Authorization Views and Conditional Query Containment

Zheng Zhang and Alberto O. Mendelzon

University of Toronto
Department of Computer Science
{zhzhang, mendel}@cs.toronto.edu

Abstract. A recent proposal for database access control consists of defining "authorization views" that specify the accessible data, and declaring a query valid if it can be completely rewritten using the views. Unlike traditional work in query rewriting using views, the rewritten query needs to be equivalent to the original query only over the set of database states that agree with a given set of materializations for the authorization views. With this motivation, we study conditional query containment, *i.e.* , containment over states that agree on a set of materialized views. We give an algorithm to test conditional containment of conjunctive queries with respect to a set of materialized conjunctive views. We show the problem is Π_2^p-complete. Based on the algorithm, we give a test for a query to be conditionally authorized given a set of materialized authorization views.

1 Introduction

Access control is an integral part of databases and information systems. Traditionally, access control has been achieved by presenting users with a set of views that their queries must operate on. An alternative approach achieves "authorization transparency" [13–16] by using views in a different way. A set of "authorization views" specifies what information a user is allowed to access. The user writes the query in terms of the base relations, and the system tests the query for validity by determining whether it can be completely rewritten using the authorization views. For flexibility, views can be parameterized with information specific to a session, such as the user-id, location, date, time, etc., which are instantiated before access control is performed.

Example 1. Consider a database with the following relations: *Employees(**eid**, name, rank)*, *Projects(**pid**, name, headid)*, *EP(**eid, pid**)*, *Progress(**eid**, **pid**, prgs)*. The *EP* relation associates employees with projects, while a tuple in *Progress* represents a progress report by an employee on a project that the employee is working on. We use this schema as a running example here and in Section 4. The following authorization view V_1 states the policy that an employee can see the progress of his or her colleagues in the projects that the employee is working on.

$$V_1(eid, pid, prgs) \leftarrow Progress(eid, pid, prgs), EP(\$userid, pid).$$

For simplicity, we assume that a user's id is the same as his or her employee-id. The parameter \$user-id is instantiated to the actual user-id before access control is performed. The set of authorization view definitions resulting from instantiating all the parameters that occur in them is called the *instantiated authorization views*. These define exactly what information is accessible to the user in the current session.

Now suppose employee '88' wants to see the progress in all the projects that he or she is associated with, using the following query q.

$$q(eid, pid, prgs) \leftarrow Progress(eid, pid, prgs), EP(88, pid).$$

The instantiated authorization view V_1 is as follows.

$$V_1(eid, pid, prgs) \leftarrow Progress(eid, pid, prgs), EP(88, pid).$$

The following query q' is an equivalent rewriting of q in terms of the instantiated view V_1, showing that q is authorized.

$$q'(eid, pid, prgs) \leftarrow V_1(eid, pid, prgs).$$

Now suppose the same employee wants to see who are the employees who have reported progress in both projects 'XP1' and 'XP2', using the following query q.

$$q(eid) \leftarrow Progress(eid, XP1, prgs_1), Progress(eid, XP2, prgs_2).$$

Since it is the same employee, the instantiated authorization view remains the same. It is easy to see that there is no rewriting q' of q in terms of V_1 such that q' is equivalent to q over all database states; hence, the query will be rejected. But this is unnecessarily harsh. Intuitively, if EP says that employee '88' is working on projects 'XP1' and 'XP2', then q should be authorized. The problem is that the requirement that there be a rewriting q' that is equivalent to q *over all database states* is too strong. For example, the following q' is equivalent to the last query q, not over all database states, but only over those states where employee '88' is working on projects 'XP1' and 'XP2'.

$$q'(eid) \leftarrow V_1(eid, XP1, prgs_1), V_1(eid, XP2, prgs_2).$$

In sum, we adopt the definition of [14]: a query q is *conditionally valid* with respect to a set of views V and a set of materializations of these views MV if there is a rewriting q' of q using the views V such that, for all database states where the values of the views V agree with MV, q agrees with q'.

Note that unconditional authorization (*i.e.* , with equivalence required over all database states) reduces to the well-known problem of whether a query can be rewritten using views [10, 11]. Just as query containment plays a crucial role in the theory of rewriting queries using views, the problem of *conditional query containment* with respect to a set of view materializations must be solved in order to solve the problem of conditional authorization. We study conditional containment of conjunctive queries with respect to a set of materialized conjunctive views. We show that this problem is Π_2^p-complete and use it to give a test for conditional query authorization.

The rest of the paper is organized as follows. Section 2 is the preliminaries. Section 3 presents the test for conditional containment between conjunctive queries. Section 4 presents our solution to conditional query authorization. Section 5 describes related work, and Section 6 concludes the paper and gives directions for future work.

2 Preliminaries

We consider the usual class of conjunctive queries, CQ. The *conjunctive queries with arithmetic comparisons* (CQ^{AC}) extend the conjunctive queries CQ by allowing subgoals with built-in predicates of the form $x_j \theta x_k$, where x_j and x_k are either variables or constants and θ is $\neq, =, <$, or $\leq$. Every variable occurring in an equality or inequality must also occur in a regular subgoal. The predicates used in the regular subgoals are called *EDB* (extensional database) predicates. In particular, denote by $CQ^{\neq}$ the subclass of CQ^{AC} with only disequations ($\neq$).

The *normalization* of a query q in CQ^{AC} creates a new query nq from q in two steps: first replace each occurrence of a shared variable x in the regular subgoals, except the first occurrence, by a new distinct variable x_i and add $x = x_i$ to nq; then replace each constant c by a new distinct variable t, and add $t = c$ to nq.

