Complexity of Answering Queries Using Materialized Views

Serge Abiteboul, Olivier Duschka

Complexity of Answering Queries Using Materialized Views

Serge Abiteboul
INRIA Rocquencourt
78153 Le Chesnay Cedex, France

Oliver M. Duschka
Stanford University
Stanford, CA 94305

Abstract

We study the complexity of the problem of answering queries using materialized views. This problem has attracted a lot of attention recently because of its relevance in data integration. Previous work considered only conjunctive view definitions. We examine the consequences of allowing more expressive view definition languages. The languages we consider for view definitions and user queries are: conjunctive queries with inequality, positive queries, datalog, and first-order logic. We show that the complexity of the problem depends on whether views are assumed to store all the tuples that satisfy the view definition, or only a subset of it. Finally, we apply the results to the view consistency and view self-maintainability problems which arise in data warehousing.

1 Introduction

The notion of materialized view is essential in databases [34] and is attracting more and more attention with the popularity of data warehouses [24]. The problem of answering queries using materialized views [25, 7, 12, 5, 43, 30, 27, 36, 15, 11, 13, 26] has been studied intensively. We propose a systematic study of its complexity. We also briefly consider the related problems of view consistency and view self-maintainability [19]. Our results exhibit strong connections with two among the most studied problems in database theory, namely query containment [6, 31, 23, 32, 8, 21, 10, 28] and incomplete information querying, e.g. [20, 2]. Indeed, the works most closely related to our complexity results are perhaps those of van der Meyden [37, 38, 39] and Vardi [41] on (indefinite) database queries. Our results highlight the basic roles played by negation (and in its weak form inequality) and recursion, and a crucial difference between *open* and *closed world assumption* in the view definition.

The main focus of the paper is the study of the *data complexity* of the problem of *answering queries using materialized views*. More precisely, the problem is for a fixed view definition and a fixed query, given a view instance I and a tuple t, is t a *certain* answer, i.e. is t in the answer to the query on the database no matter which is the database yielding the view instance I. This articulation of the problem highlights the main parameters: (i) What are the database and the view models? (ii) What are the query and the view definition languages? (iii) Is *yielding* assuming an open or a closed world?

In the present paper, we use the relational model for the database and the view model. However, our work strongly suggests moving towards an incomplete information model, e.g. conditional tables [20]. Indeed, we will briefly show how these tables can be used for solving the problem in most solvable cases. For the query and view definition languages, we consider the most popular formal query languages, namely conjunctive queries, conjunctive queries with inequality, positive queries, datalog, and first-order logic. We focus on *certain* answers, i.e. tuples that are in the answer for *any* database yielding this particular view instance.

Not surprisingly, our results indicate that recursion and negation in the view definition lead to undecidability. Somewhat also expectedly, we show that the closed world assumption sharply complicates the problem. For instance, under the open world assumption the certain answers in the conjunctive view definitions/datalog queries case can be computed in polynomial time. On the other hand, already the conjunctive view definitions/conjunctive queries case is co-NP-complete under the closed world assumption. This is an a-posteriori argument for a number of recent works that postulate an open world interpretation of views. Perhaps more unexpectedly, we prove that inequalities (a very weak form of negation) lead to intractability. Even under the open world assumption, adding inequalities to the queries, or disjunction to the view definitions makes the problem co-NP-hard.

2 The problem

In this section, we present the problem. We assume some familiarity with database theory [34, 1]. We start with a database instance D, a view definition $\mathcal{V}$, and a view instance I.

The database consists of a set of relations and so does the view. Now, given a query $\mathcal{Q}$, we would like to compute $\mathcal{Q}(D)$. However, we assume that we ignore D and only have access to I, so we will try to get the best possible estimate of $\mathcal{Q}(D)$ given I.

Let us be more precise. Under the *closed world assumption* (CWA), the view instance I stores *all* the tuples that satisfy the view definitions in $\mathcal{V}$, i.e. $I = \mathcal{V}(D)$. Under the *open world assumption* (OWA), on the other hand, instance I is possibly incomplete and might only store *some* of the tuples that satisfy the view definitions in $\mathcal{V}$, i.e. $I \subseteq \mathcal{V}(D)$. As we can see from the following example, in reasoning about the underlying database, it makes a difference whether we are working under the open or closed world assumption.

Example 2.1 Consider the following view definition where the view consists of two relations:

$$v_1(X) :- p(X, Y)$$
$$v_2(Y) :- p(X, Y)$$

and assume that the view instance consists of $\{v_1(a),\ v_2(b)\}$. Under OWA, we only know that some p tuple has value a as its first component, and some (possibly different) p tuple has value b as its second component. Under CWA, however, we can conclude that all p tuples have value a as their first component and value b as their second component, i.e. p contains exactly the tuple $\langle a, b\rangle$. □

Given some view definition and a view instance, observe that there may be a number of possible databases, i.e. database instances that yield this view instance for this view definition. So, we can think of the database as the *incomplete database* [20] consisting of this set of possible databases. To answer a query, we focus on *certain* answers, i.e. on tuples that are in the answer for each possible database. As seen in Example 2.1, this depends on whether we are assuming an open or a closed world. Indeed, an answer that is certain under OWA is also certain under CWA, but the converse does not hold in general. For instance, in the previous example, the query "is $\langle a, b\rangle$ certainly in p?" is answered positively under CWA and negatively under OWA. In fact, we will show that computing certain answers under CWA is harder than under OWA. The following definition formalizes the concept of certain answer under both assumptions:

Definition 2.1 (**certain answer**) Let $\mathcal{V}$ be a view definition, I be an instance of the view, and $\mathcal{Q}$ a query. A tuple t is a *certain answer under OWA* if t is an element of $\mathcal{Q}(D)$ for each database D with $I \subseteq \mathcal{V}(D)$. A tuple t is a *certain answer under CWA* if t is an element of $\mathcal{Q}(D)$ for each database D with $I = \mathcal{V}(D)$. □

We briefly recall the query languages we consider and the standard notion of complexity we use.

2.1 Query and view languages

A *datalog rule* is an expression of the form:

$$p(\bar{X}) :- p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)$$

where p, and $p_1, \ldots, p_n$ are predicate names, and $\bar{X}$, $\bar{X}_1$, ..., $\bar{X}_n$ are tuples of variables and constants. Each variable in the head of a rule must also occur in the body of the rule. A *datalog* query is a finite set of datalog rules. The notion of *recursive* datalog query/rule is defined in the standard way. A *conjunctive query* (CQ) is a single non-recursive datalog rule. If the body of a conjunctive query is allowed to contain the inequality predicate ($\neq$), then the query is called a *conjunctive query with inequality* ($CQ^{\neq}$). Every variable in a query with inequality must occur at least once in a relational predicate. A *positive query* (PQ) is a non-recursive datalog query together with one particular predicate defined by the query. The query language $PQ^{\neq}$ is obtained by also allowing $\neq$. Finally, *first-order queries* (FO) are defined in the standard way.

A *materialized view*, also called *view instance*, is the stored result of previously executed queries. A *view definition* $\mathcal{V}$ therefore consists of a set of queries defining a finite set of predicates. So, for a query language $\mathcal{L}$, we write $\mathcal{V} \subseteq \mathcal{L}$ to denote the fact that each predicate in the view is defined using a query in $\mathcal{L}$.

2.2 Data complexity

We will be interested in the *data complexity* of the problem of computing certain answers under the open and closed world assumption. The *data complexity* is the complexity of the problem as a function of the size of the view instance. We will also refer to the *query* and *combined complexity* of the problem. The *query complexity* is the complexity of the problem as a function of the size of the view definition $\mathcal{V}$ and the query $\mathcal{Q}$. The *combined complexity* is the complexity of the problem as a function of these two arguments plus the size of the view instance. (These three notions are due to [40].) In the remaining of the paper, when we discuss complexity, we always mean data complexity unless specified otherwise.

In Section 5, we prove that the problem is in co-NP for a wide range of cases. We also highlight some connections with conditional table querying. In Section 3, we examine the complexity of the problem of computing certain answers under OWA and in Section 4 under CWA. In Section 6, we consider view self-maintainability and view consistency.

2.3 Upper bounds

In this section, we briefly sketch a solution to the problem for the open and closed world assumption, when the view definition is in $PQ^{\neq}$ and the query is in $datalog^{\neq}$.

First we see next that, for $PQ^{\neq}$ views and $datalog^{\neq}$ queries, the problem is in co-NP. So within these limits, it will suffice in the following of the paper to prove co-NP-hardness to establish co-NP-completeness.

Theorem 2.1 *For $\mathcal{V} \subseteq PQ^{\neq}$, $\mathcal{Q} \in datalog^{\neq}$, the problem of determining, given a view instance, whether a tuple is a certain answer under OWA or CWA, is in co-NP.*

Proof. We prove the claim first for OWA. Assume that t is not a certain answer. Then there is a database D with $I \subseteq \mathcal{V}(D)$ and t is not in $\mathcal{Q}(D)$. Let n be the total number of tuples in I and let k be the maximal length of conjuncts in the view definitions. Each tuple in I can be generated by at most k tuples in D. Therefore, there is a database $D' \subseteq D$ with

	— query —				
views	CQ	$CQ^{\neq}$	PQ	*datalog*	FO
CQ	PTIME	co-NP	PTIME	PTIME	undec.
$CQ^{\neq}$	PTIME	co-NP	PTIME	PTIME	undec.
PQ	co-NP	co-NP	co-NP	co-NP	undec.
datalog	co-NP	undec.	co-NP	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Figure 1: *Data complexity of the problem of computing certain answers under the open world assumption.*

at most nk tuples such that still $I \subseteq \mathcal{V}(D')$. Because t is not in $\mathcal{Q}(D)$ and $\mathcal{Q}$ is monotone, t is also not in $\mathcal{Q}(D')$. It follows that there is a database D' whose size is polynomially bounded in the size of I and $\mathcal{V}$ such that $I \subseteq \mathcal{V}(D')$, and t is not in $\mathcal{Q}(D')$. Moreover, checking that $I \subseteq \mathcal{V}(D')$ and that t is not in $\mathcal{Q}(D')$ can be done in polynomial time.

For CWA, the proof is essentially the same with $I = \mathcal{V}(D)$ in place of $I \subseteq \mathcal{V}(D)$. □

The proof of Theorem 2.1 gives a construction of how to compute certain answers under OWA or CWA in co-NP time. However, this construction is not very useful in practice because it requires the enumeration of a large number of possible databases. In Section 5 we will describe a more practical way to compute the certain answers.

3 Open world assumption

Figure 1 gives an overview of the complexity of computing certain answers under OWA. Under OWA, the problem of computing certain answers is closely related to the query containment problem. Therefore, decidability and undecidability results carry over in both directions. As shown in Theorem 3.1, if the problems are decidable, then their *query complexity* is the same.

