QUERY REWRITING USING VIEWS IN THE PRESENCE OF FUNCTIONAL AND INCLUSION DEPENDENCIES

Jarek Gryz

Department of Computer Science
York University, Toronto, Canada

September 22, 1999

Abstract — Query rewriting using views is a technique for determining how a query may be answered using a given set of resources, which may include materialized views, cached results of previous queries, or queries answerable by other databases. The power of query rewriting can be considerably enhanced by taking into account integrity constraints that are known to hold on base relations. This paper describes an extension of query rewriting that utilizes inclusion dependencies to find rewritings of queries that would otherwise be overlooked. We describe a complete strategy for finding rewritings in the presence of inclusion dependencies and present a basic algorithm that implements that strategy. We also describe extensions to this algorithm when both inclusion and functional dependencies are considered.

1. INTRODUCTION

Query rewriting using views (QRV), also known as query folding [17], is the process of determining whether and how a query can be answered using a given set of resources. These resources can be materialized views, cached results of previous queries, or even queries answerable by other databases. QRV is important because the relations referred to in a query might be stored remotely and hence too expensive to access, or might not be available for access because of network disconnection, or might be conceptual relations only and hence not exist physically. QRV has obvious applications in centralized databases. For example, data warehouses maintain materialized views, and a query can be answered by accessing views instead of base relations if the query can be folded using the views [4]. In multiple query answering, the result of one query can be used to at least partially answer another query. QRV is even more important in a distributed environment. In client-server applications, queries might be cached at the client site. Client queries can be answered more efficiently if the cached data [11, 6] is reused.

Example 1 Consider a patient record database **DB** that includes the following relations:

Patient	(name, dob, address, insurer)
Procedure	(patient_name, physician_name, procedure_name, time)
Insurer	(company, address, phone)
Event	(event_name, description, patient_name, location)

Suppose that **DB** maintains materialized views:

VIEW **Clinical_History** (name, dob, proc_name, time)
SELECT name, dob, procedure_name, time
FROM **Patient, Procedure**
WHERE **Patient**.name = **Procedure**.patient_name

VIEW **Billing**(name, address, insurer_address)
SELECT name, **Patient**.address, **Insurer**.address
FROM **Patient, Insurer**
WHERE **Patient**.insurer = **Insurer**.company

A user could ask the following query to obtain the names and dates of birth of patients who have had an MRI scan:

SELECT DISTINCT name, dob
FROM **Patient, Procedure**
WHERE **Patient**.name = **Procedure**.patient_name
AND **Procedure**.procedure_name = "MRI"

Using the view **Clinical_History**, the above query can be answered as follows:

SELECT DISTINCT name, dob
FROM **Clinical_History**
WHERE procedure_name = "MRI"

A number of algorithms [4, 12, 18, 19, 22] have been developed for QRV of conjunctive queries and views. Some of these, including [4, 19, 22], were developed for purposes of optimization and, accordingly, seek to produce rewritings (as in Example 1) that are equivalent to the original query. However, there are also applications for rewritings that are known only to be *contained* in the original query. For example, a user may wish to retrieve some initial answers quickly; or, a partial answer set to a query may be acceptable in a distributed environment, when part of the database is unavailable. More importantly, rewritings that are contained in (but not necessarily equivalent to) the original query can be used to provide answers from heterogeneous information sources [13]. As observed by Ullman in [21], rewriting a query in a heterogeneous environment can be formulated as asking for all *partial* solutions to the query, the union of which then becomes the "solution" to the query. This approach to query answering is used, for example, in the Information Manifold [13]. Thus, the crucial issues for QRV in a heterogeneous environment are (1) finding partial answers (containment) and (2) finding *all* such partial answers (completeness). Algorithms that address both these issues are described in [14, 17]. For a given query, they generate rewritings that are contained in the query, and they are complete for conjunctive queries and views when semantic information (such as integrity constraints) for the views is unavailable. The algorithm of [19] uses a variety of integrity constraints, in establishing equivalence of potential rewritings and the query. Thus, it will yield some rewritings not produced by other algorithms. However, to gain efficiency, the algorithm uses pruning conditions that sacrifice completeness. A complete algorithm for QRV in the presence of functional dependencies (FDs) is presented in [7]. The following two examples illustrate that it is possible to find additional rewritings of a query, when semantic information expressed as integrity constraints is available. We consider the two most ubiquitous integrity constraints enforced on databases: functional and inclusion dependencies.