Given an instance d, a valuation ρ from a query Q into d is a total function ρ from the variables of Q to the domain of constants from d such that $\rho(X_i) \in d(p_i)$ for each regular subgoal $p_i(X_i)$ of Q. The answer to a query Q on instance d is denoted by $Q(d)$ and defined as follows.

$$Q(d) = \{\rho(X) \mid \rho \text{ is a valuation for } Q \text{ into } d, q(X) \text{ is the head predicate of } Q\}.$$

A query Q is *satisfiable* if there exists a database instance d such that $Q(d)$ is nonempty. Unlike the CQ's, which are always satisfiable, a query in CQ^{AC} is unsatisfiable when its equality and inequality subgoals are unsatisfiable.

For any two queries Q_1 and Q_2, Q_1 is said to be *unconditionally contained* in Q_2, denoted by $Q_1 \subseteq Q_2$, if for all database instances d, $Q_1(d) \subseteq Q_2(d)$. Many algorithms exist to test containment of CQ's and their extensions in set theoretical semantics [2, 4, 8, 17]. Among them, the concept of *containment mapping* is widely used. A containment mapping from query Q_2 to query Q_1 is a function from the variables and constants of Q_2 to those of Q_1 that is the identity on constants and that induces a mapping from the subgoals of Q_2 to those of Q_1.

Theorem 1. *[4] For any two CQ's Q_1 and Q_2, $Q_1 \subseteq Q_2$ if and only if there exists a containment mapping ρ from Q_2 to Q_1 such that ρ maps the head of Q_2 to the head of Q_1.*

This theorem remains true when Q_1 contains built-in predicates [10]. We say a query Q_r is a *complete rewriting* of Q using V if Q_r is written using only the views in V and is equivalent to Q. The paper just cited also gives an algorithm to determine whether a query Q has a complete rewriting in a set of views V.

3 Conditional Query Containment

This section presents our solution to the conditional query containment problem. We assume a fixed set of view definitions $V = \{v_1, \ldots v_n\}$. For each view v_i, we are given an instance of it called mv_i. The set of all materialized view instances is called MV. The set of database instances $D = \{d \mid v_j(d) = mv_j, 1 \leq j \leq n\}$ is called the set of *valid instances* for V and MV. Note that it is possible for the set of materializations to be inconsistent, *i.e.* , the set D can be empty. Although our methods work in this case also, we do not treat it explicitly in this paper for lack of space.

The default variable set is $\{x, y, z, \ldots\}$ while $\{X, Y, Z, \ldots\}$ are tuples of variables and constants. We only consider queries and views in CQ (*i.e.* , no arithmetic comparisons) in the following sections, unless otherwise noted.

Definition 1. *For any two queries Q_1 and Q_2, Q_1 is said to be conditionally contained in Q_2 w.r.t. V and MV, denoted by $Q_1 \subseteq_{V,MV} Q_2$, if for every d in D, $Q_1(d) \subseteq Q_2(d)$. Q_1 is said to be conditionally equivalent to Q_2 w.r.t. V and MV, denoted by $Q_1 \equiv_{V,MV} Q_2$, if $Q_1 \subseteq_{V,MV} Q_2$ and $Q_2 \subseteq_{V,MV} Q_1$.*

Definition 2. *A query Q is conditionally empty w.r.t. V and MV if $Q(d)$ is empty for every d in D.*

3.1 A Necessary Condition

Given two CQ's Q_1 and Q_2 such that $Q_1 \subseteq Q_2$, by Theorem 1, the set of EDB predicates appearing in Q_2 must be contained in the set of those appearing in Q_1. This, however, may not be the case for conditional containment.

Example 2. Given a view $v(x) \leftarrow r_2(x)$ and two queries $Q_1 : q_1() \leftarrow r_1(x), r_2(x)$, $Q_2 : q_2() \leftarrow r_2(x), r_3(x)$. If mv is empty, Q_1 and Q_2 are conditionally empty. Hence, $Q_1 \equiv_{v,mv} Q_2$ even though their sets of EDB predicates do not contain each other.

If mv is nonempty, query results depend on r_1 and r_3, respectively. The materialized view does not have r_1 and r_3 in its body, so there is no conditional containment relationship between the two queries.

Theorem 2. *If $Q_1 \subseteq_{V,MV} Q_2$, then either Q_1 is conditionally empty w.r.t. V and MV, or the set of EDB predicates of Q_2 is contained in the set of EDB predicates appearing either in Q_1 or in the definition of some view whose materialization is nonempty.*

Thus, we know that if Q_1 is not conditionally empty, and the set of EDB predicates of Q_2 is not contained in the set of EDB predicates appearing in Q_1 or in the definition of one or more nonempty materialized views, then we can conclude $Q_1 \not\subseteq_{V,MV} Q_2$. We will discuss testing conditional emptiness in Section 3.5. From now on, we assume that the condition of the Theorem holds.

Our plan for testing conditional containment is as follows. First, for any conjunctive query Q we shall construct a query Q' that has the property that

Q' agrees with Q on the valid instances and is empty on the invalid ones. Given two CQ's Q_1 and Q_2, it will follow that $Q_1 \subseteq_{V,MV} Q_2$ if and only if $Q'_1 \subseteq Q_2$. That is, we have transformed the problem of conditional containment to one of standard, unconditional containment. Unfortunately, we are not done yet, because Q'_1 is not a conjunctive query, or even a union of queries in CQ^{AC}; it is a nonrecursive Datalog program with negation. The second step therefore is to transform Q'_1 into a query Q''_1 that is a union of queries in CQ^{AC} and still has the property that $Q_1 \subseteq_{V,MV} Q_2$ if and only if $Q''_1 \subseteq Q_2$.