Theorem 3.1 *Let $\mathcal{L}_1, \mathcal{L}_2 \in \{CQ, CQ^{\neq}, PQ, datalog, FO\}$ be a view definition language and query language respectively. Then the problem of computing certain answers under OWA of a query $\mathcal{Q} \in \mathcal{L}_2$ given a view definition $\mathcal{V} \subseteq \mathcal{L}_1$ and a view instance is decidable if and only if the containment problem of a query in $\mathcal{L}_1$ in a query in $\mathcal{L}_2$ is decidable. Moreover, if the problems are decidable then the combined complexity of the view problem and the query complexity of the containment problem are identical, so the data complexity of the problem of computing certain answers under OWA is at most the query complexity of the query containment problem.*

Proof. The claim is established by giving reductions between the two problems in both directions. We first consider the reduction from the problem of computing certain answers under OWA to the query containment problem. Let $\mathcal{V} = \{v_1, \ldots, v_k\} \subseteq \mathcal{L}_1$ be a view

definition, $\mathcal{Q} \in \mathcal{L}_2$ a query, I a view instance, and t a tuple of the same arity as the head of $\mathcal{Q}$. Let $\mathcal{Q}'$ be a query consisting of the rules of definition $\mathcal{V}$ together with the rule[1]

$$q'(t) :- v_1(t_{11}),\ \ldots,\ v_1(t_{1n_1}), \ldots, v_k(t_{k1}),\ \ldots,\ v_k(t_{kn_k})$$

where I is the instance with $I(v_1) = \{t_{11}, \ldots, t_{1n_1}\}$, $\ldots$, $I(v_k) = \{t_{k1}, \ldots, t_{kn_k}\}$. If $\mathcal{L}_1$ is CQ or $CQ^{\neq}$, then the view definitions in $\mathcal{V}$ can be substituted in for the view literals in this new rule. This yields just one conjunctive query. In all cases, $\mathcal{Q}'$ is in $\mathcal{L}_1$. We show that tuple t is a certain answer of $\mathcal{Q}$ given $\mathcal{V}$ and I if and only if $\mathcal{Q}'$ is contained in $\mathcal{Q}$.

"$\Rightarrow$": Assume that t is a certain answer under OWA. Let D be a database. If $I \not\subseteq \mathcal{V}(D)$, then $\mathcal{Q}'(D) = \{\}$, and therefore $\mathcal{Q}'(D)$ is trivially contained in $\mathcal{Q}(D)$. If $I \subseteq \mathcal{V}(D)$, then $\mathcal{Q}'(D) = \{t\}$ and $t \in \mathcal{Q}(D)$. Again, $\mathcal{Q}'(D)$ is contained in $\mathcal{Q}(D)$.

"$\Leftarrow$": Assume that $\mathcal{Q}'$ is contained in $\mathcal{Q}$. Let D be a database with $I \subseteq \mathcal{V}(D)$. Then $\mathcal{Q}'(D) = \{t\}$, and therefore $t \in \mathcal{Q}(D)$. Hence, t is a certain answer.

The remaining of the proof consists of a reduction from the query containment problem to the problem of computing certain answers under OWA. Let $\mathcal{Q}_1 \in \mathcal{L}_1$ and $\mathcal{Q}_2 \in \mathcal{L}_2$ be two queries. Let p be a new predicate, and let q_1 and q_2 be the answer predicates of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ respectively. Consider as view definition the rules of $\mathcal{Q}_1$ together with the additional rule

$$v(c) :- q_1(X),\ p(X)$$

and the instance $I = \{v(c)\}$. Let the query $\mathcal{Q}$ be defined by all the rules of $\mathcal{Q}_2$ together with the following rule:

$$q(c) :- q_2(X),\ p(X).$$

Again, if $\mathcal{L}_1$ or $\mathcal{L}_2$ are CQ or $CQ^{\neq}$, then the definition of $\mathcal{V}$ and query $\mathcal{Q}$ respectively can be transformed into a conjunctive query. Therefore, $\mathcal{V} \subseteq \mathcal{L}_1$ and $\mathcal{Q} \in \mathcal{L}_2$. We show that $\mathcal{Q}_1$ is contained in $\mathcal{Q}_2$ if and only if $\langle c \rangle$ is a certain answer of $\mathcal{Q}$ given $\mathcal{V}$ and I.

"$\Rightarrow$": Suppose that $\langle c \rangle$ is not a certain answer. Then there exists a database D with $I \subseteq \mathcal{V}(D)$ and $\mathcal{Q}(D)$ does not contain $\langle c \rangle$. It follows that $\mathcal{Q}_1(D)$ contains a tuple that $\mathcal{Q}_2(D)$ does not contain. Therefore, $\mathcal{Q}_1$ is not contained in $\mathcal{Q}_2$.

"$\Leftarrow$": Assume that $\mathcal{Q}_1$ is not contained in $\mathcal{Q}_2$. Then there exists a database D such that $\mathcal{Q}_1(D)$ contains a tuple t that is not contained in $\mathcal{Q}_2(D)$. Database D can be assumed to have $p(D) = \{t\}$. Then $\mathcal{V}(D) = I$ and $\mathcal{Q}(D) = \{\}$. Therefore, $\langle c \rangle$ is not a certain answer. □

The previous theorem involves query complexity. However, we are primarily concerned by data complexity, and query complexity results can be misleading. For example, the query complexity of the containment problem of a conjunctive query in a datalog query is EXPTIME-complete, whereas the containment problem of a conjunctive query in a conjunctive query with inequality is considerably easier, namely Π_2^p-complete [39]. In comparison, the data complexity of computing certain answers under OWA for conjunctive view definitions and datalog queries is polynomial, whereas it is presumably harder, namely co-NP-complete, for conjunctive view definitions and conjunctive queries with inequality.

[1] In the case of FO, we use the first-order formula corresponding to this rule.

3.1 Conjunctive view definitions

In this section we consider the complexity of the problem of computing certain answers under OWA in the case of conjunctive view definitions. We will consider queries of different expressive power.

3.1.1 Polynomial cases

The main tool for proving polynomial time bounds is the notion of maximally-contained query plans. We recall the relevant definitions here.

The input of a datalog query $\mathcal{Q}$ consists of a database D storing instances of all EDB predicates in $\mathcal{Q}$. Given such a database D, the output of $\mathcal{Q}$, denoted $\mathcal{Q}(D)$, is an instance of the answer predicate q as determined by, for example, naive evaluation [35]. A datalog query $\mathcal{Q}'$ is *contained* in a datalog query $\mathcal{Q}$ if, for all databases D, $\mathcal{Q}'(D)$ is contained in $\mathcal{Q}(D)$.

A datalog query $\mathcal{P}$ is a *query plan* if all EDB predicates in $\mathcal{P}$ are view literals. The *expansion* $\mathcal{P}^{exp}$ of a datalog query plan $\mathcal{P}$ is obtained from $\mathcal{P}$ by replacing all view literals with their definitions. Existentially quantified variables in view definitions are replaced by new variables in the expansion. A query plan $\mathcal{P}$ is *maximally-contained* in a datalog query $\mathcal{Q}$ w.r.t. a view definition $\mathcal{V}$ if $\mathcal{P}^{exp} \subseteq \mathcal{Q}$, and for each query plan $\mathcal{P}'$ with $(\mathcal{P}')^{exp} \subseteq \mathcal{Q}$, it is the case that $\mathcal{P}'$ is also contained in $\mathcal{P}$. Intuitively, a maximally-contained query plan is the best of all datalog query plans in using the information available from the view instances. As shown in [12], it is easy to construct these maximally-contained query plans in the case of conjunctive view definitions.

Theorem 3.2 shows that maximally-contained query plans compute exactly the certain answers under OWA.

Theorem 3.2 *For $\mathcal{V} \subseteq PQ$, $\mathcal{Q} \in$ datalog, and query plan $\mathcal{P}$ that is maximally-contained in $\mathcal{Q}$ with respect to $\mathcal{V}$, $\mathcal{P}$ computes exactly the certain answers of $\mathcal{Q}$ under OWA for each view instance I.*

Proof. Assume for the sake of contradiction that there is an instance I of the view such that $\mathcal{P}$ fails to compute a certain answer t of $\mathcal{Q}$ under OWA. Let $\mathcal{P}'$ be the query plan that consists of all the rules of $\mathcal{P}$, together with two additional rules r_1 and r_2:

r_1: $q'(X) :- q(X)$

r_2: $q'(t) :- v_1(t_{11}),\ \ldots,\ v_1(t_{1n_1}),\ \ldots,\ v_k(t_{k1}),\ \ldots,\ v_k(t_{kn_k})$

where q is the answer predicate of $\mathcal{P}$, and I is the instance with $I(v_1) = \{t_{11}, \ldots, t_{1n_1}\}$, $\ldots$, $I(v_k) = \{t_{k1}, \ldots, t_{kn_k}\}$. We are going to show that $(\mathcal{P}')^{exp}$ is contained in $\mathcal{Q}$. Since $\mathcal{P}'$ is not contained in $\mathcal{P}$, this contradicts the maximal containment of $\mathcal{P}$ in $\mathcal{Q}$. Therefore, there cannot be a certain answer t under OWA that $\mathcal{P}$ fails to compute.

In order to see that $(\mathcal{P}')^{exp}$ is contained in $\mathcal{Q}$, we have to show that $\mathcal{P}'(\mathcal{V}(D))$ is contained in $\mathcal{Q}(D)$ for each database D. Let D be an arbitrary database. Because $\mathcal{P}^{exp}$ is known to be contained in $\mathcal{Q}$, it suffices to show that $r_2(\mathcal{V}(D))$ is contained in $\mathcal{Q}(D)$. If I is not contained in $\mathcal{V}(D)$, then $r_2(\mathcal{V}(D))$ is the empty set, which is trivially contained in $\mathcal{Q}(D)$. So let us

assume that I is contained in $\mathcal{V}(D)$. Then $r_2(\mathcal{V}(D)) = \{t\}$. Because t is a certain answer under OWA, it follows by definition that t is an element of $\mathcal{Q}(D)$. Therefore, $r_2(\mathcal{V}(D))$ is contained in $\mathcal{Q}(D)$. □

Remark The previous theorem has been generalized in [14]. It is shown that for $\mathcal{V}$ and $\mathcal{Q}$ formulated in arbitrary languages and for query plans formulated in a language $\mathcal{L}$, if there exists a query plan $\mathcal{P} \in \mathcal{L}$ that is maximally-contained in $\mathcal{Q}$ with respect to $\mathcal{V}$, then $\mathcal{P}$ computes all certain answers of $\mathcal{Q}$ under OWA for each view instance I. Moreover, if the language $\mathcal{L}$ is monotone, then $\mathcal{P}$ computes exactly the certain answers. □

As shown in [12] for all $\mathcal{V} \subseteq CQ$ and $\mathcal{Q} \in \mathit{datalog}$, corresponding maximally-contained datalog query plans can be constructed. In the following, we will give an example of the construction of maximally-contained query plans used in [12].

Example 3.1 Consider the view definition $\mathcal{V}$ with

$$v_1(X,Y) :- p(X,Y),\ m(X),$$
$$v_2(X,Y) :- p(X,Z),\ p(Z,Y),\ f(X),$$

and the query $\mathcal{Q}$ with

$$q(X,Y) :- p(X,Y),$$
$$q(X,Y) :- p(X,Z),\ q(Z,Y).$$

v_1 and v_2 can be seen as storing the *mother of* and *grandfather of* relation respectively, and query $\mathcal{Q}$ asks for the ancestor relation. Certainly, the complete ancestor relation cannot be computed given just the data provided by the views.

