decide, given a view \( V \) and a query \( Q \), whether \( V \) is invertible relative to \( Q \).

The query preservation problem is to determine, given \( V \), \( Q \) and a query language \( \mathcal{L}_q \), whether \( V \) is query preserving relative to \( Q \) and \( \mathcal{L}_q \).

We parametrize the problems with various \( \mathcal{L}_s \) and \( \mathcal{L}_v \), the query languages in which selection queries \( Q \) are expressed and in which views \( V \) are defined, respectively. We consider the following \( \mathcal{L}_s \) and \( \mathcal{L}_v \): DATALOG, first-order queries (FO), and conjunctive queries (CQ). We also consider SP, PC and SC, subclasses of CQ denoted by listing the operators supported (selection, projection and Cartesian product).

We show that both problems are undecidable when one of \( \mathcal{L}_s \) and \( \mathcal{L}_v \) is CQ while the other is either DATALOG or FO. These results carry over to the problem for deciding whether \( V \) determines \( Q \). While it is known that the view determinacy problem is undecidable when \( V \) or \( Q \) is in FO [5], the results on DATALOG are new additions to the study of view determinacy.

When both \( \mathcal{L}_s \) and \( \mathcal{L}_v \) are CQ, the invertibility problem is as hard as the view determinacy problem when \( V \) and \( Q \) are in CQ, which remains open [5]. We focus on special cases when \( Q \) is a CQ query and views \( V \) are defined in SP, SC or PC. We show that the invertibility problem is in \( \text{PTIME} \) for PC views, but it becomes \( \text{NP}-\text{complete} \) for SP and SC views. Moreover, we show that the problem is also in \( \text{PTIME} \) for SP views when \( Q \) is a minimal CQ query (see, e.g., [8] for minimal CQ queries). These complexity bounds remain intact for their view determinacy counterparts. In addition, we show that these results carry over to the query preservation problem when \( \mathcal{L}_q \) is CQ.

Complete rewriting: Another notion introduced in [1] concerns the completeness of a rewriting language. In a query language \( \mathcal{L} \), a query \( Q \) can be rewritten using a view \( V \) iff there exists a query \( Q^{-1} \) in \( \mathcal{L} \) such that \( Q(D) = Q^{-1}(V(D)) \) for all databases \( D \) [1]. That is, the inverse \( Q^{-1} \) of \( Q \) is definable in \( \mathcal{L} \). Clearly, if \( Q \) can be rewritten using a view \( V \) with a query \( Q^{-1} \) in a language \( \mathcal{L} \), then \( V \) determines \( Q \), while the converse may not be true. The language \( \mathcal{L} \) is said to be complete for \( \mathcal{L}_v \)-to-\( \mathcal{L}_v \) rewritings if \( \mathcal{L} \) can be used to rewrite any query \( Q \) in \( \mathcal{L}_s \) using \( V \) in \( \mathcal{L}_v \) whenever \( V \) determines \( Q \). That is, \( \mathcal{L} \) is expressive enough to capture rewritings of \( \mathcal{L}_s \) queries using \( \mathcal{L}_v \) as long as the views determine those queries.

It is known that CQ is not complete for CQ-to-CQ rewritings [1]. Nevertheless, we show that CQ is complete for \( \mathcal{L} \)-to-\( \mathcal{C} \) rewritings when \( \mathcal{L} \) ranges over SP, PC and SC.

This work is a first step towards characterizing the preservation of selected information in data transformations. Our results reveal the connection and differences between the two notions for information preservation, namely, invertibility and query preservation. In addition, the complexity results of the paper are of interest to both the study of data transformations and research on query rewriting using views. A variety of techniques are used to prove the results, including characterizations of CQ subclasses, reductions and constructive proofs with algorithms.

Related work: Closest to this work is the study of view determinacy, introduced in [1]. A number of results have been developed for the view determinacy problem and the completeness of rewriting languages, briefly summarized as follows [4,5,1]. (1) The view determinacy problem is undecidable when either queries or views are in FO. Furthermore, FO is not complete for FO-to-FO rewritings. In fact, it has been shown that any language that is complete for FO-to-FO rewritings must be Turing-complete. (2) The problem remains undecidable for UCQ queries and UCQ views, and moreover, UCQ is not complete for UCQ-to-UCQ rewritings. Indeed, no monotonic language is complete for CQ-to-CQ rewriting. (3) It remains unknown whether the view determinacy problem is decidable when the views and queries are in CQ [5].

In light of the practical interests in CQ queries, view determinacy has been studied for a variety of special classes of CQ queries and views in [4,5,1]. It has been
shown there that the problem is decidable and that CQ is complete for rewritings in the following cases: (1) arbitrary CQ queries and Boolean CQ views; (2) arbitrary CQ queries and monadic CQ views (i.e., CQ views with only one free variable); and (3) arbitrary CQ queries and a single path CQ view, which is defined over a single binary relation and has the form \( Q(x,y) = \exists x_1, \ldots, x_k (R(x_1, x_2) \land R(x_2, x_3) \land \cdots \land R(x_{k-1}, x_k) \land R(x_k, y)) \).

Special cases of the view determinacy problem for CQ have been also studied in [2,3,6,7]. (1) The packed fragment of FO (PFO) was considered in [3], which is a generalization of the guarded fragment of FO. It was shown that PFO is complete for PFO-to-PFO rewritings, and the determinacy problem for PFO queries and PFO views is decidable in \( \text{2EXPTIME} \). Moreover, for the packed fragment of conjunctive queries (PCQ), PCQ is complete for PCQ-to-PCQ rewritings and thus the determinacy problem is decidable. These results also extend to unions of PCQs. (2) Chain CQ queries, denoted as CQ\_chain, were studied in [2], which extend path CQ queries by allowing multiple binary relations. It was shown there that determinacy is decidable for chain queries and chain views, and that FO is complete for CQ\_chain-to-CQ\_chain rewritings. (3) These results were extended in [6] to connected graph CQ queries, denoted by CQ\_graph, which are binary CQ queries whose body, if viewed as an undirected graph, is connected. It was reported there that FO is complete for CQ\_chain-to-CQ\_graph rewritings. (4) Ref. [7] studied CQ queries that are defined over unary database schemas, in which each relation has only one attribute. It was shown there that for this class of queries and views, determinacy is decidable in \( \text{PTIME} \) and CQ is complete for rewritings. Nevertheless, none of these results transfers to the cases we consider.

As observed in [5], view determinacy (invertibility) is equivalent to the notion of lossless views under the exact view assumption, which has been studied for regular path queries [9,10]. Also related is the large amount of work on query dominance, which were proposed in [13] to specify relative information capacity, and were studied for data integration [14–16]. A schema \( S \) is said to dominate another schema \( T \) if there exist schema mappings \( V \) and \( V^{-1} \) from \( S \) to \( T \) and from \( T \) to \( S \), respectively, such that for any source instance \( D \) of \( S \), \( D = V^{-1}(V(D)) \). Schema \( S \) calculously dominates \( T \) if \( S \) dominates \( T \) with \( (V, V^{-1}) \) and moreover, both \( V \) and \( V^{-1} \) are expressible in relational calculus. Clearly, dominance is a special case of invertibility when selection query \( Q \) is the identity query, and calculus dominance is the special case when \( Q \) is the identity and views are in FO. These notions were also considered in the XML settings in [17,18]. No previous results on (calculus) dominance can carry over to the cases studied in this work.

Organization: Section 2 presents the notions of query preservation, invertibility and view determinism, and investigates their connections. Section 3 states the decision problems studied in this paper. Section 4 provides the undecidability results for DATALOG and FO, followed by the decidable cases for subclasses of CQ in Section 5. Finally, Section 6 summarizes the main results of the paper and identifies open questions.

2. Selected information preservation

In this section, we first introduce the notions of query preservation, invertibility and view determinism. We then investigate the connections between these concepts.