3.2 Construction of Q'

For each view v_i and materialization mv_i, such that mv_i is not empty, we create a set of subgoals P_i that we will add to the body of Q. Abusing notation slightly, we say that P_i is true on instance d when there is a valuation that embeds P_i in d. Intuitively, P_i is true on instance d if and only if every tuple in mv_i is in $v_i(d)$. Suppose that view v_i is given by

$$v_i(x_1, x_2, \ldots, x_{s_i}) \ \leftarrow \ r_s(\ldots, x_1, \ldots), \ldots, r_t(\ldots, x_2, \ldots), \ldots$$

and its nonempty materialization mv_i consists of tuples: $t_1 = (x^1_1, x^1_2, \ldots, x^1_{s_i}), \ldots,$ $t_{K_i} = (x^{K_i}_1, x^{K_i}_2, \ldots, x^{K_i}_{s_i})$ where $s_i \geq 0$ is the size of the head predicate of v_i and $K_i \geq 1$ is the cardinality of mv_i. For $(1 \leq j \leq K_i)$, let

$$c_{i_j} = \{r_s(\ldots, x_1, \ldots), \ldots, r_t(\ldots, x_2, \ldots), \ldots, x_1 = x^j_1, x_2 = x^j_2, \ldots, x_{s_i} = x^j_{s_i}\}.$$

Rename the variables in each c_{i_j} so that they are disjoint from every other c_{i_j} and also disjoint from those in Q. Let $P_i = \bigcup_{j=1}^{K_i} c_{i_j}$.

Lemma 1. *There is a valuation from P_i into d if and only if $mv_i \subseteq v_i(d)$.*

In addition to the P_i's, we define a set of negated subgoals called N_i's, one for each view v_i. Each N_i is the negation of a subgoal $c_i()$, where c_i is a new intensional predicate. Intuitively, $c_i()$ will be true (nonempty) on instance d if and only if $v_i(d)$ contains some tuple not in mv_i; so that N_i will be true on instance d if and only if $v_i(d) \subseteq mv_i$. The rule that defines c_i is the following.

$$\begin{aligned} c_i() \leftarrow\ & r_s(\ldots, x_1, \ldots), \ldots, r_t(\ldots, x_2, \ldots), \ldots, \\ & \bigwedge_{k=1}^{K_i} \neg(x_1 = x^k_1, x_2 = x^k_2, \ldots, x_{s_i} = x^k_{s_i}). \quad (*) \end{aligned}$$

Note that the rule for c_i is expressed for convenience with a negated conjunction of subgoals in the body; this is a shorthand for the union c_i of all the rules whose body contains one disequation from each of the negated subgoals.

Lemma 2. *N_i is true on instance d if and only if $v_i(d) \subseteq mv_i$.*

Now we can rewrite Q as a nonrecursive Datalog program by adding all the P_i and N_i subgoals to its body, and attaching the rules that define the c_i's.

$$\begin{aligned} Q' : q(X) \leftarrow\ & p_1(X_1), p_2(X_2), \ldots, p_n(X_n), P_1, \ldots, P_m, \\ & N_1, \ldots, N_m, N_{m+1}, \ldots, N_{m'}. \\ c_i() \leftarrow\ & r_s(\ldots, x_1, \ldots), \ldots, r_t(\ldots, x_2, \ldots), \ldots, \\ & \bigwedge_{k=1}^{K_i} \neg(x_1 = x_1^k, x_2 = x_2^k, \ldots, x_{s_i} = x_{s_i}^k). \end{aligned}$$

where m' is the number of views and m is the number of views with nonempty materializations. Q' has the following properties.

Lemma 3. *$Q'(d) = Q(d)$ for all valid database instances d, and $Q'(d') = \emptyset$ for all invalid database instances d'.*

Example 3. Consider three views $v_1() \leftarrow r(x)$ with mv_1 containing just the one tuple () (*i.e.* , true), $v_2(x) \leftarrow s(x)$ with mv_2 containing just the tuple (e), and $v_3(x) \leftarrow t(x)$ with empty materialization, as well as a CQ $Q : q(x) \leftarrow r(x)$. The Datalog program is

$$\begin{aligned} Q' : q(x) &\leftarrow r(x), r(x_1), s(e), \neg c_2(), \neg c_3(). \\ c_2() &\leftarrow s(x), x \neq e. \\ c_3() &\leftarrow t(x). \end{aligned}$$

Proposition 1. *Given two CQ's Q_1 and Q_2 as well as a set of conjunctive views V with materializations MV, $Q_1' \subseteq Q_2$ if and only if $Q_1 \subseteq_{V,MV} Q_2$.*

Proof. (only if) Suppose $Q_1' \subseteq Q_2$. Let d be a valid database instance. Then $Q_1'(d) = Q1(d)$ by Lemma 3. Therefore, $Q1(d) \subseteq Q_2(d)$.