The maximally-contained datalog query plan can be constructed as follows. First, *inverse rules* are generated that encode how to construct tuples of the database predicates from the view. The inverse rules in our example are the following:

$$\begin{aligned}
m(X) &:- v_1(X,Y)\\
f(X) &:- v_2(X,Y)\\
p(X,Y) &:- v_1(X,Y)\\
p(X,g(X,Y)) &:- v_2(X,Y)\\
p(g(X,Y),Y) &:- v_2(X,Y)
\end{aligned}$$

g is a new function symbol. To understand the role of the function term assume that $I(v_2)$ contains a tuple $\langle a,b\rangle$. Then $\langle a,c\rangle, \langle c,b\rangle \in p(D)$ for *some* constant c. This unknown constant c is automatically named $g(a,b)$ in the inverse rules.

Function symbols can be eliminated from the query plan formed by the relevant inverse rules and query $\mathcal{Q}$. The resulting datalog query plan is the following:

$$\begin{aligned}
q(X,Y) &:- p(X,Y),\\
q(X,Y) &:- p(X,Z),\ q(Z,Y)\\
q(X,Y) &:- p_1(X,Z_1,Z_2),\ q_2(Z_1,Z_2,Y)\\
q_2(X,Y,Z) &:- p_2(X,Y,Z),
\end{aligned}$$

$$
\begin{aligned}
q_2(X_1, X_2, Y) &:- p_2(X_1, X_2, Z),\ q(Z, Y) \\
p(X, Y) &:- v_1(X, Y) \\
p_1(X, X, Y) &:- v_2(X, Y) \\
p_2(X, Y, Y) &:- v_2(X, Y)
\end{aligned}
$$

It has been shown in [12] that the datalog query plans constructed in this fashion are guaranteed to be maximally-contained in $\mathcal{Q}$ with respect to $\mathcal{V}$.

□

The data complexity of evaluating datalog queries is polynomial [40]. The existence of maximally-contained query plans for all $\mathcal{V} \subseteq CQ$ and $\mathcal{Q} \in$ *datalog* shown in [12] therefore implies that the problem of computing certain answers under OWA can be answered in polynomial time.

Corollary 3.1 *For $\mathcal{V} \subseteq CQ$ and $\mathcal{Q} \in$ datalog, the problem of computing certain answers under OWA can be answered in polynomial time.*

3.1.2 Inequalities in the view definition

We next show (Theorem 3.3) that adding inequalities just to the view definition doesn't add any expressive power. The certain answers are exactly the same as if the inequalities in the view definition were omitted. This means that the maximally-contained datalog query constructed from the query and the view definition but disregarding the inequality constraints computes exactly the certain answers. Therefore, the problem remains polynomial.

Theorem 3.3 *Let $\mathcal{V} \subseteq CQ^{\neq}$ and $\mathcal{Q} \in$ datalog. Define $\mathcal{V}^-$ to be the same view definition as $\mathcal{V}$ but with the inequality constraints deleted. Then a tuple t is a certain answer under the open world assumption given $\mathcal{V}$, $\mathcal{Q}$ and a view instance I if and only if t is a certain answer under OWA given $\mathcal{V}^-$, $\mathcal{Q}$ and I.*

Proof. "⇒": Assume that t is a certain answer under OWA given $\mathcal{V}$, $\mathcal{Q}$ and I. Let D be a database with $I \subseteq \mathcal{V}^-(D)$. If also $I \subseteq \mathcal{V}(D)$, then it follows immediately that t is in $\mathcal{Q}(D)$. Otherwise, there is a view definition v in $\mathcal{V}$ and a tuple $s \in I$ such that $s \in v^-(D)$, but $s \notin v(D)$. Let $C \neq C'$ be an inequality constraint in v that disabled the derivation of s in $v(D)$. Because we can assume that s is in $v(D')$ for some database D', at least one of C or C' must be an existentially quantified variable X. Add tuples to D that correspond to the tuples that generate s in $v^-(D)$, but with the constant that X binds to replaced by a new constant. These new tuples then satisfy the inequality constraint $C \neq C'$. By repeating this process for each such inequality constraint $C \neq C'$ and each such tuple s, we arrive at a database D'' with $I \subseteq \mathcal{V}(D'')$. Because t is a certain answer given $\mathcal{V}$, it follows that t is in $\mathcal{Q}(D'')$. Therefore, there are tuples $t_1, \ldots, t_k \in D''$ that derive t. If any t_i contains one of the new constants, replace it by the tuple $t_i' \in D$ that it was originally derived from. Because t doesn't contain any new constants, and because $\mathcal{Q}$ cannot test for inequality, it follows that t is also derived from $t_1', \ldots, t_k'$. Hence t is in $\mathcal{Q}(D)$.

"⇐": Assume that t is a certain answer under OWA given $\mathcal{V}^-$, $\mathcal{Q}$ and I. Let D be a database with $I \subseteq \mathcal{V}(D)$. Because $\mathcal{V}$ is contained in $\mathcal{V}^-$, it follows that $I \subseteq \mathcal{V}^-(D)$, and therefore t is in $\mathcal{Q}(D)$. □

3.1.3 Inequalities in the query

On the other hand, we see next (Theorem 3.4) that adding inequalities to queries does add expressive power. A single inequality in a conjunctive query, even combined with purely conjunctive view definitions, suffices to make the problem co-NP-hard. Van der Meyden proved a similar result [37], namely co-NP hardness for the case $\mathcal{V} \subseteq CQ^<$ and $\mathcal{Q} \in CQ^<$. Our theorem strengthens this result to $\mathcal{V} \subseteq CQ$ and $\mathcal{Q} \in CQ^{\neq}$.

Theorem 3.4 *For $\mathcal{V} \subseteq CQ$, $\mathcal{Q} \in CQ^{\neq}$, the problem of determining whether, given a view instance, a tuple is a certain answer under OWA is co-NP-hard.*

Proof. Let φ be a 3CNF formula with clauses $c_1, \ldots, c_n$, and variables $x_1, \ldots, x_m$. Consider the following view definition $\mathcal{V}$, query $\mathcal{Q}$, and view instance I:

$$\begin{aligned}
\mathcal{V}: \;& v_1(X,Y) :- p(X,Y) \\
& v_2(X,Z) :- p(X,Y),\ p(Y,Z) \\
\mathcal{Q}: \;& q(c) \quad :- p(X,Y_1),\ p(X,Y_2),\ p(X,Y_3),\ p(X,Y_4),\ Y_i \neq Y_j \text{ for } i \neq j \\
I: \;& I(v_1) = \{\langle c_i, p_j\rangle \mid \text{if clause } c_i \text{ contains } x_j\} \\
& \qquad \cup \{\langle c_i, n_j\rangle \mid \text{if clause } c_i \text{ contains } \neg x_j\} \\
& \qquad \cup \{\langle p_j, n_j\rangle, \langle p_j, q_j\rangle, \langle n_j, p_j\rangle, \langle n_j, q_j\rangle, \langle q_j, p_j\rangle, \langle q_j, n_j\rangle, \langle q_j, q_j\rangle \mid j = 1, \ldots, m\} \\
& I(v_2) = \underbrace{\{\langle q_j, 0\rangle \mid j = 1, \ldots, m\}}_{(1)} \cup \underbrace{\{\langle c_i, 1\rangle \mid i = 1, \ldots, n\}}_{(2)}
\end{aligned}$$

We will show that φ is satisfiable if and only if $\langle c\rangle$ is not certain for $\mathcal{V}$, $\mathcal{Q}$, and I. Because the problem of testing a 3CNF formula for satisfiability is NP-complete [9], this implies the claim.

"$\Rightarrow$": Suppose φ is satisfied by some valuation ν. Choose D such that

$$\begin{aligned}
p(D) = I(v_1) \cup \{&\langle p_j, 1\rangle, \langle n_j, 0\rangle \mid \text{if } \nu(x_j) = \mathit{true}\} \\
\cup \{&\langle p_j, 0\rangle, \langle n_j, 1\rangle \mid \text{if } \nu(x_j) = \mathit{false}\}
\end{aligned}$$

Because for $j = 1, \ldots, m$ either $\langle q_j, p_j\rangle, \langle p_j, 0\rangle \in p(D)$ or $\langle q_j, n_j\rangle, \langle n_j, 0\rangle \in p(D)$, (1) is contained in $v_2(D)$. For $i = 1, \ldots, n$, because $\nu(c_i) = \mathit{true}$ there is a variable x_k that appears positively in c_i with $\nu(x_k) = \mathit{true}$ or a variable $x_{k'}$ that appears negatively in c_i with $\nu(x_{k'}) = \mathit{false}$. Therefore, (2) is contained in $v_2(D)$. Hence, I is contained in $\mathcal{V}(D)$. No node has more than three outgoing edges. Hence, $\langle c\rangle$ is not in $q(D)$. It follows that $\langle c\rangle$ is not certain for $\mathcal{V}$, $\mathcal{Q}$, and I.

"$\Leftarrow$": Suppose there is a database D with I contained in $\mathcal{V}(D)$, but $q(D)$ is the empty set. Consider the following valuation ν:

$$\nu(x_j) = \begin{cases} \mathit{true} & : \text{ if } \langle p_j, 1\rangle \in p(D) \\ \mathit{false} & : \text{ otherwise} \end{cases}$$

Consider clause c_i. Because (2) is contained in $v_2(D)$, there is a node z with $\langle c_i, z\rangle, \langle z, 1\rangle \in p(D)$. Because $q(D) = \emptyset$, z is either p_j or n_j for some j in $\{1, \ldots, m\}$. If z is p_j, then x_j

occurs in c_i positively, and therefore $\nu(c_i) = \mathit{true}$. If z is n_j, then x_j occurs in c_i negatively. Because $q(D) = \emptyset$, $\langle n_j, 0\rangle$ can not be in $p(D)$. Because (1) is contained in $v_2(D)$, $\langle p_j, 0\rangle$ must therefore be in $p(D)$. Again because of $q(D) = \emptyset$ this implies that $\langle p_j, 1\rangle$ is not in $p(D)$. Therefore, $\nu(x_j) = \mathit{false}$, and $\nu(c_i) = \mathit{true}$. This proves that ν satisfies φ. □

By Theorem 3.2, we know that maximally-contained queries compute exactly the certain answers under OWA. Because evaluating datalog queries has polynomial data complexity [40], it follows that in general there are *no* datalog queries that are maximally-contained in a conjunctive query with inequality.

3.1.4 First-order queries

We saw that even adding recursion to positive queries leaves the data complexity of the problem of computing certain answers under OWA still polynomial. On the other hand, adding negation makes the problem undecidable for both OWA and CWA, as the following theorem shows.