A database schema \( R = (R_1, \ldots, R_k) \) consists of a finite set of relation symbols \( R_i \), each of which is associated with an arity \( n_i \geq 0 \). Let \( \text{dom} \) be an infinite set of values. A (database) instance \( D = (I_1, \ldots, I_k) \) of \( R \) associates with each symbol \( R_i \) a relation \( I_i \), consisting of \( n_i \)-ary tuples over \( \text{dom} \). In this paper, we only consider finite instances. We denote by \( I(R) \) the set of all instances of \( R \) that take values from \( \text{dom} \). The active domain of a relation \( I_i \), denoted by \( \text{adom}(I_i) \), is the set of values in \( \text{dom} \) that occur in \( I_i \). Similarly, for \( D = (I_1, \ldots, I_k) \) we define \( \text{adom}(D) \) as the union of \( \text{adom}(I_i) \) for \( i \in [1,k] \).

A query \( Q \) over \( R \) is defined as a computable (generic) mapping from \( I(R) \) to \( I(R) \), for some output relation \( R \). Let \( R = (R_1, \ldots, R_k) \) and \( V = (V_1, \ldots, V_l) \) be two database schemas. A view \( V \) from \( R \) to \( V \) is a set of queries \( Q_i \), from \( I(R) \) to \( I(V_i) \), one for each \( i \in [1,l] \). For a query language \( L \), we say that \( V \) is a view in \( L \), if \( Q_i \) is in \( L \) for each \( i \in [1,l] \). We refer to \( R \) and \( V \) as the input and output schema of \( V \), respectively.

2.1. Invertibility and query preservation

Let \( Q \) be a query over source schema \( R \) and let \( V \) be a view from \( R \) to \( V \). We say that \( V \) is invertible relative to \( Q \) if there exists a query \( Q^{-1} \) over \( V \) such that for every instance \( D \) of \( R \), \( Q(D) = Q^{-1}(V(D)) \). Intuitively, invertibility says that the selected part of source data, identified by \( Q \), can be recovered from the view. It does not say, however, whether the inverse \( Q^{-1} \) belongs to a certain query language or whether the inverse can be computed efficiently.

Let \( L \) be a query language. We say that a view \( V \) is query preserving relative to \( Q \) and \( L \) if there exists a computable function \( F : L \rightarrow L \) such that for any query
Proposition 1. Let $Q$ be a query, $V$ a view and $L_q$ a query language.

- If $V$ is query preserving relative to $Q$ and $L_q$, and the identity query $id$ is expressible in $L_q$, then $V$ is invertible relative to $Q$, and moreover, $Q^{-1}$ is a query in $L_q$ as well.
- If $V$ is invertible relative to $Q$, the inverse $Q^{-1}$ is expressible in $L_q$, and $L_q$ is closed under composition, then $V$ is query preserving relative to $Q$ and $L_q$.

Here a query language $L_q$ is closed under composition if for any $Q_1, Q_2$ in $L_q$, $Q_1 \circ Q_2$ (if defined) is also in $L_q$.

Proof. Suppose that $V$ is query preserving relative to $Q$ and $L_q$. Then there exists a computable function $F : L_q \to L_q$ such that for any query $Q' \in L_q$ and any instance $D$, $Q'(Q(D)) = F(Q)(V(D))$. By assumption, $id$ is expressible in $L_q$ and thus $Q(D) = id(Q(D)) = F(id)(V(D))$ for any instance $D$. That is, $Q^{-1} = F(id)$. Hence $V$ is invertible relative to $Q$ and moreover, the inverse $Q^{-1}$ is a query in $L_q$.

Suppose that $V$ is invertible relative to $Q$ and the inverse $Q^{-1}$ is in $L_q$. By assumption, $L_q$ is closed under composition and therefore, we can define a function $F : L_q \to L_q$ as $F(Q) = Q \circ Q^{-1}$ for any $Q' \in L_q$. Clearly, for any $Q' \in L_q$ and any instance $D$, $Q'(Q(D)) = Q \circ Q^{-1}(V(D)) = F(Q)(V(D))$. That is, $V$ is query preserving relative to $Q$ and $L_q$.

Observe that $id$ is definable in all commonly used relational query languages. In the sequel we consider w.l.o.g. only query languages in which $id$ is definable. Hence, the notion of query preservation is generally stronger than invertibility. This is verified by the separation result below, which we shall prove shortly.

Proposition 2. There exist a CQ query $Q$ and a view $V$ in CQ such that (1) $V$ is invertible relative to $Q$, but (2) $V$ is not query preserving relative to $Q$ and $CQ$.

2.2. View determinacy

It turns out that the notion of invertibility coincides with the notion of view determinacy [5], which we recall next. Let $Q$ be a query over source schema $R$ and let $V$ be a view from $R$ to $V$. A view $V$ determines $Q$, denoted by $V \Rightarrow Q$, iff for all instances $D_1, D_2$ of $R$, if $V(D_1) = V(D_2)$ then $Q(D_1) = Q(D_2)$.

Lemma 1. Let $Q$ be a query and $V$ a view. Then $V$ is invertible relative to $Q$ iff $V$ determines $Q$.

Proof. Suppose that $V$ is invertible relative to $Q$. Then for any pair of instances $D_1$ and $D_2$ we have that $Q(D_1) = Q^{-1}(V(D_1))$ and $Q(D_2) = Q^{-1}(V(D_2))$. Thus if $V(D_1) = V(D_2)$ then clearly $Q(D_1) = Q(D_2)$, and hence $V \Rightarrow Q$.

Conversely, suppose that $V \Rightarrow Q$. Let $\sigma$ be the mapping that associates $V(D)$ with the corresponding value of $Q(D)$, for every instance $D$. It is easily verified (see e.g., [1]) that $\sigma$ is generic, computable and furthermore can be taken as the inverse $Q^{-1}$. Hence, $V$ is indeed invertible relative to $Q$. □

The completeness of rewriting languages has also been studied in [5]. We say that a query $Q$ can be rewritten using $V$ in a language $L$ iff there exists some query $Q^{-1} \in L$ over the schema $V$ such that $Q(D) = Q^{-1}(V(D))$ for all instances $D$ of $R$. We denote this by $Q \Rightarrow_V Q^{-1}$. Observe that if $Q \Rightarrow_V Q^{-1}$ for a query $Q^{-1}$ in some query language $L$, then obviously $V \Rightarrow Q$. The converse is, however, generally not true.

Given a view language $L_V$ and a query language $L_q$, we say that a query language $L$ is a complete rewriting language for $L_V$-to-$L_q$ rewritings if for all query $Q \in L_q$ and view $V \in L_V$, $L$ can be used to rewrite $Q$ using $V$ whenever $V \Rightarrow Q$.

It is known that $CQ$ is not complete for CQ-to-CQ rewritings [5]. Capitalizing on this, we give a proof of Proposition 2.

Proof of Proposition 2. It is known that there exist a CQ query $Q$ and a CQ view $V$ such that $V \Rightarrow Q$, but the inverse $Q^{-1}$ is not definable in CQ. Such concrete examples can be found in [2,5]. Let $Q$ and $V$ be such a pair. By Lemma 1, $V$ is invertible relative to $Q$. Hence to prove Proposition 2, it suffices to show that $V$ is not query preserving relative to $Q$ and $CQ$.

Assume by contradiction that $V$ is query preserving relative to $Q$ and $CQ$. Then there exists a computable function $F : CQ \to CQ$ such that for any query $Q' \in CQ$ and any instance $D$, $Q'(Q(D)) = F(Q)(V(D))$. Let $Q'$ be $id$, then $Q(D) = F(id)(V(D))$, i.e., $Q \Rightarrow_V F(id)$, and $F(id)$ is a CQ query. This contradicts the fact that $Q^{-1}$ is not definable in CQ. □

3. Problem statements

We investigate the following decision problems. Let $L_q, L_v$ and $L_q$ be query languages. The first problem is referred to as the invertibility problem, stated as follows.