(if) Suppose $Q_1 \subseteq_{V,MV} Q_2$. Let d be any database instance. If d is valid, from $Q_1(d) \subseteq Q_2(d)$, it follows that $Q_1'(d) \subseteq Q_2(d)$. If d is not valid, $Q_1'(d) = \emptyset$, so $Q_1'(d) \subseteq Q_2(d)$. Therefore, $Q_1' \subseteq Q_2$. □

3.3 Construction of Q''

With the above proposition, we are on our way to transform the conditional containment problem into an unconditional problem. We would like to eliminate the c_i's and replace the corresponding N_i's to create a CQ or one of its extensions. Consider a c_i whose mv_i is nonempty. Query (*) is equivalent to a union c_i of queries c_{ik} in $CQ^{\neq}$ which share the same regular subgoals. Given a database instance e, $N_i = \mathit{true}$ for e means there is no valuation over the regular subgoals of c_i into e such that $(x_1, x_2, \ldots, x_{s_i})$ is mapped to a tuple not in mv_i. We will relax this restriction by replacing N_i with N_i', where $N_i' = \mathit{true}$ for e means that, if d is any sub-instance obtained by some valuation ρ of the regular subgoals in Q' over e, then there is no valuation over the regular subgoals of c_i

into d such that $(x_1, x_2, \ldots, x_{s_i})$ is mapped to a tuple not in mv_i. To obtain N_i', we first normalize each query c_{ik} in c_i (see Section 2) and obtain a rule whose body has three parts: the regular subgoals nc_{ik}^+, a set of equalities Eq_{ik}, and a set of negations Neq_{ik} in c_{ik}. Since c_{ik}'s share the regular subgoals, nc_{ik}^+'s are the same, denoted by nc_i^+. So are Eq_{ik}'s, denoted by Eq_i. Clearly, $\bigcup_k Neq_{ik}$ is equivalent to the disequations in c_i,

$$Neq_i = \bigwedge_{k=1}^{K_i} \neg(x_1 = x_1^k, x_2 = x_2^k, \ldots, x_{s_i} = x_{s_i}^k).$$

Consider all the containment mappings $\{mp_{i1}, mp_{i2}, \ldots, mp_{ig}\}$ from nc_i^+ to Q'. Since we are assuming that mv_i is nonempty, $\bigcup_k Neq_{ik}$ is nonempty. Let N_i' be:

$$\bigwedge_{j=1}^{g} \neg mp_{ij}(Eq_i) \vee \neg mp_{ij}(\bigcup_k Neq_{ik}).$$

Notice all subgoals in $mp_{ij}(nc_i^+)$ exist in Q', hence they are redundant and omitted here. Furthermore, $\bigcup_k Neq_{ik}$ is just Neq_i, hence $\neg mp_{ij}(\bigcup_k Neq_{ik})$ can be simplified to $\neg mp_{ij}(Neq_i)$ which is equivalent to its positive form, say $mp_{ij}(Peq_i)$,

$$\bigvee_{l=1}^{K_i} (mp_{ij}(x_1) = x_1^l, mp_{ij}(x_2) = x_2^l, \ldots, mp_{ij}(x_{s_i}) = x_{s_i}^l).$$

Example 4. Given a view $v(x) \leftarrow r_1(x, y, y)$ with mv containing only the tuple (1) and a query $Q : q(x, y) \leftarrow r_1(x, y, z), r_2(z)$, the Datalog program Q' is

$$\begin{aligned} Q' : q(x, y) &\leftarrow r_1(x, y, z), r_2(z), r_1(1, y_1, y_1), \neg c_1, \\ c_1() &\leftarrow r1(x_2, y_2, y_2), x_2 \neq 1. \end{aligned}$$

The normalization of c_1 is $nc_1 \leftarrow r1(x_2, y_2, y_3), y_2 = y_3, x_2 \neq 1$. There are two containment mappings from nc_1^+ to Q':

$$\{mp_{11} : \ r1(x_2, y_2, y_3) \rightarrow r_1(x, y, z), \quad mp_{12} : \ r1(x_2, y_2, y_3) \rightarrow r_1(1, y_1, y_1)\}.$$

So $\neg mp_{11}(y_2 = y_3)$ is $(y \neq z)$, $mp_{11}(x_2 = 1)$ is $(x = 1)$ and $\neg mp_{12}(y_2 = y_3)$ is $(y_1 \neq y_1)$, $mp_{12}(x_2 = 1)$ is $(1 = 1)$. Thus, $(y \neq z \vee x = 1) \wedge (y_1 \neq y_1 \vee 1 = 1)$ can replace the $\neg c_1$ in Q'. Thus, Q'' is a union of queries in $CQ^{\neq}$.

$$\begin{aligned} Q'' : q(x, y) &\leftarrow r_1(x, y, z), r_2(z), r_1(1, y_1, y_1), y \neq z. \\ q(1, y) &\leftarrow r_1(1, y, z), r_2(z), r_1(1, y_1, y_1). \end{aligned}$$

We would also like to replace N_i's with empty mv_i in a similar fashion. Since in this case there is no P_i in Q', we cannot guarantee that every subgoal in nc_i^+ can be mapped to a regular subgoal in Q'. Consider all the containment mappings $\{mp_{i1}, mp_{i2}, \ldots, mp_{ig}\}$ from nc_i^+ to Q', where if a subgoal $p(X)$ in

nc_i^+ cannot be mapped to a subgoal in Q' (*i.e.* , the EDB predicate p does not appear in Q'), then $p(X)$ is mapped to itself. We define N_i' in this case as:

$$\bigwedge_{j=1}^{g} \neg mp_{ij}(nc_i^+) \vee \neg mp_{ij}(Eq_i).$$

If every EDB predicate in v_i appears in Q or in some view with a nonempty materialization, then for a containment mapping mp_{ij}, all subgoals in $mp_{ij}(nc_i^+)$ are in Q', and if Eq_i is nonempty, we can replace N_i by $\bigwedge_{j=1}^{g} \neg mp_{ij}(Eq_i)$. If Eq_i is empty, we can conclude that Q is conditionally empty with respect to V and MV (See Section 3.5). For all other cases, we simply delete N_i from Q'.