Theorem 3.5 *For $\mathcal{V} \subseteq CQ$, $\mathcal{Q} \in FO$, the problem of determining, given a view definition together with a view instance, whether a tuple is a certain answer under the open or closed world assumption is undecidable.*

Proof. Let φ be a first-order formula. Consider the query

$$q(c) :- \neg\varphi.$$

Clearly, $\langle c\rangle$ is a certain answer if and only if φ is not satisfiable. Testing whether a first-order formula admits a finite model is undecidable (see [16]). This implies the claim. □

3.2 Positive view definitions

In the previous section, we proved that adding inequalities to the query results in co-NP-completeness of the problem of computing certain answers under OWA. The following theorem shows that allowing disjunction in the view definition has the same effect on the data complexity. The same result was proved by van der Meyden in [38] while studying indefinite databases. We include the theorem for the sake of completeness.

Theorem 3.6 *[38] For $\mathcal{V} \subseteq PQ$, $\mathcal{Q} \in CQ$, the problem of determining, given a view instance, whether a tuple is a certain answer under OWA is co-NP-hard.*

Proof. Let $G = (V, E)$ be an arbitrary graph. Consider the view definition:

$$v_1(X) \quad :- \mathit{color}(X, \mathit{red}) \ \vee \ \mathit{color}(X, \mathit{green}) \ \vee \ \mathit{color}(X, \mathit{blue})$$
$$v_2(X, Y) :- \mathit{edge}(X, Y)$$

and the instance I with $v_1 = V$ and $v_2 = E$. We will show that the query

$$q(c) :- \mathit{edge}(X, Y), \ \mathit{color}(X, Z), \ \mathit{color}(Y, Z)$$

has the tuple $\langle c \rangle$ as a certain answer if and only if graph G is not 3-colorable. Because testing a graph's 3-colorability is NP-complete [22], this implies the claim. For each database D with $I \subseteq \mathcal{V}(D)$, relation *edge* contains at least the edges from E, and relation *color* relates at least the vertices in V to either *red*, *green*, or *blue*. This means, that the databases D with $I \subseteq \mathcal{V}(D)$ are all the assignments of supersets of the vertex set V to colors such that the vertices in V are assigned to *red*, *green*, or *blue*.

"$\Rightarrow$": Assume that $\langle c \rangle$ is a certain answer of the query. It follows that for each assignment of the vertices to *red*, *green*, and *blue*, there is an edge $\langle e_1, e_2 \rangle$ in E such that e_1 and e_2 are assigned to the same color. Therefore, there is not a single assignment of vertices to the three colors *red*, *green*, and *blue* such that all adjacent vertices are assigned to different colors. Hence G is not 3-colorable.

"$\Leftarrow$": Assume G is not 3-colorable. Then for each assignment of supersets of the vertex set V to *red*, *green*, and *blue* there is at least one edge $\langle e_1, e_2 \rangle$ such that e_1 and e_2 are assigned to the same color. It follows that the query will produce $\langle c \rangle$ for each database D with $I \subseteq \mathcal{V}(D)$, i.e. the query has $\langle c \rangle$ as a certain answer. □

Theorem 3.2 tells us that a maximally-contained datalog query $\mathcal{P}$ computes exactly the certain answers under OWA, even if the view definition uses positive queries. On the other hand, the data complexity of datalog is polynomial [40], while the data complexity of the problem of computing certain answers under OWA for positive view definitions is co-NP-complete, as we have just seen. Therefore, maximally-contained datalog queries cannot exist for all $\mathcal{V} \subseteq PQ$ and $\mathcal{Q} \in CQ$. In the following example we are going to give an intuitive reason why maximally-contained datalog queries do not exist for certain choices of $\mathcal{V}$ and $\mathcal{Q}$.

Example 3.2 Consider the following view definition and query:

$$
\begin{aligned}
v_1(X) &:- color(X, red) \vee color(X, green) \vee color(X, blue) \\
v_2(X, Y) &:- edge(X, Y) \\
q(c) &:- edge(X, Y),\ color(X, Z),\ color(Y, Z).
\end{aligned}
$$

In the proof of Theorem 3.6 we showed that if the view instance I is such that $v_1 = V$ and $v_2 = E$ for some graph $G = (V, E)$, then $\langle c \rangle$ is a certain answer if and only if G is not 3-colorable. This implies that each query that makes certain that the input graph is not

Figure 2: *Examples of graphs that are not 3-colorable.*

3-colorable is contained in $\mathcal{Q}$. Figure 2 shows some graphs that are not 3-colorable. For example, queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ that correspond to graphs G_1 and G_2 in Figure 2 respectively are contained in $\mathcal{Q}$:

$$\begin{aligned}\mathcal{Q}_1(c) :- \; & edge(X_1, X_2),\ edge(X_2, X_3),\ edge(X_3, X_1),\\ & edge(X_1, Y),\ edge(X_2, Y),\ edge(X_3, Y)\end{aligned}$$

$$\begin{aligned}\mathcal{Q}_2(c) :- \; & edge(X_1, X_2),\ edge(X_2, X_3),\ edge(X_3, X_4),\ edge(X_4, X_5),\ edge(X_5, X_1),\\ & edge(X_1, Y).\ edge(X_2, Y),\ edge(X_3, Y),\ edge(X_4, Y),\ edge(X_5, Y)\end{aligned}$$

The union of all queries that correspond to non-3-colorable graphs is maximally-contained in $\mathcal{Q}$. But this infinite union of conjunctive queries is not representable by a datalog query. □

Whereas maximally-contained datalog programs might not exists in the case of positive views, maximally-contained *disjunctive* datalog query plans are guaranteed to exist as shown in [14].

3.3 Datalog view definitions

Theorem 2.1 established that the problem can be solved in co-NP for $\mathcal{V} \subseteq PQ^{\neq}$ and $\mathcal{Q} \in datalog^{\neq}$. Here we examine the effect on the complexity of the problem of computing certain answers if we allow datalog as view definition language. For positive queries, the problem stays in co-NP as was shown by van der Meyden in [38]. However, Theorem 3.7 and Corollary 3.2 respectively establish that the problem becomes undecidable for conjunctive queries with inequality and datalog queries.

3.3.1 Inequalities

In the case of conjunctive view definitions, adding inequalities to the query increased the complexity of the problem of computing certain answers under OWA from polynomial to co-NP. With datalog view definitions, adding inequalities to the query raises the problem from co-NP complexity to undecidability. In [37], van der Meyden showed undecidability for the case of $\mathcal{V} \subseteq datalog$ and $\mathcal{Q} \in PQ^{\neq}$. The following theorem proves that the problem is already undecidable for *conjunctive* queries with inequality.

Theorem 3.7 *For $\mathcal{V} \subseteq datalog$, $\mathcal{Q} \in CQ^{\neq}$, the problem of determining, given a view instance, whether a tuple is a certain answer under OWA is undecidable.*

Proof. The proof is by reduction of the Post Correspondence Problem [29] to the problem in the claim.

Let $w_1, \ldots, w_n, w'_1, \ldots, w'_n$ be words over alphabet $\{a, b\}$. In the following, p is a ternary base relation. Consider the following datalog query that defines view v:

$$\begin{aligned}v(0,0) \quad &:- s(e,e,e)\\ v(X,Y) &:- v(X_0, Y_0),\end{aligned}$$

$s(X_0, X_1, \alpha_1), \ldots, s(X_{k-1}, X, \alpha_k),$
$s(Y_0, Y_1, \beta_1), \ldots, s(Y_{l-1}, Y, \beta_l)$
where $w_i = \alpha_1 \ldots \alpha_k$ and $w'_i = \beta_1 \ldots \beta_l$;
one rule for each $i \in \{1, \ldots, n\}$.

$$s(X, Y, Z) :- p(X, X, Y),\ p(X, Y, Z)$$

and query $\mathcal{Q}$ defined by:

$$q(c) :- p(X, Y, Z),\ p(X, Y, Z'),\ Z \neq Z'$$

Let the view instance I be defined by $I(v) = \{\langle e, e\rangle\}$ and $I(s) = \{\}$. We will show that there exists a solution to the instance of the Post Correspondence Problem given by $w_1, \ldots, w_n, w'_1, \ldots, w'_n$ if and only if $\langle c\rangle$ is *not* a certain answer under OWA. The result then follows from the undecidability of the Post Correspondence Problem [29].

Figure 3: *The instance of the Post Correspondence Problem given by the words* $w_1 = ba$, $w_2 = b$, $w_3 = bba$, $w'_1 = ab$, $w'_2 = bb$, *and* $w'_3 = ba$ *has solution "2113" because* $w_2w_1w_1w_3 = bbababba = w'_2w'_1w'_1w'_3$. *The figure shows a database* D *with* $\langle e, e\rangle \in v(D)$, *but* $\mathcal{Q}(D) = \{\}$. *In the rows labeled* $p(D)$ *and* $s(D)$ *in the figure, an arrow from* c_1 *to* c_2 *labeled* c_3 *means that* $\langle c_1, c_2, c_3\rangle$ *is in* $p(D)$ *and* $s(D)$ *respectively. The row labeled* $v(D)$ *illustrates the derivation of tuple* $\langle e, e\rangle$ *in view* v.

"⇒": Assume that the instance of the Post Correspondence Problem given by the words $w_1, \ldots, w_n, w'_1, \ldots, w'_n$ has a solution $i_1, \ldots, i_k$. Then $w_{i_1} \ldots w_{i_k} = w'_{i_1} \ldots w'_{i_k} = \gamma_1 \ldots \gamma_m$ for some characters $\gamma_1, \ldots, \gamma_m \in \{a, b\}$. Consider the database D with

$$p(D) = \{\langle 0, 1, \gamma_1\rangle, \ldots, \langle m-2, m-1, \gamma_{m-1}\rangle, \langle m-1, e, \gamma_m\rangle,$$
$$\langle 0, 0, 1\rangle, \ldots, \langle m-2, m-2, m-1\rangle, \langle m-1, m-1, e\rangle, \langle e, e, e\rangle\}.$$

Clearly, $\mathcal{Q}(D) = \{\}$. Moreover, it is easy to verify that $s(D)$ and $v(D)$ are as follows:

$$s(D) = \{\langle 0, 1, \gamma_1\rangle, \ldots, \langle m-2, m-1, \gamma_{m-1}\rangle, \langle m-1, e, \gamma_m\rangle, \langle e, e, e\rangle\}$$

$$\begin{aligned} v(D) = \{&\langle 0, 0\rangle, \\ &\langle |w_{i_1}|, |w'_{i_1}|\rangle, \\ &\langle |w_{i_1}| + |w_{i_2}|, |w'_{i_1}| + |w'_{i_2}|\rangle, \ldots, \\ &\langle |w_{i_1}| + \ldots + |w_{i_{k-1}}|, |w'_{i_1}| + \ldots + |w'_{i_{k-1}}|\rangle, \\ &\langle e, e\rangle\} \end{aligned}$$

Since $I \subseteq v(D)$ and $\mathcal{Q}(D) = \{\}$, it follows that $\langle c\rangle$ is not a certain answer.