PROBLEM: \text{VDet($L_q, L_q$)}

INPUT: A query $Q \in L_q$, a view $V = \{Q_1, \ldots, Q_t\}$ defined in terms of queries in $L_v$.

QUESTION: Is $V$ invertible relative to $Q$?

By Lemma 1, \text{VDet($L_q, L_q$)} can be equivalently stated as the view determinacy problem for $(L_q, L_q)$. It is the problem to determine, given $Q, V \in L_q$ and $V = \{Q_1, \ldots, Q_t\}$ such that $Q_i \in L_q$ for $i \in [1, t]$, whether $V$ determines $Q$. We shall use these two statements interchangeably in the sequel.
We shall also consider the query preservation problem:

**Problem:** QPre($L_1, L_2, L_3$)

**Input:** A query $Q \in L_3$, a view $V = \{Q_1, \ldots, Q_n\}$ defined in terms of queries in $L_1$, and a query language $L_2$.

**Question:** Is $V$ query preserving relative to $Q$ and $L_3$?

Query languages used in this paper range over: (1) CQ, the class of conjunctive queries built up from relation atoms, by closing under conjunction $\land$ and existential quantification $\exists$; (2) FO, first-order logic queries built from atomic formulas using $\land$, disjunction $\lor$, existential quantification $\exists$ and universal quantification $\forall$; and (3) DATALOG,atalog queries defined as a collection of rules $p(x) : \neg p_1(x_1), \ldots, p_n(x_n)$, where each $p_i$ is either an atomic formula (a relation atom in $\mathcal{R}$, or equality $=$), or an IDB predicate. That is, DATALOG is an extension of union of conjunctive queries with an inflationary fixpoint operator. We refer to [8] for more details concerning these languages.

Recall that the class of conjunctive queries, CQ, is the class of SP queries built up from the relational algebra operators: selection ($S$), projection ($P$) and Cartesian product ($\times$). We also consider fragments of CQ, denoted by listing the operators allowed in the fragment. In particular, we consider the following three classes of CQ queries:

- **SP:** the fragment defined with $S$ and $P$ operators only;
- **PC:** the fragment defined with $P$ and $C$ operators only;
- **SC:** the fragment defined with $S$ and $C$ operators only.

4. Undecidability results

In this section we study VDet($L_1, L_2$) and QPre($L_1, L_2, L_3$) when $L_3$ or $L_2$ is either FO or DATALOG. The main results are negative: both problems are undecidable in these settings.

We first consider the invertibility problem for DATALOG. It is known that the view determinacy problem is undecidable when either $L_1$ or $L_3$ is FO, and when both $L_1$ and $L_3$ are UCQ [5]. By Lemma 1, the undecidability results carry over to VDet($L_1, L_2$). We next show that VDet($L_1, L_2, L_3$) (and hence view determinacy) is also undecidable when $L_2$ is DATALOG and $L_3$ is CQ, and when $L_2$ is CQ and $L_3$ is DATALOG.

**Theorem 1.** VDet($L_1, L_2$) is undecidable when

1. $L_1$ is DATALOG and $L_2$ is CQ, or
2. $L_2$ is CQ and $L_3$ is DATALOG.

**Proof.** Both proofs are by reduction from the containment problem for DATALOG, which is to determine, given two DATALOG queries $Q_1$ and $Q_2$, whether $Q_1(D) \subseteq Q_2(D)$ for every instance $D$. This problem is known to be undecidable [19].

1. VDet(DATALOG,CQ). Let $Q_1$ and $Q_2$ be two DATALOG queries defined over schema $\mathcal{R}$, with answer predicates $ans_1(x)$ and $ans_2(x)$, respectively. Let $N$ be a nullary relation symbol not appearing in $\mathcal{R}$. We define a DATALOG query $Q$ over $(\mathcal{R}, N)$ consisting of the rules of $Q_2$ and $Q_2$ together with $Q(x)$: $\neg ans_1(x), N()$.

Q(\mathcal{X}): $\neg ans_2(\mathcal{X})$

The CQ view $V$ over $(\mathcal{R}, N)$ is defined such that for any instance $(D, J_N)$ of $(\mathcal{R}, N)$, $V(D, J_N) = D$. We next show that $Q_1 \subseteq Q_2$, if $V = Q$. Suppose that $Q_1 \subseteq Q_2$. Then $Q$ is equivalent to $Q_2$. Since $V$ simply copies the instance $D$ of $\mathcal{R}$ we have that $Q = \neg V_\mathcal{Q}_2$, from which $V = Q$ follows. Conversely, if $Q_1 \not\subseteq Q_2$, then there exists an instance $D$ of $\mathcal{R}$ such that $Q_1(D) \not\subseteq Q_2(D)$. Given such $D$, we define two database instances $D_1 = (\mathcal{R}, (1))$ and $D_2 = (\mathcal{R}, \emptyset)$ of $(\mathcal{R}, N)$. Because $V(D_1) = V(D_2) = D$ but $Q_1(D_1) \neq Q_2(D_2)$, we can conclude that $V$ does not determine $Q$.

2. VDet(CQ,DATALOG). Let $Q_1$ and $Q_2$ be two DATALOG queries defined over relational schema $\mathcal{R}$, with answer predicates $ans_1(x)$ and $ans_2(x)$, respectively. Let $R_1, R_2$ be two relation symbols not appearing in $\mathcal{R}$, which have the same arity as $ans_1$ and $ans_2$. We define the DATALOG view $V$ over $(\mathcal{R}, R_1, R_2)$ as $V = \{V_1, V_2, V_3\}$, where $V_1(x): \neg R_1(x), ans_2(x), R_2(x)$

$V_2(x): \neg R_1(x)$

$V_3(x): \neg ans_1(x), R_2(x)$

We define the CQ query $Q$ such that for any instance $D' = (D_1, \mathcal{I})$ of schema $(\mathcal{R}, R_1, R_2)$, if $V_3(D')$ is nonempty, then $V_1(D')$ returns $I_1$. If $V_3(D') = \emptyset$, which means that $Q_1(D_1) \cap I_2 = \emptyset$, then from $Q_1 \subseteq Q_2$ we can conclude that $Q_1(D_1) \cap I_2 = \emptyset$ and hence $V_3(D')$ returns $I_1$. That is, $Q = \neg V_\mathcal{Q}$ and hence $V = Q$. Conversely, suppose that $Q_1 \not\subseteq Q_2$. Then there exists an instance $D$ of relational schema $\mathcal{R}$ and a tuple $T$ such that $T \in Q_1(D)$ and $T \notin Q_2(D)$. Given such $D$ and $T$, we define two instances $D' = (D, \{T\})$ and $D'' = (D, \{T\})$. It is easy to see that $V(D') = V(D'') = (\emptyset, \{T\}, \emptyset)$ but $Q(D') = (\{T\}) \neq Q(D'') = \emptyset$. Thus $V$ does not determine $Q$.

When it comes to QPre($L_1, L_2, L_3$), the query preservation problem, we show that it is also beyond reach in practice when any of $L_1$ and $L_2$ is either FO or DATALOG.

**Theorem 2.** QPre($L_1, L_2, L_3$) is undecidable when

1. $L_1$ is FO, and $L_2$ and $L_3$ are CQ.
2. $L_2$ is CQ, $L_1$ is FO and $L_3$ is CQ.
3. $L_2$ is DATALOG, $L_1$ is CQ and $L_3$ is DATALOG, or
4. $L_1$ is CQ, $L_2$ is DATALOG and $L_3$ is FO.

**Proof.** We show the undecidability of (1) and (2) by reduction from the satisfiability problem of FO, which is known to be undecidable [8]. The undecidability of (3) and (4) follows from the proofs of Theorem 1.
query preserving relative to Q and CQ iff Q0 is not satisfiable.

First assume that Q0 is not satisfiable. Then for any instance D = (I0, I1, I2) of R, Q(D) = ∅. Moreover, for any Q ∈ CQ we have that Q(∅) = ∅. Hence FI(Q) = ∅ satisfies the desired properties. Hence V is query preserving relative to Q and V.