In sum, we rewrite Q' into the following:

$$\begin{aligned} Q'' : q(X) \;\; \leftarrow \;\; & p_1(X_1), \ldots, p_n(X_n), P_1, \ldots, P_m, \\ & \bigwedge_{k,j} \neg mp_{kj}(Eq_k), \bigwedge_{i,j} \neg mp_{ij}(Eq_i) \vee mp_{ij}(Peq_i) \end{aligned}$$

where each P_i and $\bigwedge_j \neg mp_{ij}(Eq_i) \vee mp_{ij}(Peq_i)$ represent a view v_i with nonempty materialization, and each $\bigwedge_j \neg mp_{kj}(Eq_k)$ represents a view v_k whose EDB predicates appear in Q or in the views with nonempty materializations, and whose mv_k is empty. Q'' is equivalent to a union of queries in $CQ^{\neq}$. Let the view definitions be fixed. The number of queries in the union is exponential in the sizes of the query and the view materializations. Notice that Example 3 covers all possible cases in the construction of Q'; Example 4 shows how to replace N_i when mv_i is nonempty in the construction of Q''. Before we show more examples to cover different cases when mv_i is empty in the construction of Q'', we state that Q'' has the following properties.

Lemma 4. *Given a CQ Q and a set of conjunctive views V with materializations MV, let ρ be a valuation of Q'' on any input database instance. Then the set $\{\rho(p(X)) \mid p(X)$ is a regular subgoal of $Q''\}$ is a valid database instance.*

Lemma 5. *Given a CQ Q and a set of conjunctive views V with materializations MV, $Q(d) = Q'(d) = Q''(d)$ for all valid database instances d.*

Lemma 6. *Given a CQ Q and a set of conjunctive views V with materializations MV, $Q'(d) \subseteq Q''(d)$ for all database instances d.*

Theorem 3. *Let Q_1 and Q_2 be two CQ's, V be a set of conjunctive views with materializations MV. $Q_1'' \subseteq Q_2$ if and only if $Q_1 \subseteq_{V,MV} Q_2$.*

Example 5. Given two queries $Q_1 : q_1(x) \leftarrow r(x), s(y)$; $Q_2 : q_2(x) \leftarrow r(x)$ and a view $v() \leftarrow r(x), s(x)$ with no tuple. The set of EDB predicates of Q_2 is contained in the set of Q_1's EDB predicates. All EDB predicates in v appear in Q_1. Therefore, $Q_1'' : q_1(x) \leftarrow r(x), s(y), x \neq y$. Clearly, Q_1'' is unconditionally contained in Q_2, which implies $Q_1 \subseteq_{V,MV} Q_2$. If the view is $v() \leftarrow r(x)$ with no tuple, then its Eq is empty and all EDB predicates in v appear in Q_1. Thus, Q_1 is conditionally empty with respect to V and MV.

Example 6. Given two queries $Q_1 : q_1(x) \leftarrow s(x), t(x);\; Q_2 : q_2(x) \leftarrow s(x)$ and a view $v() \leftarrow r(x)$ with no tuple. View v has no effect over the containment relationship between Q_1 and Q_2. $Q_1'' : q_1(x) \leftarrow s(x), t(x)$ which is unconditionally contained in Q_2. Thus, $Q_1 \subseteq_{V,MV} Q_2$.

Example 7. Given two queries $Q_1 : q_1(x) \leftarrow s(x), t(x);\; Q_2 : q_2(x) \leftarrow s(x)$ and a view $v() \leftarrow r(x), t(x)$ with no tuple. The EDB predicate r does not appear in Q_1 and Q_2. Q_1'' is Q_1. For any valuation ρ of Q_1'', the valuation of the regular subgoals in Q_1'' is a valid database instance since the empty r makes mv empty. $Q_1'' \subseteq Q_2$ implies $Q_1 \subseteq_{V,MV} Q_2$.

The construction of Q'', and Theorem 3, can in fact be generalized to the case when Q_1 and Q_2 are unions of queries in $CQ^{\neq}$. First, consider the case when Q is in $CQ^{\neq}$. Then Q' and Q'' are constructed as before, *i.e.* , we leave the disequations of Q untouched. Notice a valuation of Q'' satisfies all the equality and inequality subgoals in Q.

When Q is a union of queries in $CQ^{\neq}$, let q be one of them in the union. We can create q' and q'' as above. Then Q'' is a union of such q'''s. In particular, consider Q'' for some CQ Q. Q'' is a union of queries in $CQ^{\neq}$. Since the views and their materializations are unchanged, the P_i's in $(Q'')'$ are the same as the P_i's in Q'. So are N_i's in $(Q'')'$ and Q'. Since Q'' contains P_i's of Q' as its subgoals, adding another set of P_i's does not change the semantics of Q''. Thus, the P_i's in $(Q'')'$ can be deleted. Next, we would like to replace the N_i's in $(Q'')'$ to create $(Q'')''$. Since Q' shares the regular subgoals with Q'', which has the same regular subgoals of $(Q'')'$ (after deleting the extra P_i's), the containment mappings from nc_i^+'s to Q' are the same as those from nc_i^+'s to $(Q'')'$. Thus, the replacement of N_i's in Q' is the same as the one for $(Q'')'$ (Notice all the equations and disequations only depend on the view definitions and materializations, which are not changed). Hence $(Q'')''$ is Q''. We conclude that the Q'' construction can be generalized to unions of queries in $CQ^{\neq}$.

Theorem 4. *Let Q_1 and Q_2 be two unions of queries in $CQ^{\neq}$, V be a set of conjunctive views with materializations MV. $Q_1'' \subseteq Q_2$ if and only if $Q_1 \subseteq_{V,MV} Q_2$.*

3.4 Complexity of Conditional Query Containment

For standard containment, complexity is given as a function of query size. However, for conditional containment, the sizes of the queries, the view definitions, and the view materializations are all important factors on the problem's complexity. In our analysis, we chose to assume that the view definitions change much more slowly than the underlying database instance and the user queries, so the view definitions are fixed and we measured the combined complexity as a function of query size and materialization size.