"$\Leftarrow$": Assume that $\langle c\rangle$ is not a certain answer under OWA. Then there is a database D with $I \subseteq v(D)$ such that $\mathcal{Q}(D) = \{\}$. Because tuple $\langle e, e\rangle$ is in $v(D)$, there must be constants $c_0, c_1, \ldots, c_m$ with $c_0 = 0$ and $c_m = e$ and characters $\gamma_1, \ldots, \gamma_m \in \{a, b\}$ such that

$$\langle c_0, c_1, \gamma_1\rangle,\ \langle c_1, c_2, \gamma_2\rangle, \ldots, \langle c_{m-1}, c_m, \gamma_m\rangle \in s(D). \quad (*)$$

Let $d_0, d_1, \ldots, d_{m'}$ be constants with $d_0 = 0$ and $\delta_1, \ldots, \delta_{m'} \in \{a, b\}$ be characters such that

$$\langle d_0, d_1, \delta_1\rangle,\ \langle d_1, d_2, \delta_2\rangle, \ldots, \langle d_{m'-1}, d_{m'}, \delta_{m'}\rangle \in s(D).$$

We are going to show by induction on m' that for $m' \leq m$, $d_i = c_i$ and $\delta_i = \gamma_i$ for $i = 0, \ldots, m'$. The claim is trivially true for $m' = 0$. For the induction case, let $m' > 0$. We know that $\langle c_{i-1}, c_i, \gamma_i\rangle \in s(D)$ and $\langle d_{i-1}, d_i, \delta_i\rangle \in s(D)$, and that $c_{i-1} = d_{i-1}$. By definition of s, this implies that tuples $\langle c_{i-1}, c_{i-1}, c_i\rangle$, $\langle c_{i-1}, c_{i-1}, d_i\rangle$, $\langle c_{i-1}, c_i, \gamma_i\rangle$, and $\langle c_{i-1}, d_i, \delta_i\rangle$ are all in $p(D)$. Because $\mathcal{Q}(D) = \{\}$, it follows that $d_i = c_i$ and $\delta_i = \gamma_i$.
Assume for the sake of contradiction that $m' > m$. Then there is a tuple $\langle d_m, d_{m+1}, \gamma_{m+1}\rangle \in s(D)$, and therefore $\langle d_m, d_m, d_{m+1}\rangle$, $\langle d_m, d_{m+1}, \gamma_{m+1}\rangle \in p(D)$. Because $\langle e, e, e\rangle \in s(D)$, it follows that $\langle e, e, e\rangle \in p(D)$. Since $d_m = c_m = e$ this implies that $d_{m+1} = e$ and $\gamma_{m+1} = e$, which contradicts the fact that $\gamma_{m+1} \in \{a, b\}$. Hence, $m' = m$.
We proved that there is exactly one chain of the form in (*). Because $\langle e, e\rangle \in v(D)$, there is a sequence $i_1 \ldots i_k$ with $i_1, \ldots, i_k \in \{1, \ldots, n\}$ such that $w_{i_1} \ldots w_{i_k} = \gamma_1 \ldots \gamma_m$ and $w'_{i_1} \ldots w'_{i_k} = \gamma_1 \ldots \gamma_m$. Therefore, $i_1, \ldots, i_k$ is a solution to the instance of the Post Correspondence Problem given by $w_1, \ldots, w_n, w'_1, \ldots, w'_n$. □

Theorem 3.7 has an interesting consequence for the query containment problem of a recursive datalog query in a nonrecursive datalog query with inequality. It shows that the technique in [8] to prove decidability of a datalog query in a nonrecursive datalog query does not carry to datalog with inequality. Indeed, it is an easy corollary of Theorems 3.1 and 3.7 that the problem is undecidable.

3.3.2 Datalog queries

As we saw, there is a close relationship between the problem of computing certain answers under OWA and query containment. Not surprisingly it is therefore the case that the problem becomes undecidable for datalog view definitions and datalog queries.

Corollary 3.2 *For $\mathcal{V} \subseteq$ datalog, $\mathcal{Q} \in$ datalog, the problem of determining, given a view instance, whether a tuple is a certain answer under OWA is undecidable.*

Proof. The containment problem of a datalog query in another datalog query is undecidable [33]. Therefore, the claim follows directly from Theorem 3.1. □

3.4 First-order view definitions

Theorem 3.5 showed that adding negation in queries leads to undecidability. The following theorem now shows that the same is true for adding negation to view definitions.

Theorem 3.8 *For $\mathcal{V} \in FO$, $\mathcal{Q} \in CQ$, the problem of determining, given a view instance, whether a tuple is a certain answer under the open or the closed world assumption is undecidable.*

Proof. Let φ be a first-order formula, and p a new predicate. Consider the view definition

$$v(c) :- \varphi(X) \vee p(X)$$

together with the instance $I = \{v(c)\}$ and the query $\mathcal{Q}$ defined by:

$$q(c) :- p(X)$$

We will show that $\langle c \rangle$ is a certain answer under the open or closed world assumption if and only if formula φ is not satisfiable. By Trahtenbrot's theorem, testing whether a first-order formula admits a finite model is undecidable (see [16]). This implies the claim.

"$\Rightarrow$": Suppose that φ is satisfiable. Then there exists a database D such that $\varphi(D)$ is satisfied, and such that $p(D)$ is empty. For this database, $I = v(D)$ and $\mathcal{Q}(D) = \{\}$. Therefore, $\langle c \rangle$ is not a certain answer.

"$\Leftarrow$": Suppose that $\langle c \rangle$ is not certain. Then there is a database D with $I \subseteq \mathcal{V}(D)$ (or with $I = \mathcal{V}(D)$) such that $\langle c \rangle$ is not in $\mathcal{Q}(D)$. Since $p(D)$ is empty, $\varphi(D)$ must be satisfied. Therefore, formula φ is satisfiable. □

4 Closed world assumption

Figure 4 gives an overview of the complexity of the problem of computing certain answers under CWA. Computing certain answers under CWA is harder than under OWA. Whereas the problem is polynomial for $\mathcal{V} \subseteq CQ^{\neq}$ and $\mathcal{Q} \in$ *datalog* under OWA, the problem is already co-NP-complete for $\mathcal{V} \subseteq CQ$ and $\mathcal{Q} \in CQ$ under CWA. Moreover, whereas the problem is decidable for $\mathcal{V} \subseteq$ *datalog* and $\mathcal{Q} \in PQ$ under OWA, the problem is already undecidable for $\mathcal{V} \subseteq$ *datalog* and $\mathcal{Q} \in CQ$ under CWA.

4.1 Conjunctive view definitions

The following theorem shows that computing certain answers under the closed world assumption is already co-NP-hard in the very simplest case, namely in the case of conjunctive view definitions and conjunctive queries.

Theorem 4.1 *For $\mathcal{V} \subseteq CQ$, $\mathcal{Q} \in CQ$, the problem of determining, given a view instance, whether a tuple is a certain answer under CWA is co-NP-hard.*

Proof. Let $G = (V, E)$ be an arbitrary graph. Consider the view definition:

	— query —				
views	CQ	$CQ^{\neq}$	PQ	*datalog*	FO
CQ	co-NP	co-NP	co-NP	co-NP	undec.
$CQ^{\neq}$	co-NP	co-NP	co-NP	co-NP	undec.
PQ	co-NP	co-NP	co-NP	co-NP	undec.
datalog	undec.	undec.	undec.	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Figure 4: *Data complexity of the problem of computing certain answers under the closed world assumption.*

$$\begin{aligned} v_1(X) &:- color(X,Y) \\ v_2(Y) &:- color(X,Y) \\ v_3(X,Y) &:- edge(X,Y) \end{aligned}$$

and the instance I with $I(v_1) = V$, $I(v_2) = \{red, green, blue\}$ and $I(v_3) = E$. We will show that under CWA the query

$$q(c) :- edge(X,Y),\ color(X,Z),\ color(Y,Z)$$

has the tuple $\langle c \rangle$ as a certain answer if and only if graph G is not 3-colorable. Because testing a graph's 3-colorability is NP-complete [22], this implies the claim.

For each database D with $I = \mathcal{V}(D)$, relation *edge* contains exactly the edges from E, and relation *color* relates all vertices in V to either *red*, *green*, or *blue*.

"$\Rightarrow$": Assume that $\langle c \rangle$ is a certain answer of the query. It follows that for each assignment of the vertices to *red*, *green*, and *blue*, there is an edge $\langle e_1, e_2 \rangle$ in E such that e_1 and e_2 are assigned to the same color. Therefore, there is not a single assignment of vertices to the three colors *red*, *green*, and *blue* such that all adjacent vertices are assigned to different colors. Hence G is not 3-colorable.

"$\Leftarrow$": Assume G is not 3-colorable. Then for each assignment of vertices in V to *red*, *green*, and *blue* there exists at least one edge $\langle e_1, e_2 \rangle$ such that e_1 and e_2 are assigned to the same color. It follows that the query will produce $\langle c \rangle$ for each database D with $I = \mathcal{V}(D)$, i.e. the query has $\langle c \rangle$ as a certain answer. □

4.2 Datalog view definitions

The final theorem in this section shows that for datalog view definitions, the problem is undecidable under CWA.

Theorem 4.2 *For $\mathcal{V} \subseteq$ datalog, $\mathcal{Q} \in CQ$ the problem of determining, given a view instance, whether a tuple is a certain answer under CWA is undecidable.*

Proof. The proof is by reduction of the containment problem of datalog queries that is undecidable by [33].

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be two datalog queries with answer predicate q_1 and q_2 respectively. Consider the view definition consisting of the rules of $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and the rules

$$v_1(c) :- r(X)$$
$$v_1(c) :- q_1(X),\ p(X)$$
$$v_2(c) :- q_2(X),\ p(X)$$

where p and r are two relations not appearing in $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Consider the instance I with $I(v_1) = \{\langle c\rangle\}$ and $I(v_2) = \{\}$, and the query $\mathcal{Q}$ defined by:

$$q(c) :- r(X)$$

If $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$, then for each database D with $\mathcal{V}(D) = I$,

$$q_1(D) \cap p(D) \subseteq q_2(D) \cap p(D) = I(v_2) = \{\}.$$

Therefore,

$$r(D) = I(v_1) = \{\langle c\rangle\},$$

i.e. $\langle c\rangle$ is a certain answer under CWA.

On the other hand, if $\mathcal{Q}_1 \not\subseteq \mathcal{Q}_2$, then there is a database D such that some tuple t is in $\mathcal{Q}_1(D)$, but not in $\mathcal{Q}_2(D)$. By extending D such that $p(D) = \{t\}$ and $r(D) = \{\}$, we have that $\mathcal{V}(D) = I$. Because $q(D) = \{\}$, $\langle c\rangle$ is not a certain answer under CWA.

We established that $\langle c\rangle$ is a certain answer under CWA if and only if $\mathcal{Q}_1$ is contained in $\mathcal{Q}_2$. The claim now follows from the undecidability of containment of datalog queries. □

5 Using Conditional Tables

In this section, we first recall the notion of conditional table. (For more on this topic, see [20], where they are introduced or [1].) Then we show an effective way – based on these tables – of computing queries using views for positive views and datalog queries.