Conversely, assume that Q0 is satisfiable. That is, there exists an instance I0 of R such that Q0(I0) ≠ ∅. By Proposition 1, it suffices to show that V does not determine Q. For it holds, then by id ∈ CQ, no function F can exist that makes V query preserving. We distinguish between the following two cases: (a) Q0(∅) = ∅ and (b) Q0(∅) ≠ ∅. For case (a), consider instances \( D_1 = (0,0,0) \) and \( D_2 = (I0,0,0) \), where Q0(I0) ≠ ∅ as assumed above. We have that Q(D1) = ∅ and Q(D2) = Q(I0) ≠ ∅ whereas \( V(D_1) = \emptyset \) and \( V(D_2) = \emptyset \). For case (b), consider instances \( D_1 = (0,0,0) \) and \( D_2 = (I,0,0) \), where I is arbitrary. Then Q(D1) ≠ ∅ and Q(D2) = ∅, whereas \( V(D_1) = \emptyset \) and \( V(D_2) = \emptyset \). Hence, V does not determine Q.

5. Decidable Cases for CQ queries

We next study VDet(CQ, L) and QPre(CQ, L, V) when \( L, L_0, L_e \) and \( L_q \) are conjunctive queries (CQ). In general, it is unknown whether the view determinacy problem is decidable for conjunctive queries [5]. We focus on special cases VDet(CQ, C) and QPre(CQ, C, C), for selection queries Q in CQ and views V in a fragment L of CQ, where L is SP, PC or SC. We show that these problems are either NP-complete or in \( \text{PTIME} \) (Theorems 3, 4, Corollaries 2 and 3), and that CQ is complete L-to-CQ rewritings (Corollary 1).

The proofs of Theorems 3 and 4 are nontrivial. To simplify the discussion, we first present some notations and lemmas that will be used throughout the proofs (Section 5.1). We then study VDet and QPre for the special case when selection queries Q are minimal CQ queries (Section 5.2). Finally we extend the results to general CQ queries (Section 5.3).

5.1. Preliminaries

We use \( R = (R_1, \ldots, R_k) \) and \( V = (V_1, \ldots, V_l) \) to denote the source and target schema, respectively. Let \( v \) be an infinite set of variables that are disjoint from \( \text{dom} \). Let Q(\( \text{R} \)) be a CQ query over \( R \) with free variables \( v \).

We consider instances over the extended domain \( \text{dom} \cup v \). More specifically, we associate with each CQ query Q an instance over this extended domain in the usual way. That is, the frozen body of Q, denoted by \([Q]\), is the instance over \( R \) such that \((x_1,...,x_n)\) belongs to the relation in \([Q]\) corresponding to \( R_1 \) iff \( R_1(x_1,...,x_n) \) is an atom in Q. Note that \((x_1,...,x_n)\) may contain both constants (from \( \text{dom} \)) and variables (from \( v \)). Similarly, for a set \( V \) of CQ queries, we use \([V]\) to denote the union of the frozen bodies \([Q]\) for all Q in \( V \).

Consider a mapping \( h \) from variables to variables and constants. Let \( \bar{T} \) be a tuple over \( \text{dom} \cup v \). Then \( h(\bar{T}) \) is defined in the usual way by applying \( h \) to each component of \( \bar{T} \). Similarly, we denote by \( h([Q]) \) the instance obtained by taking the union of \( h(\bar{T}) \) for \( \bar{T} \in [Q] \). A homomorphism \( h \) from an instance I to an instance J over the extended domain, denoted as \( h : I \rightarrow J \), is a standard homomorphism that is identity on \( \text{dom} \). More specifically, \( h(\bar{T}) \subseteq J \). Recall that for a CQ query Q(\( \text{R} \)) and an instance \( D \), a tuple \( \bar{T} \) is in \( Q(D) \) iff there exists a homomorphism \( h \) from \([Q]\) to \( D \) such that \( h(\bar{T}) = \bar{T} \).

A query \( Q_1 \) is contained in a query \( Q_2 \), denoted as \( Q_1 \subseteq Q_2 \), if for any instance \( D \), \( Q_1(D) \subseteq Q_2(D) \). Two queries are equivalent, denoted as \( Q_1 \equiv Q_2 \), if \( Q_1 \subseteq Q_2 \) and \( Q_2 \subseteq Q_1 \).

A classical result in the theory of conjunctive queries is the following Homomorphism Theorem [20]: Let \( Q_1(\{x_1\}) \) and \( Q_2(\{x_2\}) \) be two CQ queries over the same schema \( R \) with free variables \( x_1 \) and \( x_2 \), respectively. Then \( Q_1 \subseteq Q_2 \) iff there exists a homomorphism \( h \) from \([Q_2]\) to \([Q_1]\) such that \( h(\bar{x}_2) = \bar{x}_1 \), or in other words, \( Q_1 \subseteq Q_2 \) iff \( \bar{x}_1 \in Q_1(h(\bar{x}_2)) \).

The following proposition (slightly modified) from [2] states some observations.

Proposition 3. Let Q(\( \text{R} \)) be a CQ query with free variables \( v \) and let \( V \) be a set of CQ views. If \( V = \emptyset \) then \( [V(\{Q\})] = \emptyset \); and (ii) all the relation symbols appearing in \( Q \) also appear in some query in \( V \).

Our results make use of the following results on view determinacy for conjunctive queries [1,5]. Let Q(\( \text{R} \)) be a CQ query and \( V = \{V_1(\{x_1\}), \ldots, V_l(\{x_l\})\} \) be a set of views in CQ. Let \( S = (S_1, \ldots, S_l) = V(\{Q\}) \). We construct an instance D over \( \text{R} \) from S as follows: For each \( i \in \{1, \ldots, l\} \) and for every
tuples \( \Gamma \) belonging to \( S \), we include in \( D \) the tuples of \( h([V_i]) \) with \( h(x_i) = \Gamma \), where \( h \) maps every variable in \([V] \) not in \( x_i \) to some new distinct value. We call this instance the \( V \)-inverse of \( S \), denoted as \( V^{-1}(S) \). Let \( Q(V) \) be the CQ query over \( V \) with free variables \( x \) and frozen body \([Q_0] = S \). The instance \( V^{-1}(S) \) is actually obtained from \([Q_0] \) by unfolding view definitions, with bound variables in view definitions renamed to new distinct variables. That is, \( V^{-1}(S) = [Q_0] \cup V \). The following proposition (slightly modified) is from [1,5].

**Proposition 4.** Let \( Q(x) \) be a CQ query and \( V \) be a set of CQ views. Let \( S = V([[Q]]) \) and \( Q(V) \) be the CQ query with \([Q_0] = S \). We have the following: (i) if \( x \in Q(V^{-1}(S)) \), then \( Q(V) \) is a rewriting of \( Q \) in terms of \( V \), and thus \( V \Rightarrow Q \); (ii) if \( x \) has a CQ rewriting in terms of \( V \), then \( Q(V) \) is such a rewriting.

5.2. Minimal CQ queries

We first consider the invertibility problem in which selection queries \( Q \) are minimal conjunctive queries. Recall that a conjunctive query \( Q \) is minimal if removing any of the rows from \([Q] \) leads to an inequivalent conjunctive query [8]. For minimal CQ queries, we show the following:

**Theorem 3.** When the queries in \( L_c \) are minimal CQ queries, \( VDet(L_c,L_x) \) is

1. in \( \text{PTime} \) when \( L_c \) is \( PC \),
2. in \( \text{PTime} \) when \( L_c \) is \( SP \),
3. \( \text{NP-complete} \) when \( L_c \) is \( SC \). $\square$

The proof is a little involved, and consists of several parts. The \( \text{PTime} \) results are shown by leveraging Proposition 4(i). More specifically, for both cases a number of conditions on the query \( Q \) and view \( V \) are identified such that (1) \( x \) satisfies the conditions or (2) \( x \) does not satisfy the conditions, the view does not determine the query. Furthermore, these conditions can be verified in \( \text{PTime} \). The intractability of \( VDet(CQ,SC) \) is shown in two steps: First, \( \text{NP-hardness} \) is established by reduction from the graph 3-colorability problem; and second, the \( \text{NP} \) upper bound is shown to hold even when queries in \( L_c \) are not minimal. The upper bound proof is deferred to Section 5.3.