Let Q_1 and Q_2 be two CQ's, V be a set of conjunctive views with materializations MV. As described in the previous sections, we can construct a query

Q_1'', a union of queries in $CQ^{\neq}$, such that $Q_1'' \subseteq Q_2$ if and only if $Q_1 \subseteq_{V,MV} Q_2$. This idea provides the following upper bound on the complexity of conditional query containment.

Theorem 5. *Let Q_1 and Q_2 be two CQ's, V be a set of conjunctive views with materializations MV. Determining whether $Q_1 \subseteq_{V,MV} Q_2$ is in Π_2^p.*

Proof (Sketch). By Theorem 3, $Q_1'' \subseteq Q_2$ if and only if $Q_1 \subseteq_{V,MV} Q_2$. Thus, if for all queries q in the union Q_1'', there exists a containment mapping from Q_2 to q such that the head of Q_2 is mapped to the head of q, then $Q_1 \subseteq_{V,MV} Q_2$. Let g_i be the number of containment mappings from nc_i^+ to Q_1'. Since the sizes of views are constants, g_i is polynomial in the sizes of Q_1 and the view materializations. The number of equality and inequality subgoals in q is the sum of all g_i's. Therefore, the size of q is polynomial in the sizes of Q_1 and the view materializations. The size of a containment mapping from Q_2 to q is polynomial in the sizes of the queries and the view materializations. Thus, the complexity is Π_2^p. □

Theorem 6. *Let Q_1 and Q_2 be two CQ's, V be a set of conjunctive views with materializations MV. Determining whether $Q_1 \subseteq_{V,MV} Q_2$ is Π_2^p-hard.*

Proof (Sketch). To reduce from the $\forall\exists$-CNF problem to our problem, we use a similar construction to the one in [12]. Our construction is slightly modified from the one in the paper just cited, because we assume the queries and the materializations can vary and the view definitions are fixed while Millstein *et al.* assumed the queries and the view definitions can vary. □

The above two theorems show that the problem is Π_2^p-complete when the queries and the materializations can vary. In comparison, the certain answer containment problem of [12] is Π_2^p-complete when the queries and the view definitions can vary. In terms of unconditional query containment where the input consists of just the two queries, determining the containment between two $CQ^{\neq}$'s is also Π_2^p-complete [18]. In contrast, unconditional CQ containment is NP-complete [4] in terms of the sizes of the input queries.

3.5 Testing Conditional Emptiness

So far we have assumed Q_1 and Q_2 are not conditionally empty queries and the set of EDB predicates of Q_2 is contained in the set of EDB predicates appearing either in Q_1 or in the definition of some view whose materialization is nonempty. Thus, we need to determine whether a query Q is conditionally empty with respect to a set of conjunctive views V with materializations MV. We first create Q'' from Q as before. From Lemma 4, we get the following result.

Proposition 2. *A CQ Q is conditionally empty w.r.t. a set of conjunctive views V with materializations MV if and only if Q'' is unsatisfiable.*

Example 8. Given $v_1(x) \leftarrow r_2(x)$ with one tuple (2) and $v_2(x) \leftarrow r_4(x)$ with one tuple (4); two queries $Q_1 : q_1(x) \leftarrow r_1(x), r_2(x), r_4(x)$ and $Q_2 : q_2(x) \leftarrow r_2(x), r_3(x), r_4(x)$. Note that the set of EDB predicates of Q_2 is not contained in the set of the EDB predicates appearing in Q_1 or the views with nonempty materializations, and similarly for Q_1. By Theorem 2, if the two queries are nonempty with respect to V and MV, then there is no conditional containment relationship between the two queries. To check if the queries are conditionally empty, we construct Q_1'' and Q_2'':

$$Q_1'' : q_1(x) \ \leftarrow \ r_1(x), r_2(x), r_4(x), r_2(2), r_4(4), x = 2, x = 4.$$

Clearly, Q_1'' is not satisfiable. Similarly, Q_2'' is unsatisfiable:

$$Q_2'' : q_2(x) \ \leftarrow \ r_2(x), r_3(x), r_4(x), r_2(2), r_4(4), x = 2, x = 4.$$

Therefore, we conclude the two queries are conditionally empty.

Theorem 7. *Given a CQ Q and a set of conjunctive views V with materializations MV, checking whether Q is conditionally empty w.r.t. V and MV is $coNP$-complete.*

Proof. Checking whether Q is conditionally empty with respect to V and MV is equivalent to checking whether Q'' is unsatisfiable. Satisfiability of Q'' can be checked in time polynomial in the sizes of V and MV by guessing a valuation for one of the disjuncts in Q'' and checking it is satisfied by that valuation. Therefore, the problem of checking whether Q is conditionally empty with respect to V and MV is in *coNP*.

The *coNP*-hardness is obtained by adapting the following result of Abiteboul and Duschka [1]. Let V be a set of conjunctive views with materializations MV, checking whether there exists a database instance d such that $MV = V(d)$ is NP-hard. We assume that MV is not empty, since when MV is empty, there is trivially an instance I (the empty instance) such that $V(I) = MV$. We reduce the complement of this problem to our problem.

Since MV is nonempty, there exists some nonempty mv_i. Let $r(X)$ be a subgoal in v_i. Define a CQ $Q : q() \ \leftarrow \ r(X)$. If for all database instances d, $MV \neq V(d)$, then Q'' is unsatisfiable. Otherwise, by Lemma 4, there exists a d such that $MV = V(d)$. Therefore, by Proposition 2, Q is conditionally empty with respect to V and MV.