Intuitively, a conditional table is a database instance which might have variables as entries in its tuples. There is also a global condition [1] on the set of variables and for each tuple, a local condition controlling the actual presence of the tuple. A *possible database* for a table T is obtained by choosing a valuation satisfying the global condition, keeping only those tuples with a true local condition and valuating the variables in those tuples. So, a conditional table represents a set of possible databases. These tables were introduced by Imielinski and Lipski [20] who also showed how to query conditional tables.

The technique presented in this section is based on the intuition that given a view instance, one can represent the set of possible databases (or, more precisely, a sufficient set of possible databases) by a conditional table, and then answer queries using conditional table technology. To simplify the presentation, we consider here that both the database instance D and the view instance I consist of a single relation. As a consequence, we may focus on conditional tables over a single relation. The generalization to multiple relations is straightforward.

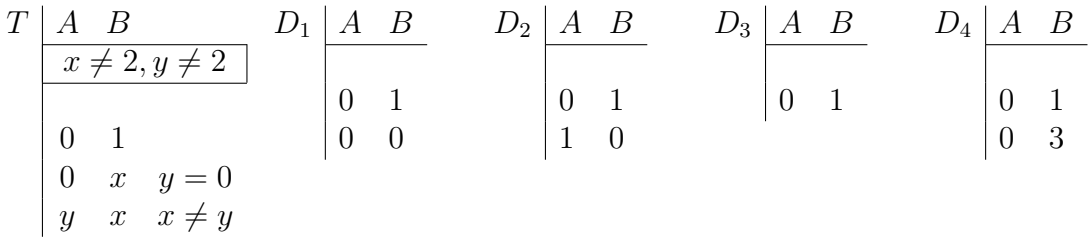

Figure 5: A conditional table and some possible databases

A *condition* is a *conjunct* of *equality atoms* of the form $x = y$ or $x = c$, and of *inequality atoms* of the form $x \neq y$ or $x \neq c$, where x and y are variables and c is a constant. Note that *true* and *false* can be encoded as $x = x$ and $x \neq x$ respectively. A valuation ν (of variables to constants) *satisfies* a condition Φ, denoted $\nu \models \Phi$, if its assignment of variables to constants makes the formula true.

A *table* is a relation where the entries may be either constants or variables. Conditions may be associated with a table T in two ways: (1) a *global* condition Φ is associated with the entire table T; (2) a *local* condition φ_t is associated with tuple t of table T. A *conditional table* (*c-table* in short) is a triple (T, Φ, φ) where

- T is a table,
- Φ is a global condition,
- φ is a mapping over T that associates a local condition φ_t to each tuple t of T.

A c-table is shown in Figure 5. If we omit listing a condition, then it is by default the atom *true*. Note also that conditions Φ and φ_t for t in T may contain variables not appearing in T or t respectively. For brevity, we usually refer to a c-table (T, Φ, φ) simply as T, when Φ and φ are understood.

Given a valuation ν and a c-table T, the *valuation* of T by ν is defined by:

$$\nu(T) = \{\nu(t) \mid t \text{ in } T,\ \nu \models \varphi_t\}$$

Then a c-table T (with Φ and φ understood) represents a set of possible databases as follows:

$$rep(T) = \{\nu(T) \mid \nu \models \Phi\}$$

We will say that a tuple t is a *certain* answer for query $\mathcal{Q}$ given a table T, if for each D in $rep(T)$, t is in $\mathcal{Q}(D)$. Imielinski and Lipski also consider an open-world interpretation for tables. But we will mostly ignore this aspect here in order not to confuse with our CWA/OWA for views.

Consider the table T in Figure 5. Then D_1, D_2, D_3, and D_4 are obtained by valuating x, y to $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(3, 0)$ respectively.

We are interested in querying incomplete information. The answer to a query $\mathcal{Q}$ on an infinite set of possible databases $\{D_1, D_2, \ldots\}$ can be defined as $\{\mathcal{Q}(D_1), \mathcal{Q}(D_2), \ldots\}$. It turns out that if we start with a c-table, we are able to represent the answers by a c-table as well if the query is in relational algebra [20]. Using similar techniques, one can show:

Theorem 5.1 *[18] For each c-table T over U and a datalog$^{\neq}$ query $\mathcal{Q}$ over U, one can construct a c-table $T_{\mathcal{Q}}$ such that $rep(T_{\mathcal{Q}}) = \mathcal{Q}(rep(T))$.*

We next see (Theorem 5.2) how the problem of querying materialized views can be reduced to the problem of querying conditional tables, thereby highlighting the strong connection between materialized views and incomplete databases. The construction used in the proof is illustrated first by an example.

Example 5.1 Suppose the view is specified by:

$$v(0, Y) \ :- \ p(0, Y)$$
$$v(X, Y) :- \ p(X, Z), \ p(Z, Y)$$

and the view instance I consists of $\{v(0,1), v(1,1)\}$. Then there are two different ways to obtain the first tuple and only one for the second. This yields the following conditional table for p (the global condition is *true*):

0	1	$w = 1$
0	z	$w \neq 1$
z	1	$w \neq 1$
1	u	*true*
u	1	*true*

This is the table needed for OWA. For instance, let $w = z = u = 2$. Then the corresponding database D is $\{p(0,2), p(2,1), p(1,2)\}$. Observe that, because of the second rule, $v(2,2)$ is in $\mathcal{V}(D)$. So, some tuples may be in the view that are not in I.

For CWA, we have to introduce the constraint that only tuples in I may be in the view. So, consider the last two tuples of the table. They may join in the second rule to yield $\langle u, u\rangle$. This tuple must be in I, so u has to be 1. Indeed, one can find that under CWA the result is the following (complete) table for p:

0	1	*true*
1	1	*true*

Note that this table is a "regular" relation because no incompleteness is expressed. □

The following construction will be needed towards Theorem 5.2. Let $\mathcal{V} \subseteq PQ^{\neq}$ be a view definition and let I be a view instance. We assume here that the view is consistent under OWA, i.e. that there exists a database D such that $I \subseteq \mathcal{V}(D)$. We now describe the construction of a table T_{owa} such that the certain answers to any query $\mathcal{Q} \in datalog^{\neq}$ using the view are the same as the certain answers to $\mathcal{Q}$ using T_{owa}. So, in particular, we could use this table to compute certain answers to queries. For the construction, we first consider conjunctive queries, and then treat disjunctions.

Conjunctive view definition (with inequality): We can assume without loss of generality that the view is defined by a rule r of the form:[2]

$$v(X_1, ..., X_n) :- p(Y_1^1, \ldots, Y_{n'}^1), \ldots, p(Y_1^{q_r}, \ldots, Y_{n'}^{q_r}),\ A_1, \ldots, A_p$$

where the A_i's are equalities and inequalities, and $X_1, \ldots, X_n, Y_1^1, \ldots, Y_{n'}^1, \ldots, Y_1^{q_r}, \ldots, Y_{n'}^{q_r}$ are all distinct variables. Let $u = \langle a_1, ..., a_n \rangle$ be a tuple in I. Consider the substitution θ_u such for each i, $\theta_u(X_i) = a_i$, and for each i, k, $\theta_u(Y_i^k) = Y_{i,u}^k$, where for each k, i, u, $Y_{i,u}^k$ is a distinct new variable. Now let, for each $u \in I$ and for $k = 1, \ldots, q_r$:

$$\begin{aligned} t_{u,r}^k &= \langle Y_{1,u}^k, \ldots, Y_{n,u}^k \rangle, \\ \psi_r(t_{u,r}^k) &= \theta_u(A_1) \wedge \ldots \wedge \theta_u(A_p). \end{aligned}$$

The resulting c-table consists of $(T_{owa}, true, \psi_r)$ with

$$T_{owa} = \{t_{u,r}^k \mid u \in I,\ k = 1, \ldots, q_r\}.$$

Of course, a lot of simplifications can be performed on this table, notably replacing each variable Z by constant c if $Z = c$ is a condition.

Positive view definition (with inequality): We can assume without loss of generality that the view definition is given in the form of a set of conjunctive queries $\{r_1, \ldots, r_m\}$. Now a difficulty is that a tuple u in I may be obtainable with several conjunctive queries. It would be wrong to force the table to have machineries to derive the tuple u m times. We generalize the technique used in the example. For each tuple u in view I, let Z_u be a new variable. The resulting c-table consists of $(T_{owa}, true, \varphi)$ with

$$\begin{aligned} T_{owa} &= \{t_{u,r_i}^k \mid u \in I,\ i = 1, \ldots, m,\ k = 1, \ldots, q_r\}, \\ \varphi(t_{u,r_i}^k) &= \psi_{r_i}(t_{u,r_i}^k) \wedge Z_u = i \quad \text{for } i = 1, \ldots, m-1, \\ \varphi(t_{u,r_m}^k) &= \psi_{r_m}(t_{u,r_m}^k) \wedge Z_u \neq 1 \wedge \ldots \wedge Z_u \neq m-1. \end{aligned}$$

We next turn to the theorem.

Theorem 5.2 *Let $\mathcal{V} \subseteq PQ^{\neq}$ and let I be a view instance. Then one can construct a conditional table T_{owa}, such that for each query $\mathcal{Q} \in datalog^{\neq}$, a tuple u is a certain answer to $\mathcal{Q}$ given $\mathcal{V}$ and I under OWA if and only if it is a certain answer to query $\mathcal{Q}$ using T_{owa}.*

Proof. Let $\mathcal{V} \subseteq PQ^{\neq}$ and let I be a view instance and T_{owa} defined as above. We first prove that:

$$\text{For each } D \text{ in } rep(T_{owa}),\ I \subseteq \mathcal{V}(D). \tag{3}$$

Let D be in $rep(T_{owa})$ for some valuation ν. Consider some u in I. If $\nu(Z_u)$ is in $\{1, \ldots, m-1\}$, let i be $\nu(Z_u)$. Otherwise, let i be m. It is easy to see that $\{\nu(t_{u,r_i}^k) \mid k = 1, \ldots, q_r\}$ (a subset of D) allows to derive u. The assumption that I is consistent is crucial here. With an inconsistent I, a local condition *false* may prevent from obtaining all I. So, u is in $\mathcal{V}(D)$. Hence $I \subseteq \mathcal{V}(D)$.

Thus, using view I under the OWA, each database in $rep(T_{owa})$ is a possible database. We now show that each possible database contains a database in $rep(T_{owa})$:

[2]Recall that we assume for the sake of a clear presentation that the database consists of a single relation.

For each D such that $I \subseteq \mathcal{V}(D)$, there exists $D' \in rep(T_{owa})$ such that $D' \subseteq D$. (4)

Let D be such that $I \subseteq \mathcal{V}(D)$. Let u be a tuple in I. This tuple must be derivable using some rule, say r_j, and some tuples in D. We choose $\nu(Z_u)$ to be j and fix for the variables of t^k_{u,r_j} appropriate values for ν in the obvious way. For the variables in t^k_{u,r_i} for $i \neq j$, we can choose arbitrary values for ν since they have no effect. We do this for each u. Consider the valuation ν that is obtained and $D' = \nu(T_{owa})$. Then D' is a subset of D.