Before giving the details of the proof, we elaborate the impact of the minimality assumption for CQ queries in \( L_c \). As previously described, the \( \text{PTime} \) results rely on the identification of necessary and sufficient conditions for view determinacy. In order to show that these conditions are necessary, we show that if the conditions fail to hold, then there exist two instances \( D_1 \) and \( D_2 \) such that \( V(D_1) = V(D_2) \) but \( Q(D_1) \neq Q(D_2) \). We show next that if \( Q \) is minimal, then there is a principled way to find (in \( \text{PTime} \)) two instances \( D_1 \) and \( D_2 \) such that \( Q(D_1) \neq Q(D_2) \). We shall show in the proof of Theorem 3 that these instances can further be taken such that \( V(D_1) = V(D_2) \).

More formally, given a CQ query \( Q(x) \) and a tuple \( \Gamma \in [Q] \), we call a set \( \Delta \) of tuples critical for \( \Gamma \) and \( Q \) if the CQ queries \( Q_1(x) \) and \( Q_2(x) \) with frozen bodies \([Q_1] = [Q] \cup \Delta \) and \([Q_2] = ([Q]\{\Gamma\}) \cup \Delta \), respectively, satisfy the following two properties: (1) \( Q_1 \equiv Q \), or in other words, adding \( \Delta \) does not change the query \( Q \); and (2) \( Q_2 \equiv Q \), that is, replacing \( \Gamma \) with \( \Delta \) results in a query strictly more general than \( Q \). Critical sets of tuples allow us to construct instances on which \( Q \) differs.

**Lemma 2.** Let \( Q(x) \) be a CQ query, \( \Gamma \in [Q] \) and \( \Delta \) be a set of critical tuples for \( \Gamma \) and \( Q \). Then for \( D_1 = [Q] \cup \Delta \) and \( D_2 = ([Q]\{\Gamma\}) \cup \Delta \) we have that \( Q(D_1) \neq Q(D_2) \).

**Proof.** Let \( Q_1(x) \) and \( Q_2(x) \) be the CQ queries with frozen bodies \([Q_1] = D_1 \) and \([Q_2] = D_2 \), respectively. Assume by contradiction that \( Q(D_1) = Q(D_2) \). By assumption, \( Q_1 \equiv Q \), and hence we also have that \( Q_1([Q_1]) = Q_2([Q_2]) \). Furthermore, \( x \in Q_1([Q_1]) \) and thus also \( x \in Q_2([Q_2]) \). By the Homomorphism Theorem, \( Q_1 \equiv Q_2 \). From the assumption that \( Q \equiv Q_2 \), and hence \( Q_1 \equiv Q_2 \), we can then conclude that \( Q \equiv Q_2 \). This contradicts the fact that \( Q \equiv Q_2 \) and therefore, \( Q(D_1) \neq Q(D_2) \). $\square$

The crucial observation is that when \( Q(x) \) is a minimal query, one can construct critical sets of tuples easily. More precisely, let \( \Gamma \in [Q] \) and let \( \Delta \) be a tuple obtained from \( \Gamma \) by replacing some occurrences of (i) a constant; or (ii) a variable that appears in multiple rows in \([Q] \); or (iii) a variable that appears in \( x \), with a distinct new variable; or (iv) replacing some occurrences of a variable that appears multiple times but only in \( \Gamma \), with a distinct new variable while keeping the other occurrences of this variable unchanged. We then have the following:

**Lemma 3.** Let \( Q(x) \) be a minimal CQ query, \( \Gamma \in [Q] \) and \( \Delta \) be a set of tuples obtained from \( \Gamma \) as described in (i)–(iv). Then \( Q \) is critical for \( \Gamma \) and \( Q \).

**Proof.** We need to show that for \( Q_1(x) \) with \([Q_1] = [Q] \cup \Delta \) and \( Q_2(x) \) with \([Q_2] = ([Q]\{\Gamma\}) \cup \Delta \), we have that (a) \( Q \equiv Q_1 \) and (b) \( Q \equiv Q_2 \). For (a) it suffices to observe that \([Q] \equiv [Q_1] \) and therefore, \( Q \equiv Q_1 \). Furthermore, the trivial homomorphism \( h : [Q] \rightarrow [Q] \) can be extended to a homomorphism \( h' : [Q_1] \rightarrow [Q] \) since, by construction, every tuple \( S \in \Delta \) is equal to \( \Gamma \) except that some occurrences of a constant \( \Delta \) or variable \( \Delta \) are replaced by a new variable that does not introduce additional equality constraints. Hence, \( h((Q_1)) \equiv [Q] \), \( h'(x) = x \) and therefore, \( Q \equiv Q_1 \). We can thus conclude that \( Q \equiv Q_1 \), as desired.

For (b) we first observe that \([Q_2] \equiv [Q_1] \) and thus \( Q_2 \equiv Q_1 \). From (a) we can also infer that \( Q \equiv Q_2 \). Assume by contradiction that \( Q \equiv Q_2 \). Since \([Q_2] \equiv [Q] \), \( Q_2 \equiv Q \) and \( Q \) is minimal, there exists a subset \( T \) of \([Q_2] \) such that \([Q_2] \setminus T \) is equivalent to \( Q \) and \([Q_2] \setminus T \) is equivalent to \( Q \). We consider the following cases: (i) \([Q_2] \equiv [Q] \setminus T \) and (ii) \([Q_2] \equiv [Q] \setminus T \).

Case (i): Note that \([Q_2] \equiv [Q] \setminus T \) plus a newly constructed tuple \( S \in \Delta \). Since \( Q_2 \) and \( Q \) are minimal and equivalent, the tableaux \(([Q_2] \setminus T) \) and \(([Q] \setminus T) \) are the same up to renaming of variables (cf. [8, Proposition 6.2.9]). This is impossible, however, by the construction of tuples in \( \Delta \).

Case (ii): Observe that there exists a tuple \( \Gamma \in [Q] \setminus T \) such that \( \pi_\Delta(Q_2) \). Let \( Q'(x) \) be the CQ query with \([Q'] = [Q'] \setminus T \). It is easily verified that there exists a homomorphism \( h_1 : [Q_2] \rightarrow [Q'] \) with \( h_1(\Gamma) = x \). On the other hand,
Q ≡ Q′ and there exists a homomorphism h₂: [Q] → [Q′] with h₂(η) = η. Thus there exists a homomorphism h = h₁ ∘ h₂: [Q] → [Q′] with h(η) = η and hence, Q ≡ Q. Furthermore, from [Q] ⊆ [Q] we infer that Q ≡ Q′. Hence, Q′ ≡ Q. This, however, contradicts the assumption that Q is minimal and therefore, Q = Q′. □

Proof of Theorem 3. We are now ready to prove Theorem 3.

(1) VDet(CQ,PC). We first consider the case when V consists of a single view V. We then show the result for general views.

Single PC view: Let Q(XQ) be a minimal CQ query and V(XV) be a PC view defined over relational schema R = (R₁, . . . , Rₖ). Since V is a PC query, [V] contains no constants and each variable in [V] appears only once. This implies that for any pair of tuples TₓV ∈ [V] and T_Q ∈ [Q] over the same relation in R, there is a unique homomorphism h from TₓV to T_Q.

We show that V determines Q iff the following conditions are satisfied: (1) the relation symbols appearing in V are exactly the same as those appearing in V; (2) for each tuple T_Q ∈ [Q] there exists a tuple TₓV ∈ [V] over the same relation in R such that for each variable x in TₓV and for the homomorphism h from TₓV to T_Q, if (a) h(x) is a constant; or (b) h(x) appears more than once in [Q]; or (c) h(x) appears in T_Q, then x must appear in T_Q. These conditions can be easily checked in PTIME.