If Q is conditionally empty with respect to V and MV, there does not exist a database instance d such that $MV = V(d)$. Otherwise, since mv_i is nonempty, $Q(d)$ is nonempty for the valid database instance d.

Thus, checking conditionally emptiness is also *coNP*-hard. □

4 Conditional Authorization

In the Introduction, we discussed parameterized authorization views. Given a user query, our approach always first instantiates the parameterized views using the parameter values associated with the user and the session, before we determine whether a query should be conditionally authorized. Thus, we can assume in this section that the views have already been instantiated. First, we define conditional authorization.

Definition 3. *A query Q is conditionally authorized w.r.t. authorization views V with materializations MV, if there is a query Q_r that is written using only the views in V, and that is conditionally equivalent to Q.*

We have shown how to construct Q'' for a CQ Q. We know Q is conditionally empty if and only if Q'' is unsatisfiable. If Q is conditionally empty, there are many complete rewritings of Q'' using views V. Therefore, Q should be authorized.

When Q is not conditionally empty, we have shown that $Q(d) = Q''(d)$ for all valid database instances d. Therefore, if there is a complete rewriting of Q'' using V, the query Q is conditionally authorized. We would like to show that if there is no complete rewriting of Q'' using V, then Q is not conditionally authorized. Suppose Q'' does not have a complete rewriting and Q is still conditionally authorized. Then there exists a query Q_1 that is conditionally equivalent to Q and that can be rewritten using only the views in V. Let us abuse notation and call Q_1 also the query obtained by expanding the view subgoals in this rewriting.

Lemma 7. *1. Every EDB predicate of Q'' occurs in Q_1 or in the definition of some view with nonempty materialization.*
2. Every EDB predicate of Q_1'' occurs in Q'' or in the definition of some view with nonempty materialization.

Since $Q(d) = Q''(d)$ for all valid database instances d, $Q_1 \equiv_{V,MV} Q''$. Thus, by the above lemma and Theorem 4, $Q_1 \subseteq_{V,MV} Q''$ implies $Q_1'' \subseteq Q''$. On the other hand, we know $Q_1'' \equiv_{V,MV} Q_1 \equiv_{V,MV} Q \equiv_{V,MV} Q''$. Since $(Q'')''$ is still Q'', $Q'' \subseteq_{V,MV} Q_1''$ implies $Q'' \subseteq Q_1''$. Then, we have $Q'' \equiv Q_1''$. However, we know Q_1'' is completely rewritable in V, yet Q'' does not have a complete rewriting using V, which is a contradiction.

Theorem 8. *Let Q be a CQ and V be a set of conjunctive views with materializations MV. Q is conditionally authorized if and only if there is a complete rewriting of Q'' using V.*

Similar to the discussion of Theorem 3, the above theorem also applies when Q is a union of queries in $CQ^{\neq}$. Since the query contains inequalities while the views are conjunctive, the algorithm in [10] can be used here to check whether Q'' has a complete rewriting in V.

In the paper just cited, the algorithm depends on a bound for the number of view literals that need to appear in a complete rewriting. The same bound applies for conditional complete rewritings.

Corollary 1. *Let Q be a CQ with n subgoals, and V be a set of conjunctive views with materializations MV. If there is a query Q_r that is written using only the views in V, and that is conditionally equivalent to Q, then it has such a rewriting with at most n subgoals.*

Example 9 (Example 1 continued). Recall the query q is

$$q(eid) \leftarrow Progress(eid, \mathit{XP1}, prgs_1), Progress(eid, \mathit{XP2}, prgs_2),$$

and the instantiated authorization view is

$$V_1(eid, pid, prgs) \leftarrow Progress(eid, pid, prgs), EP(88, pid).$$

We consider the following four cases of the materialization of the instantiated authorization view.

1. MV_1 is not empty, and in the current database state, employee '88' is working on projects 'XP1' and 'XP2', and some other employee has reported progress for both projects. Let MV_1 be $\{\ (99, \mathit{XP1}, P_1), (99, \mathit{XP2}, P_2)\ \}$.

$$\begin{aligned} q''(99) \leftarrow\ & Progress(99, \mathit{XP1}, P_1), Progress(99, \mathit{XP2}, P_2), \\ & Progress(99, \mathit{XP1}, P_1), EP(88, \mathit{XP1}), \\ & Progress(99, \mathit{XP2}, P_2), EP(88, \mathit{XP2}). \end{aligned}$$

   Thus, the complete rewriting is $q''(99) \leftarrow V_1(99, \mathit{XP1}, P_1), V_1(99, \mathit{XP2}, P_2)$. Therefore, we authorize this query q.
2. MV_1 is not empty, and in the current database state, employee '88' is not working on both projects, say only on project 'XP2', and some other employee has reported progress for 'XP2'. Let MV_1 be $\{(99, \mathit{XP2}, P_2)\}$.

$$\begin{aligned} q''(eid) \leftarrow\ & Progress(eid, \mathit{XP1}, prgs_1), Progress(99, \mathit{XP2}, P_2), \\ & Progress(99, \mathit{XP2}, P_2), EP(88, \mathit{XP2}). \end{aligned}$$