Now we are in a position to prove the claim:

"$\Rightarrow$": Assume that u is a certain answer under OWA to $\mathcal{Q}$ given $\mathcal{V}$ and I. Then for all D such that $I \subseteq \mathcal{V}(D)$, u is in $\mathcal{Q}(D)$. Let D be in $rep(T_{owa})$. By (3), $I \subseteq \mathcal{V}(D)$. Thus, u is in $\mathcal{Q}(D)$. Hence, u is a certain answer to $\mathcal{Q}$ using T_{owa}.

"$\Leftarrow$": Assume that u is not a certain answer under OWA to $\mathcal{Q}$ given $\mathcal{V}$ and I. This means that for some D such that $I \subseteq \mathcal{V}(D)$, $u \notin \mathcal{Q}(D)$. Then by (4) there is D' in $rep(T_{owa})$ such that $D' \subseteq D$. By monotonicity, $u \notin \mathcal{Q}(D')$. Thus u is not a certain answer to $\mathcal{Q}$ using T_{owa}. □

The previous theorem provides an algorithm for evaluating *datalog*$^{\neq}$ queries on a database given some materialized view defined using $PQ^{\neq}$:

(i) compute the c-table T_{owa} representing the database (preprocessing);

(ii) for each query $\mathcal{Q}$, compute the c-table representing $\mathcal{Q}(T_{owa})$; and

(iii) compute the certain tuples in the c-table for $\mathcal{Q}(T_{owa})$.

The first two steps are in PTIME although the *simplification* of the result table (finding a more compact (minimal) equivalent c-table is not). Of course, we cannot expect to obtain in general a better bound than co-NP.

The following remark shows that T_{owa} is not only sufficient to obtain the certain answers to queries but that it describes in some sense perfectly the knowledge we have of the database given $\mathcal{V}$ and I. Although not done here this would permit to use T_{owa} for other purposes such as verifying whether a tuple may *possibly* be in D given $\mathcal{V}$ and I.

Remark As previously mentioned, one can give an OWA of a c-table. It turns out that the set of possible databases according to the OWA of T_{owa} is exactly the set of possible databases under OWA given view instance I and view definition $\mathcal{V}$. In the CWA case, we will not be able to describe so accurately the set of possible databases given a view. But, we will approach close enough to again be able to obtain the certain answers to datalog queries. □

To conclude this section, we briefly consider CWA for views.

Closed world assumption

Theorem 5.3 *Let $\mathcal{V} \subseteq PQ^{\neq}$ and let I be a view instance. Then one can construct a conditional table (with global condition) T_{cwa}, such that for each $\mathcal{Q} \in datalog^{\neq}$, a tuple u is a certain answer to $\mathcal{Q}$ under CWA given $\mathcal{V}$ and I if and only if it is a certain answer to query $\mathcal{Q}$ using T_{cwa}.*

Proof. When we considered the OWA, we made sure that the proper machinery was in the c-table to at least derive I. However, many of the databases that we considered possible could derive more tuples and we considered that fine. Now, under the CWA, some of these databases are not feasible. This can be viewed as a *constraint* on the set of possible databases. We know that for each D in $rep(T_{owa})$, $I \subseteq \mathcal{V}(D)$. We would like also the *constraint*, that $\mathcal{V}(D) \subseteq I$. Let us illustrate how this constraint can be handled in Example 5.1. The second rule may be applied on the first and 4th row of the c-table. It yields the tuple $\langle 0, u\rangle$ assuming the condition $w = 1 \wedge true$. (We do not perform any simplification to make the point clearer.) Now, the CWA, forces this tuple to be in I, so we have to add as a global condition that this tuple is in I, i.e., that it is $\langle 0, 1\rangle$ or $\langle 1, 1\rangle$. This introduces the following conjunct to the global condition:

$$w = 1 \wedge true \wedge ((0 = 0 \wedge u = 1) \vee (0 = 1 \wedge u = 1))$$

By applying the same construction, we end up with a very large global condition Φ. (This condition is likely to simplify dramatically in practical cases.) Consider T_{cwa} consisting of the same table, the same local condition, but Φ as the global condition. On can verify:

$$\text{For each } D \text{ in } rep(T_{cwa}),\ I = \mathcal{V}(D). \tag{5}$$

$$\text{For each } D \text{ such that } I = \mathcal{V}(D), \text{ there exists } D' \in rep(T_{owa}) \text{ such that } D' \subseteq D. \tag{6}$$

To see (5): Let D be in $rep(T_{cwa})$. Clearly, D is also in $rep(T_{owa})$, so $I \subseteq \mathcal{V}(D)$ by (3). By construction of Φ, $\mathcal{V}(D) \subseteq I$. To see (6): The construction is as in the OWA case. The fact that the global condition is verified is guaranteed because of D.

By (5), any certain answer under CWA given $\mathcal{V}$ and I is also a certain answer using T_{cwa}. By (6) and the fact that the query is monotone, any answer that is not certain under CWA given $\mathcal{V}$ and I, is also not certain using T_{cwa}. □

It should be observed that we do not have to require as before that I is a consistent view instance. If we start with an arbitrary instance I that happens to be inconsistent, the table T_{cwa} that will be constructed will just happen to describe an empty set of databases. Furthermore, suppose that we also know that the database satisfies certain *total dependencies* [17, 42]. Then we can *chase* [4, 3] the c-table as in [20, 18].

We conclude this section by a remark that states the impossibility to completely capture, with c-tables, the possible databases given a view under CWA.

Remark In the previous remark we observed that the set of possible databases for T_{owa} under OWA coincides with the set of possible databases given $\mathcal{V}$ and I under OWA. The same property does not hold for views under CWA. Indeed, one can show that there is no c-table (neither OWA or CWA) that captures exactly the set of possible databases given a view I under CWA. So, to capture such forms of incompleteness a more refined model of incompleteness has to be used. We sketch a proof of that fact. This sketch may be skipped by readers not familiar with c-tables under CWA and OWA.

Suppose the database D is over a 2-ary relation and the view is simply the selection:

$$v(X) :- p(0, X)$$

Consider an empty view instance I. Let $\mathcal{D}$ be the set of possible databases given $\mathcal{V}$ and I under CWA. Suppose that $\mathcal{D}$ can be described by a c-table T under CWA. Then, by definition, $\mathcal{D} = rep(T)$. But suppose that T has n tuples for some n. This would imply that any instance in $\mathcal{D}$ has at most n tuples, which is clearly in contradiction with the definition of $\mathcal{D}$. Now suppose that $\mathcal{D}$ can be described by a c-table T under OWA. Then, for each D in $\mathcal{D}$ and each binary relation D', $D \cup D'$ is in $\mathcal{D}$ (by definition of OWA for c-tables). This is clearly in contradiction with the definition of $\mathcal{D}$. □

6 View consistency and view self-maintainability

In this section, we consider two other important problems on materialized views, view consistency and view self-maintainability. We do it in the context of CWA since both of these problems make more sense in that context than under OWA.

Definition 6.1 (**view consistency**) Let $\mathcal{V}$ be a view definition and I an instance of the view. Then the view is *consistent* if there is a database D such that $I = \mathcal{V}(D)$. □

Definition 6.2 (**view self-maintainability**) Let D be a database. An *update* to D is either a deletion $d(t)$ of a tuple t in D, or an insertion $i(t)$ of some tuple t not in D. Let $\mathcal{V}$ be a view definition and I a consistent view instance. Then the view is *self-maintainable* for an update α if there exists a view instance J such that for each D with $I = \mathcal{V}(D)$, $J = \mathcal{V}(\alpha(D))$. □

views	consistency	self-maintainability
CQ	NP	co-NP
$CQ^{\neq}$	NP	co-NP
PQ	NP	co-NP
datalog	undec.	undec.
FO	undec.	undec.

Figure 6: *Data complexity of the view consistency and the view self-maintainability problem.*

The complexity of these problems is shown in Figure 6. The complexity table for self-maintainability is the same as the one for the problem of computing certain answers under CWA in Figure 4 for conjunctive queries. The complexity of the view consistency problem is similar with NP in place of co-NP. Note that the undecidable cases for the view consistency problem are r.e., whereas for computing certain answers and self-maintainability, they are co-r.e.

Theorem 6.1

(i) For $\mathcal{V} \subseteq PQ^{\neq}$, the view consistency problem is in NP*, and the view self-maintainability problem is in co-*NP *(w.r.t. the size of the view).*

(ii) For $\mathcal{V} \subseteq CQ$, the view consistency problem is NP*-hard, and the view self-maintainability problem is co-*NP*-hard (w.r.t. the size of the view).*

(iii) For $\mathcal{V} \subseteq$ datalog or $\mathcal{V} \subseteq FO$, the view consistency problem is undecidable (r.e.), and the view self-maintainability problem as well (co-r.e.).

Proof.

(i) *(view consistency)* Let $\mathcal{V} \subseteq PQ^{\neq}$ be a view definition and let I be a consistent view instance. Then there is a database D with $I = \mathcal{V}(D)$. Let n be the total number of tuples in I and let k be the maximal number of conjuncts in the view definition. For each tuple u, there exist k' tuples in D for some $k' \leq k$ that suffice to generate u. Therefore (since $\mathcal{V}$ is monotone), there is a database D' with at most nk tuples such that $I = \mathcal{V}(D')$. Because checking that $I = \mathcal{V}(D')$ can be done in polynomial time it follows that the view consistency problem is in NP for $\mathcal{V} \subseteq PQ^{\neq}$.

Let $\mathcal{Q} \in CQ$, $\mathcal{V}$, I, and t be an instance of the problem of computing certain answers. Let $\mathcal{Q}_{body}[t]$ be the body of $\mathcal{Q}$ with head variables instantiated corresponding to t. Let v be a new view literal. Consider the view definition $\mathcal{V}'$ that contains all of $\mathcal{V}$ and the following additional definition:

$$v(c) :- \mathcal{Q}_{body}[t]$$

We will show that $\mathcal{V}'$ and I are consistent if and only if t is *not* a certain answer under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I. It follows from Theorems 4.1, 4.2, and 3.8 that the view consistency problem is NP-hard for $\mathcal{V} \subseteq CQ$, and undecidable for $\mathcal{V} \subseteq$ *datalog* and $\mathcal{V} \subseteq FO$.

"$\Rightarrow$": Assume that $\mathcal{V}'$ and I are consistent. Then there is a database D with $I = \mathcal{V}'(D)$. Because $v(c) \notin I$, it follows that $t \notin q(D)$. Because it is the case that $I = \mathcal{V}(D)$, t is not a certain answer under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I.

"$\Leftarrow$": Assume that t is not a certain answer under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I. Then there is a database D with $I = \mathcal{V}(D)$ and $t \notin q(D)$. Therefore, $\mathcal{V}'$ and I are consistent.