We first show that if these conditions hold then V → Q. More specifically, we show that the conditions imply that T_Q ∈ Q(V⁻¹(S)). Let S = V(Q()). By condition (1), S ≠ ∅, and one can construct the instance V⁻¹(S) from S. Consider an arbitrary tuple T_Q ∈ [Q] and let TₓV be a tuple [V] that satisfies condition (2). Let h be the homomorphism from TₓV to T_Q. Since V is a PC query, we can extend h to be a homomorphism h̃ from V to [Q]. Moreover, by the construction of V⁻¹(S), there is a tuple T ∈ V⁻¹(S) such that T = h̃(TₓV), where h̃(x) = h(x) if x appears in TₓV, and h̃(x) is a new distinct variable otherwise. Now consider tuples T_Q and T. By conditions (2a) and (2b), T_Q and T are isomorphic, and the homomorphism h̃ from T_Q to T is identity on variables that appear more than once in [Q]. By gathering all homomorphism h̃ between tuples T_Q ∈ [Q] and tuples T ∈ V⁻¹(S), constructed as above from tuples TₓV that satisfy condition (2), we thus obtain a homomorphism h̃ from [Q] to V⁻¹(S). By condition (2c), h̃ is identity on variables that appear in X_Q, and thus h̃(x_Q) = x_Q and x_Q ∈ Q(V⁻¹(S)). From Proposition 4(i) it follows that V → Q.

We next show that the conditions are also necessary. We first consider condition (1). Suppose that Q has less relation symbols than V. In this case, V(Q()) = ∅ and by Proposition 3(i), V does not determine Q. If Q has more relation symbols than Q, then by Proposition 3(ii), V cannot determine Q either. In other words, condition (1) needs to be satisfied.

Next, consider condition (2). Suppose that there exists a tuple T_Q ∈ [Q] such that for any tuple TₓV ∈ [V] over the same relation as T_Q, one of the conditions (2a)–(2c) is not satisfied. Let T = T_Q. For each tuple TₓV ∈ [V] over the same relation as T, we construct a tuple T from T as follows. Let h be the unique homomorphism from T_Q to T. Let x be the variable in TₓV that does not occur in X_V but either (a) h(x) is a constant; (b) h(x) occurs multiple times in [Q]; or (c) h(x) appears in X_Q. We then construct T from T by replacing each h(x) in T with a new distinct variable. For each TₓV we put the resulting tuple S in the set Δ. Lemma 3 implies that Δ is critical for T and Q, and Lemma 2 tells us that Q(D₁) ≠ Q(D₂) for D₁ = [Q] ∪ A and D₂ = ([Q] ∪ A). Hence if V(D₁) ≠ V(D₂) then V does not determine Q.

We now verify that V(D₁) = V(D₂). Since D₂ ⊆ D₁, we have that V(D₁) ⊆ V(D₂). Hence, we need to show that V(D₁) ⊆ V(D₂). Let T ∈ V(D₁) and let h′ : [V] → D₁ such that h′(T) = T. Then for the tuple TₓV ∈ [V] such that h′(TₓV) = T, we have a tuple T ∈ A that coincides with T on variables in TₓV that occur in X_V but may be different on some other attributes. Since V is a PC query, however, we can define h′ : [V] → D₂ such that h′ = h on [V] \ TₓV and h′(TₓV) = T. Clearly, h′(TₓV) = T, and since this argument works for every tuple in V(D₁), we have that V(D₁) ⊆ V(D₂). Hence V(D₁) = V(D₂).

Multiple PC views: We next consider VDet(CQ,PC) when V consists of multiple views (V₁(X₁), . . . , Vₖ(Xₖ)). We show that VDet(CQ,PC) is in PTIME by reducing the multiple view case to the single-view case.

The reduction is given as follows. First, we divide V into two sets: V₁ is the set of views Vᵢ ∈ V such that Vᵢ contains more relation symbols than Q, and V₂ = V \ V₁. Next, we consider the product query V₀ of the views in V₂. Here we assume that the variables in the Vᵢ's are all distinct and consider V₀ = ∏ᵢ Vᵢ in which the free variables in V₀ is the union of the free variables in the Vᵢ's. We show that V → Q iff V₂ = ∅ and V₀ → Q. Note that both conditions can be checked in PTIME.

Suppose first that the conditions hold. Consider two instances D₁ and D₂ such that V(D₁) = V(D₂). This implies that V₀(D₁) = V₀(D₂). Because V₀ → Q, we can then conclude that Q(D₁) = Q(D₂). In other words, V → Q.

Conversely, suppose that one of the conditions does not hold. Clearly, when all views Vᵢ in V address more relation symbols than Q, then Vᵢ(Q()) = ∅ for i ∈ [1, k] and hence V(Q()) = ∅. Proposition 3(i) then tells us that V cannot determine Q. In other words, V₂ must be nonempty. Suppose next that V₀ does not determine Q. Since V₀ is a single view, this implies that the conditions for the single view case (as stated in the proof above) do not hold. As a consequence, we can construct the two instances D₁ and D₂, as in the proof for single PC views, which have the property that V₀(D₁) = V₀(D₂) but Q(D₁) ≠ Q(D₂). Furthermore, observe that V₀(D₁) = V₀(D₂) ≠ ∅ and therefore V(D₁) = V(D₂) for any D ∈ V₂. Since D₁ and D₂ are constructed from [Q], these instances are empty for all relations not addressed by Q. As a result V(D₁) = V(D₂) = ∅ for any D ∈ V₂. Putting these together, V(D₁) = V(D₂) but Q(D₁) ≠ Q(D₂). Hence V does not determine Q.

(2) VDet(CQ,SP). We first consider the case when V consists of a single view V and then extend the result to general views.

Single SP view: Let Q(X_Q) be a minimal CQ query and V(X_V) be an SP view defined over relational schema R = (R₁, . . . , Rₖ). Since V is an SP query, [V] consists of a single tuple TₓV over some relation R in R.

We provide necessary and sufficient conditions on V and Q to decide whether V determines Q or not. More specifically, we show that V → Q iff (1) Q only contains the
relation symbol $R$; (2) for each tuple $T_0 \in \{Q\}$, there exists a homomorphism $h$ from $T_V$ to $T_0$ and furthermore, for each variable $x$ in $T_0$, if (a) $h(x)$ is a constant; or (b) there exists a variable $y$ in $T_V$ such that $x \neq y$ but $h(x) = h(y)$; or (c) $h(x)$ appears in multiple tuples in $\{Q\}$; or finally, (d) if $h(x)$ appears in $x_Q$, then $x$ must appear in $x_V$. These conditions can be easily checked in PTIME.

We first show that if the conditions above hold then $V \rightarrow Q$, by showing that the conditions imply that $x_Q \in Q(V^{-1}(S))$. Let us consider $S = V(Q)$ in more detail. Suppose that $Q$ consists of $m$ tuples $T_1, \ldots, T_m$. Since $V$ contains only one tuple $T_V$, one can easily verify that $S$ is the projection of $(h_i(T_V))_{i \in [1,m]}$ on the attributes corresponding to $x_Q$, where $h_i$ is the homomorphism from $T_V$ to $T_i$ for each $T_i \in \{Q\}$. Consequently, we also have an explicit description of $V^{-1}(S)$. Indeed, $V^{-1}(S) = \{h_i(T_V) | i \in [1,m]\}$, where $h_i(x) = h_i(x) = x$ if $x$ is a variable in $x_Q$, and $h_i(x) = x'$ otherwise, and $x'$ is a distinct new variable not appearing anywhere else. We next show that the conditions imply that $x_Q \in Q(V^{-1}(S))$ and hence by Proposition 4(i), that $V \rightarrow Q$.