   There is no containment mapping from the body of V_1 to the body of q'' such that $Progress(eid, \mathit{XP1}, prgs_1)$ is the image of $Progress(eid, pid, prgs)$, since that requires $EP(88, \mathit{XP1})$ to occur in q''. There is no complete rewriting of q'' using V_1. Therefore, we reject the query q. In fact, the materialization can contain other information, such as employee '88' works on project 'XP3' and there is a report for 'XP3' from some employee. As long as the materialization does not contain $(99, \mathit{XP1}, P_1)$, the above analysis applies. Similarly, when employee '88' does not work on any of the two projects and MV_1 is not empty, the query should be rejected.
3. MV_1 is empty, but there is one more authorization view that allows any user to know who is working on which project: $V_2(eid, pid) \leftarrow EP(eid, pid)$. Suppose the materialization of V_2 is $(88, \mathit{XP1}), (88, \mathit{XP2})$.

$$\begin{aligned} q''(eid) \leftarrow\ & Progress(eid, \mathit{XP1}, prgs_1), Progress(eid, \mathit{XP2}, prgs_2), \\ & EP(88, \mathit{XP1}), EP(88, \mathit{XP2}), \mathit{XP1} \neq \mathit{XP1}, \mathit{XP2} \neq \mathit{XP2}. \end{aligned}$$

   This is a conditionally empty query, hence, we accept it.
4. MV_1 is empty. From the information of other materialized authorization views, employee '88' cannot infer that he or she is associated with projects

'XP1' and 'XP2'. There are no containment mappings to show $EP(88, \mathit{XP1})$ and $EP(88, \mathit{XP2})$ exist in all valid database states. Suppose there are no other authorization views. Then we have

$$q''(eid) \leftarrow \mathit{Progress}(eid, \mathit{XP1}, prgs_1), \mathit{Progress}(eid, \mathit{XP2}, prgs_2).$$

There is no complete rewriting of q'' using V_1 since there is no containment mapping from the view to q''. Therefore, we reject the query q.

5 Related Work

Chaudhuri *et al.* [5] considered the problem of optimizing queries in the presence of materialized views. They gave an incomplete set of query rewriting rules that generate conditionally equivalent queries under bag semantics. Rizvi *et al.* [14] gave an incomplete set of inference rules for conditional authorization of SQL queries using bag semantics.

Millstein *et al.* introduced the notion of *certain answer containment* with respect to a set of global views in a Data Integration System [12]. Our setting can be viewed as a Data Integration System with base relations as the global schema and authorization views as the local sources, using Local-As-View semantics [9]. However, our notion of conditional containment is different from Millstein *et al.*'s, which is based on the set of certain answers of one query being contained in the set of certain answers of the other one.

Instead of defining authorized queries in terms of rewritings, we could use Calvanese *et al.*'s notion of *lossless query* [3] and say a query is authorized if it is lossless with respect to the views V and their materializations MV, that is, for any two valid instances d and e with respect to V and MV, $Q(d) = Q(e)$. Existence of rewritings is a special case of this. Losslessness has been studied for regular path queries and materialized regular views in [3].

An alternative approach to solve the problem of conditional query containment could be to reduce it to the problem of deciding containment under a set of embedded or disjunctive dependencies, which is decidable under disjunctive chase [7]. Similarly, conditional query authorization could be solved as rewriting a query using views in the presence of embedded or disjunctive dependencies [6].

6 Conclusions and Future Work

We studied the problem of conditional query authorization. We showed that conditional query containment plays a crucial role in it and proposed an algorithm to test conditional containment for unions of queries in $CQ^{\neq}$. Then, we solved the problem of conditional authorization for a conjunctive query with respect to a set of conjunctive authorization views with materializations.

Given the high complexity of the conditional containment and authorization problems, we need to study heuristics or tractable classes of queries and views. For applying our results to the SQL setting, we would also like to solve conditional query authorization under bag semantics.

Acknowledgements

We thank the Natural Sciences and Engineering Research Council of Canada and the Institute for Robotics and Intelligent Systems for their support, and the anonymous reviewers for their careful comments.

References

1. S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. ACM PODS*, pages 254–263, 1998.
2. A. Aho, Y. Sagiv, and J. D. Ullman. Equivalence of relational expressions. *SIAM Journal of Computing*, (8)2:218–246, 1979.
3. D. Calvanese, D. G. Giuseppe, M. Lenzerini, and M. Y. Vardi. Lossless regular views. In *Proc. ACM PODS*, pages 247–258, 2002.
4. A. K. Chandra and P. M. Merlin. Optimal implementations of conjunctive queries in relational databases. In *Proc. STOC*, pages 77–90, 1977.
5. S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. ICDE*, pages 190–200, 1995.
6. A. Deutsch and V. Tannen. Reformulation of xml queries and constraints. In *Proc. ICDT*, pages 225–241, 2003.
7. G. Grahne and A. Mendelzon. Tableau techniques for querying information sources through global schema. In *Proc. ICDT*, pages 332–347, 1999.
8. A. Klug. On conjunctive queries containing inequalities. *Journal of the Association for Computing Machinery*, 35(1):146–160, 1998.
9. M. Lenzerini. Data integration: a theoretical perspective. In *Proc. ACM PODS*, pages 233–246, 2002.
10. A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. ACM PODS*, pages 95–104, 1995.
11. A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. VLDB*, pages 251–262, 1996.
12. T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, pages 67–75, 2002.
13. A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proc. ICDE*, pages 339–347, 1989.
14. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. ACM SIGMOD*, pages 551–562, 2004.
15. A. Rosenthal and E. Sciore. View security as the basis for data warehouse security. In *Intl. Workshop on Design and Management of Data Warehouses*, 2000.
16. A. Rosenthal, E. Sciore, and V. Doshi. Security administration for federations, warehouses, and other derived data. In *IFIP WG11.3 Conf. on Database Security*, 1999.
17. Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operations. *Journal of the ACM*, 27(4):633–655, 1980.
18. Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains (extended version). In *Proc. ACM PODS*, pages 331–345, 1992.