(ii) *(view self-maintainability)* Let $\mathcal{V} \subseteq PQ^{\neq}$ be a view definition and let I be a consistent view instance that is *not* self-maintainable for an update α. Then there are databases D_1 and D_2 such that $I = \mathcal{V}(D_1) = \mathcal{V}(D_2)$, but $\mathcal{V}(\alpha(D_1)) \neq \mathcal{V}(\alpha(D_2))$. Without loss of generality, let t be a tuple in $\mathcal{V}(\alpha(D_1))$ that is not in $\mathcal{V}(\alpha(D_2))$. Let $t_1 \in p_1(D_1), \ldots, t_m \in p_m(D_1)$ be the tuples needed to generate t. As before, let $D_1' \subseteq D_1$ and $D_2' \subseteq D_2$ be databases with at most nk tuples and $I = \mathcal{V}(D_1') = \mathcal{V}(D_2')$. Let D_1'' be the database $D_1' \cup \{t_1, \ldots, t_m\}$. Because of the monotonicity of $\mathcal{V}$ it is the case that $I = \mathcal{V}(D_1'')$, and that $t \notin \mathcal{V}(\alpha(D_2'))$. Since $t \in \mathcal{V}(\alpha(D_1''))$ it follows that $\mathcal{V}(\alpha(D_1'')) \neq \mathcal{V}(\alpha(D_2'))$. Because the size of D_1'' and D_2 is polynomially bounded by the size of $\mathcal{V}$ and I, and because checking that $\mathcal{V}(\alpha(D_1'')) \neq \mathcal{V}(\alpha(D_2'))$ can be done in polynomial time, it follows that the self-maintainability problem is in co-NP for $\mathcal{V} \subseteq PQ^{\neq}$.

Let $G = (V, E)$ be an arbitrary graph. Consider the view definition:

$$
\begin{aligned}
v_1(X) &:- color(X, Y) \\
v_2(Y) &:- color(X, Y) \\
v_3(X, Y) &:- edge(X, Y) \\
v_4(c) &:- edge(X, Y),\ color(X, Z),\ color(Y, Z),\ r(c)
\end{aligned}
$$

and the instance I with $I(v_1) = V$, $I(v_2) = \{red, green, blue\}$, $I(v_3) = E$, and $I(v_4) = \emptyset$. We will show that $\mathcal{V}$ and I are self-maintainable for the insertion of $\langle c \rangle$ into r if and only if G is *not* 3-colorable. Because testing a graph's 3-colorability is NP-complete [22], this proves that the self-maintainability problem for $\mathcal{V} \subseteq CQ$ is co-NP-hard.

"$\Rightarrow$": Assume G is 3-colorable. Then there are databases D_1 and D_2 with $I = \mathcal{V}(D_1) = \mathcal{V}(D_2)$, $v_4(c) \in \mathcal{V}(D_1 \cup \{r(c)\})$, and $v_4(c) \notin \mathcal{V}(D_2 \cup \{r(c)\})$. Therefore, $\mathcal{V}$ and I are not self-maintainable for the insertion of $\langle c \rangle$ into r.

"$\Leftarrow$": Assume G is not 3-colorable. Then for every database D with $I = \mathcal{V}(D)$ it is the case that $v_4(c) \in \mathcal{V}(D \cup \{r(c)\})$. Therefore, $\mathcal{V}$ and I are self-maintainable for the insertion of $\langle c \rangle$ into r.

Let $\mathcal{Q} \in CQ$, $\mathcal{V} \subseteq datalog$, I, and t be an instance of the problem of computing certain answers. Let $\mathcal{Q}_{body}[t]$ be the body of $\mathcal{Q}$ with head variables instantiated corresponding to t. Let r_1 and r_2 be new database predicates, and let v be a new view literal. Let $\mathcal{V}'$ be all of $\mathcal{V}$ plus the following definition:

$$
\begin{aligned}
v(c) &:- \mathcal{Q}_{body}[t] \\
v(c) &:- r_1(c) \\
v(c) &:- r_2(c)
\end{aligned}
$$

Let I' be $I \cup \{v(c)\}$. We will show that t is certain under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I if and only if $\mathcal{V}'$ and I' is self-maintainable for the deletion of $\langle c \rangle$ from r_1. It follows from Theorem 4.2 that the view maintainability problem is undecidable for $\mathcal{V} \subseteq datalog$. The proof for $\mathcal{V} \subseteq FO$ is similar.

"$\Rightarrow$": Assume that t is certain under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I. Then for every database D with $I' = \mathcal{V}'(D)$, it is the case that $I' = \mathcal{V}'(D - \{r_1(c)\})$. Therefore, $\mathcal{V}'$ and I' are self-maintainable for the deletion of $\langle c \rangle$ from r_1.

"$\Leftarrow$": Assume that $\mathcal{V}'$ and I' are self-maintainable for the deletion of $\langle c \rangle$ from r_1. Because there is a database D with $I' = \mathcal{V}'(D)$ and $\langle c \rangle \in r_2(D)$, it is the case that $v(c) \in \mathcal{V}'(D - \{r_1(c)\})$. Because for every database D with $I = \mathcal{V}(D)$ there is a database D' with $I' = \mathcal{V}'(D')$ and $\langle c \rangle \notin r_2(D)$, it follows that t is a certain tuple under CWA given $\mathcal{V}$, $\mathcal{Q}$, and I.

□

7 Conclusion

We presented some complexity results with respect to materialized views. A main contribution is (i) the exhibition of deep connections with incomplete databases and (as a

consequence) (ii) the point of view that a materialized view should be seen as an incomplete database. This indeed suggests using some model of incomplete information as the view model. We will illustrate briefly this direction with an example. Consider the self-maintainability problem of materialized views. Suppose we have such a view, the database is unavailable and we receive some updates to the database. A known technique is to verify whether the view is self-maintainable. If it is not, we raise our hands and in principle the view becomes unavailable. However, one could consider updating the incomplete database corresponding to the view. We could continue answering queries, and indeed, with such a model, it is possible to have more semantics in our answers, e.g. provide besides *certain* answers, *possible* answers, or indicate whether our answer is *surely* complete or not. We intend to continue the present work in that direction.

Acknowledgments

We would like to thank Moshe Y. Vardi and Alon Y. Levy for pointing us to the work of Ron van der Meyden on the complexity of querying indefinite databases, and Michael R. Genesereth, Pierre Huyn, Werner Nutt, Anand Rajaraman, and Yehoshua Sagiv for discussions on the topic. Special thanks to Marc Friedman, Alon Y. Levy, and Todd D. Millstein for pointing out a flaw in a proof in a previous version of this paper.

References

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

[2] S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. *Theoretical Computer Science*, 78(1):159–187, 1991.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimiziation of a class of relational expressions. *ACM Transactions on Database Systems*, 4(4):435–454, 1979.

[4] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal on Computing*, 8(3):218–246, 1979.

[5] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, p. 99–108, Tucson, AZ, 1997.

[6] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on the Theory of Computing*, p. 77–90, 1977.

[7] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the Eleventh International Conference on Data Engineering*, IEEE Comput. Soc. Press, p. 190–200, Los Alamitos, CA, 1995.

[8] S. Chaudhuri and M. Y. Vardi. On the equivalence of recursive and nonrecursive datalog programs. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*, p. 55–66, San Diego, CA, 1992.

[9] S. A. Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Sympoisum on Theory of Computing*, p. 151–158, Shaker Heights, OH, 1971.

[10] G. Dong and J. Su. Conjunctive query containment with respect to views and constraints. *Information Processing Letters*, 57(2):95–102, 1996.

[11] O. M. Duschka. Query optimization using local completeness. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, p. 249–255, Providence, RI, 1997.

[12] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, p. 109 – 116, Tucson, AZ, 1997.

[13] O. M. Duschka and M. R. Genesereth. Query planning in Infomaster. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, San Jose, CA, 1997.

[14] O. M. Duschka and M. R. Genesereth. Query planning with disjunctive sources. In *Proceedings of the AAAI Workshop on AI and Information Integration*, Madison, WI, 1998.

[15] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI*, Nagoya, Japan, 1997.

[16] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[17] R. Fagin and M. Y. Vardi. The theory of data dependencies: A survey. In M. Anshel and W. Gewirtz, editors, *Mathematics of Information Processing: Proceedings of Symposia in Applied Mathematics*, Vol. 34, p. 19–71. American Mathematical Society, Providence, RI, 1986.

[18] G. Grahne. *Problem of incomplete information in relational databases*. Springer-Verlag, 1991.

[19] N. Huyn. Maintaining data warehouses under limited source access. Ph.D. Thesis STAN-CS-TR-97-1595, Stanford University, 1997.

[20] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[21] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *Journal of Computer and System Sciences*, 28:167–189, 1984.

[22] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, p. 85–104, 1972.

[23] A. Klug. On conjunctive queries containing inequalities. *Journal of the Association for Computing Machinery*, 35(1):146–160, 1988.

[24] W. J. Labio, Y. Zhuge, J. L. Wiener, H. Gupta, H. Garcia-Molina, and J. Widom. The WHIPS prototype for data warehouse creation and maintenance. In *Proceedings ACM SIGMOD International Conference on Management of Data*, p. 557–559, Tucson, AZ, 1997.

[25] A. Y. Levy, A. O. Mendelzon, D. Srivastava, and Y. Sagiv. Answering queries using views. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, San Jose, CA, 1995.

[26] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the 22nd International Conference on Very Large Databases*, p. 251–262, Bombay, India, 1996.

[27] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Montreal, Canada, 1996.

[28] A. Y. Levy and D. Suciu. Deciding containment for queries with complex objects. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '97*, p. 32–43, Tucson, AZ, 1997.

[29] E. Post. A variant of a recursively unsolvable problem. *Bulletin American Mathematical Society*, 52:264–268, 1946.

[30] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1995.

[31] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[32] O. Shmueli. Decidability and expressiveness aspects of logic queries. In *Proceedings of the Sixth ACM Symposium on Principles of Database Systems*, p. 237–249, 1987.

[33] O. Shmueli. Equivalence of datalog queries is undecidable. *Journal of Logic Programming*, 15:231–241, 1993.

[34] J. D. Ullman. *Principles of Database and Knowledgebase Systems*, Vol. 1. Computer Science Press, 1988.

[35] J. D. Ullman. *Principles of Database and Knowledgebase Systems*, Vol. 2. Computer Science Press, 1989.

[36] J. D. Ullman. Information integration using logical views. In *Proceedings of the Sixth International Conference on Database Theory*, 1997.

[37] R. van der Meyden. The complexity of querying indefinite information: Defined relations, recursion and linear order. Technical report, Rutgers University, 1992.

[38] R. van der Meyden. Recursively indefinite databases. *Theoretical Computer Science*, 116(1):151–194, 1993.

[39] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *Journal of Computer and System Sciences*, 54(1):113 – 135, 1997.

[40] M. Y. Vardi. The complexity of relational query languages. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, p. 137–146, San Francisco, CA, 1982.

[41] M. Y. Vardi. Querying logical databases. *Journal of Computer and System Sciences*, 33(2):142–160, 1986.

[42] M. Y. Vardi. Fundamentals of dependency theory. In E. Borger, editor, *Trends in Theoretical Computer Science*, p. 171–224. Computer Science Press, Rockville, MD, 1987.

[43] H. Z. Yang and P.-Å. Larson. Query transformation for PSJ-queries. In *Proceedings of the Thirteenth International Conference on Very Large Data Bases*, p. 245–254, Los Altos, CA, 1987.