We show that $x_Q \in Q(V^{-1}(S))$ by constructing a homomorphism $H: Q \rightarrow V^{-1}(S)$ such that $H(x_Q) = x_Q$. Let $\overline{e} \in \{Q\}$ and consider the tuple $T_\overline{e} = h_i(T_V) \in V^{-1}(S)$. Let $u$ be a variable or constant in $\overline{e}$ and let $h_i^u : T_\overline{e} \rightarrow S$ be as follows: $h_i^u(u) = u$ in case that $h_i^u(u) \subseteq x_Q$, and $h_i^u(u) = h_i(h_i^u(u))$ in case that $h_i^u(u) \subseteq x_Q \Leftrightarrow \emptyset$. By the construction of $V^{-1}(S)$, the mapping $h_i^u$ is an isomorphism from $T_\overline{e}$ to $S$. Furthermore, condition (2c) guarantees that the union of all $h_i^u$, for $i \in [1,m]$ is a homomorphism $H$ from $\{Q\}$ to $V^{-1}(S)$ which is, by condition (2d), ensured to be identity on variables in $x_Q$. In other words, $H(x_Q) = x_Q$ and therefore, $x_Q \in Q(V^{-1}(S))$.

We next show that the conditions above are also necessary. The necessity of condition (1) follows immediately from Proposition 3(iii). We next consider condition (2) and show that if this condition does not hold, then $V$ does not determine $Q$.

First suppose that there exists a tuple $T \in \{Q\}$ such that there exists no homomorphism from $T_V$ to $T$. In this case, the instances $D_1 = \{Q\} \setminus \{\overline{T}\}$ and $D_2 = \{Q\}$ provide a counter-example for view determinacy. Indeed, the lack of homomorphism implies that $V(I(Q)) = V(D_1)$. The minimality of $Q$, however, implies that $Q(I(Q)) \neq Q(D_2)$. Hence, we may assume that for any tuple $T \in \{Q\}$ there exists a homomorphism $h$ from $T_V$ to $T$.

Suppose, however, that there exist a tuple $T_0 \in \{Q\}$ and a variable $x$ in $T_0$ such that $x$ does not occur in $x_Q$, but for the homomorphism $h$ from $T_V$ to $T_0$, either $h(x)$ is a constant, or $h(x)$ appears in multiple tuples in $\{Q\}$, or $h(x)$ appears in $x_Q$; or there exists another variable $y$ with $h(x) = h(y)$. Let $T = T_0$. We construct a tuple $\overline{T}$ from $T$ by replacing each occurrence of $h(x)$ in $T$ that corresponds to each occurrence of $x$ in $T_V$ with a new distinct variable. Note that the replacement does not affect the existence of a homomorphism from $T_V$ to $\overline{T}$. Since $x$ does not appear in $x_Q$, we have that $V(I(T)) = V(\overline{T})$. Let $D_1 = \{Q\} \setminus \{\overline{T}\}$ and let $D_2 = \{Q\} \setminus \{\overline{T}\}$. Since $V$ is an SP query we may conclude that $V(D_1) = V(D_2)$. From Lemma 3 we know that $D = \{\overline{T}\}$ is critical for $\overline{T}$ and $\{Q\}$, and hence Lemma 2 implies that $Q(D_1) \neq Q(D_2)$. In other words, $V$ does not determine $Q$.

**Multiple SP views:** We next consider the case when $V$ consists of a number of SP views. Let $Q$ be a CQ query and $V$ be a set of SP views. For each view $V \in V$, $|V|$ consists of one tuple $T_V$ over some relation in $R$. We show that $V \rightarrow Q$ iff for each tuple $T_0 \in \{Q\}$, there exist a tuple $T_V$ in $|V|$ over the same relation as $T_0$ and a homomorphism $h$ from $T_V$ to $T_0$ such that the conditions (2a)–(2d) described above are satisfied for $t_0, t_V$ and $h$. As before, these conditions can be checked in PTIME.

Along the same lines as in the proof for single SP views, one can readily verify that the conditions are sufficient to determine whether $V \rightarrow Q$. We next show their necessity.

Suppose that there exists a tuple $T_0 \in \{Q\}$ for which no tuple $T_V \in |V|$ can be found that can be mapped onto $T_0$. In this case, deleting $T_0$ from $\{Q\}$ results in $V(I(Q)) = V(\{Q\} \setminus \{T_0\})$. The minimality of $Q$, however, implies that $Q(I(Q)) \neq Q(\{Q\} \setminus \{T_0\})$. Hence, $V$ does not determine $Q$.

Next, suppose that there exists a tuple $T_0 \in \{Q\}$ such that, for each tuple $T_V \in |V|$ for which there exists a homomorphism to $T_0$, but one of the conditions (2a)–(2d) do not hold. Let $\overline{T} = T_0$. For each such tuple $T_V$, we construct a row $\overline{s}$ from $\overline{T}$, similar to the construction in the proof for single SP views. Let $\overline{A}$ be the set of all the constructed tuples. Let $D_1 = \{Q\} \setminus \{\overline{T}\}$ and $D_2 = \{Q\} \setminus \{\overline{T}\}$. It is readily verified that $V(D_1) = V(D_2)$, $\overline{A}$ is critical for $\overline{T}$ and $\{Q\}$ and hence, $Q(D_1) \neq Q(D_2)$. In other words, $V$ does not determine $Q$.

(3) **VDet(QC,SC):** We show that VDet(QC,SC) is NP-hard by reduction from the graph 3-colorability problem, which is known to be np-complete (cf. [21]). The NP upper bound holds even when queries in $\mathcal{L}$ are not minimal, which will be verified in the proof of Theorem 4(2).

The reduction is constructed as follows. Given a graph $G = (V,E)$, we define a (minimal) CQ query $Q$ and an SC view $W$ such that $W \rightarrow Q$ iff $G$ is 3-colorable. More specifically, let $C$ be a set of three variables disjoint from the set of vertices $V$, and $R$ be a binary relation. We construct a CQ query $Q(x)$ such that

$$Q = \{(c_1, c_2) | c_1, c_2 \in C, c_1 \neq c_2\}$$

and an SC view $W(x,y)$ such that

$$W = \{(u_1, u_2) | (u_1, u_2) \in E \cup \{(c_1, c_2) | c_1, c_2 \in C, c_1 \neq c_2\}\}$$

The free variables in $Q$ are given by $x = (c_1, c_2, c_1, c_3, c_2, c_1, c_3, c_2, c_2, c_1, c_3, c_2)$. The free variables of $W$ are $(x,y)$, where $x$ is as in $Q$ and $y$ consists of all edges in $E$.

We show that $W$ determines $Q$ iff $G$ is 3-colorable. Suppose that $G$ is 3-colorable and let $\gamma: V \rightarrow C$ be a 3 coloring of $V$. Consider the view $W' = (|W|, x)$. Then, $h: |W| \rightarrow |Q|$ defined as $h(v_i) = \gamma(v_i)$ and $h(c_i) = c_i$ is a homomorphism from $|W|$ to $|Q|$ such that $h(\overline{x}) = x$. Indeed, since $\gamma$ is a 3-coloring of $V$, we have that $h((w,v)) = (\gamma(v), \gamma(w)) \in (Q)$. Hence, $Q \subseteq W'$. Since $W' \subseteq Q$ we then have that $Q = \pi_W(W)$. This in turn implies that $W \rightarrow Q$. On the other hand, suppose that $G$ is not 3-colorable. Then there exists no homomorphism from $|W|$ to $|Q|$ and thus $W(I(Q)) = \emptyset$. From Proposition 3(i) we can conclude that $W$ does not determine $Q$. □
The proof of Theorem 3 also tells us that CQ is complete for rewriting CQ queries using SC, SP or PC views.

**Corollary 1.** The class of conjunctive queries is complete for ℒ-to-CQ rewritings when ℒ is SC, SP, or PC. □

**Proof.** The cases when ℒₜ is SP or PC follow immediately from the proofs of Theorems 3(1) and (2), respectively. Indeed, in those proofs it has been shown that if V → Q then Q⁻¹ is given by a CQ query. Observe that the statement remains intact for selection queries in ℒₜ that are not necessarily minimal. Indeed, for a CQ query Q, V → Q iff V → Qₚₚₑₚ, where Qₚₚₑₚ is a minimal CQ query equivalent to Q. Moreover, if there exists a query Q⁻¹ such that Q⁻¹ is a CQ rewriting of Qₚₚₑₚ using V, then Q⁻¹ is also a CQ rewriting of Q using V, and vice versa.

The case when ℒₜ is SC follows from Theorem 1 in [2] in which the completeness of CQ is shown for views that do not contain non-distinguishable variables. □

**Corollary 1 and Proposition 1,** when taken together, tell us that Qₚₑₚ(CQ, ℒₜ₂, ℒₜ₃) is equivalent to VDet(CQ, ℒₜ₃) when the language ℒₜ₂ subsumes CQ. Hence, from Theorem 3 we obtain:

**Corollary 2.** When queries in ℒₜ are minimal CQ queries, Qₚₑₚ(CQ, ℒₜ₂, ℒₜ₃) is

(1) in PTIME when ℒₜ is PC or SP, and

(2) NP-complete when ℒₜ is SC. □

5.3. Arbitrary CQ queries

We next turn our attention to the general case, that is, when the queries in ℒₜ are not necessarily minimal. We first consider the invertibility problem. While general CQ queries do not make our lives harder when views are PC queries, they do complicate the invertibility analysis for SP views. Indeed, the invertibility problem becomes intractable for SP views, in contrast to PTIME when queries in ℒₜ are minimal (Theorem 3(2)).

**Theorem 4.** VDet(CQ, ℒₜ₃) is

(1) PTIME when ℒₜ is PC, and

(2) NP-complete when ℒₜ is SP or SC. □

**Proof.** We first show that VDet is in PTIME for PC views. We then show the tractability of the problem for SP and SC views.

(1) VDet(CQ,PC). We need the following notation. Given a CQ query Q(Ђₜₓₜₒ) and a variable x in Q, we call x typed if x appears only in one column over a single relation table in Q. For a typed variable x, let Qₓ be the set of tuples in Q that contain x. If Qₓ can be mapped to a single tuple via a homomorphism, we call x single typed. Checking whether a variable is typed or single typed can be done in PTIME.

We first consider a single PC view V(Ђₜₓₜₒ). We show that V → Q iff the following conditions are satisfied: (′1′) the relation symbols appearing in Q are exactly the same as those appearing in V; and (′2′) for each row fₒ ∈ Q, there exists a row fₓ ∈ V over the same relation such that the following conditions are satisfied for each variable x in fₓ and for the (unique) homomorphism h from fₓ to fₒ: x must appear in fₓ(a) if h(x) is a constant; or (b) if h(x) is not typed; or (c) h(x) is typed but not single typed; or (d) if h(x) appears in fₒ.

We show that these conditions on Q and V translate to the conditions described in the proof of Theorem 3(1) when considering a minimal query Q equivalent to Q and the same view V. From this, the PTIME result follows. More specifically, let Q(Vₓₒ) be the minimal CQ query equivalent to Q such that [Q] ⊆ [V]. Clearly, Q′ and Q access the same set of relation symbols and thus condition (′1′) is equivalent to condition (1) in the proof of Theorem 3(1).

We next verify this for condition (′2′). Since Q′ ⊆ Q, there exists a homomorphism h : [Q] → [Q′] with h(Vₓₒ) = Vₓₒ. Let fₒ ∈ Q and fₓ ∈ V be two tuples over the same relation. Let fₒ = h(fₒ). Consider the homomorphisms h₁ : fₒ → fₒ and h₂ : fₓ → fₒ. It is clear that h₁ = h₁hₓ. Let h be a variable in fₓ. We distinguish between the following cases, depending on the conditions stated above: (′2′a) If h(x) is a constant, then h₂(x) is the same constant; (′2′b) If h(x) is not typed or (′c) if h(x) is typed but not single typed, then h₂(x) = h₁hₓ(x) appears multiple times in [Q′]; (′2′d) If h(x) occurs in fₒ, then h₂(x) = h₁hₓ(x) also occurs in fₒ. Therefore the conditions (′2′a)–′(2′d) for general CQ queries correspond to the conditions (′2a)–′(2c) for minimal CQ queries in the proof of Theorem 3(1).

When V consists of multiple PC views, we can verify the statement by reduction to the single-view case, along the same lines as the proof for the multiple view case for minimal CQ queries. We omit the details here to avoid repetition.

(2) VDet(CQ,SP) and VDet(CQ,SC): From Theorem 3(3) we already know that VDet(CQ,SC) is NP-hard when queries in ℒₜ are minimal. This lower bound trivially carries over to the general case. We therefore only need to show that VDet(CQ,SP) is NP-hard for general queries and establish a matching NP upper bound for VDet(CQ,SP) and VDet(CQ,SC).

For the lower bound, we show a stronger result. That is, we show that VDet(CQ,Σ) is already NP-hard by reduction from the containment problem for CQ queries, which is known to be NP-complete [cf. [8]]. Let Q₁(Ђxo) and Q₂(Ђxo) be two CQ queries over the same n-ary relation R. We construct a CQ query Q and an S view V = [V] such that V → Q iff Q₁ ⊆ Q₂.

More specifically, let R be an (n + 1)-ary relation obtained from R by adding an extra attribute A. We define Q(Ђxo) as a CQ query over R′ such that [Q] = (Ђxo) × [Q₂] where u is a variable not appearing anywhere else and Χ denotes the free variables in Q₁ and Q₂, respectively. Let V(Ђxo) be the S-query over R′ with

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Undecidability results. Bold entries are new results shown in this paper.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>Complexity</td>
</tr>
<tr>
<td>VDet(F0, CQ)</td>
<td>Undecidable [5, Corollary 2.2]</td>
</tr>
<tr>
<td>VDet(F0, FO)</td>
<td>Undecidable [5, Corollary 2.2]</td>
</tr>
<tr>
<td>VDet(DATALOG, CQ)</td>
<td>Undecidable (Theorem 1(1))</td>
</tr>
<tr>
<td>VDet(UCQ, UCQ)</td>
<td>Undecidable (Theorem 1(2))</td>
</tr>
<tr>
<td>VDet(DATALOG, UCQ)</td>
<td>Undecidable [5, Theorem 4.1]</td>
</tr>
</tbody>
</table>
We summarize the main complexity results for invertibility (view determinacy) in Tables 1 and 2, annotated with their corresponding theorems. All the results in Table 2 and the highlighted results in Table 1 have not appeared in the literature.

The study of selected information preservation is still preliminary. One open problem is to establish the complexity of VDet(CQ,CQ), and QPre(CQ,CQ,CQ). However, these are by no means trivial: for example, VDet(CQ,CQ) is equivalent to the view determinacy problem for CQ queries and CQ views, whose decidability remains unknown [5]. Another issue is to study VDet(L,CQ) and QPre(L,CQ,CQ), for CQ views and selection queries Q in L ranging over SP, PC, and SC. A third topic is to identify practical cases of VDet(CQ,CQ) and QPre(CQ,CQ,CQ) that are tractable. In particular, our conjecture is that the analysis would become simpler for key preserving CQ views, i.e., views that retain the keys of base relations involved [22]. Data transformations in practice are either key preserving or can be naturally extended to preserve keys.

Acknowledgments

Fan and Geerts are supported in part by an IBM scalable data analytics for a smarter planet innovation award, and the RSE-NSFC Joint Project Scheme. Fan is also supported in part by the National Basic Research Program of China (973 Program) 2012CB316200 and NSFC 61133002. Zheng is supported in part by NSFC 61070038.

References


G. Cong, W. Fan, F. Geerts, J. Li, J. Luo, On the complexity of view update analysis and its application to annotation propagation, Transactions on Knowledge and Data Engineering. (http://doi.ieee.computersociety.org/10.1109/TKDE.2011.27), in press.