Example 2 Suppose we have the query:

SELECT DISTINCT name, dob, address
FROM **Patient**

Assume that *name* is a key for the relation **Patient**, hence the following functional dependencies hold: $name \rightarrow dob$ and $name \rightarrow address$. Then, the query can be – at least partially – answered using the two materialized views, **Clinical_History** and **Billing** described above, as follows:

SELECT DISTINCT **Clinical_History**.name, dob, address
FROM **Clinical_History, Billing**
WHERE **Clinical_History**.name = **Billing**.name

The rewriting in Example 2 would not be generated by algorithms previous to the one in [7], because they attempt to find a single view to provide the three requested attributes from relation **Patient**. Only the two views together provide the needed attributes, and the join of the two views is a correct rewritings only because the functional dependencies $name \rightarrow dob$ and $name \rightarrow address$ hold on relation **Patient**.

The next example illustrates the use of inclusion dependencies (INDs) to generate new rewriting of a query.

Example 3 Let the database **DB** contain the relations and the views as described in Example 1. Assume additionally that all procedures with patients' names are also recorded in the **Event** table, that is, the following IND holds in **DB**:

$$\textbf{Procedure}(\text{procedure\_name, patient\_name}) \subseteq \textbf{Event}(\text{event\_name, patient\_name})$$

Let the query Q ask for events recorded for patients born before 1930. This query can be expressed as:

SELECT event_name
FROM **Patient, Event**
WHERE **Patient**.name = **Event**.patient_name
AND **Patient**.dob < 1930

This query does not have a rewriting in **DB**, because there are no views using the relation **Event**. However, the query can be rewritten using the inclusion dependency into the following query Q':

SELECT procedure_name
FROM **Patient, Procedure**
WHERE **Patient**.name = **Procedure**.patient_name
AND **Patient**.dob < 1930

Although Q' would return only a subset of the answers of Q, it can be rewritten using the materialized view **Clinical_History** into the query Q'':

SELECT procedure_name
FROM **Clinical_History**
WHERE dob < 1930

Thus, Q'' is a rewriting of Q. It can be useful in cases where **Clinical_history** is a materialized view or a cached query and an initial (although only partial) set of answers to Q can be retrieved fast using Q''.

The rewriting in Example 3 would not be generated by existing QRV algorithms, because there are no views that are defined in terms of the relation **Event**.

The results of this paper provide a complete solution to the problem of QRV for conjunctive queries and views in the presence of INDs. This expands the practical scope of QRV, since the use of INDs enables the generation of rewritings that might otherwise be overlooked. The paper also addresses the issue of QRV in the presence of both FDs and INDs, thus providing a solution to QRV with respect to the most ubiquitous integrity constraints enforced on databases. This paper is an extended version of [8]. It contains all of the details omitted in the short version as well as several new theorems and their complete proofs. It also contains a description of an algorithm for query rewriting in the presence of FDs.

The paper is organized as follows. Section 2 contains terminology and background on QRV. We provide a sketch of an algorithm (from [7]) for QRV in the presence of FDs in Section 3. The algorithms for QRV in the presence of INDs, their complexity analysis and the proof of their correctness is provided in Section 4. The algorithms are extended to take into account both INDs as well as FDs in Section 5. The paper is concluded in Section 6.

2. OVERVIEW OF QUERY REWRITING USING VIEWS

We begin with a brief review of terminology and notation used throughout the paper.

We consider queries and views[†] expressed as safe, conjunctive queries [21] or project-select-join queries where the selection conditions are restricted to equality. We assume that predicates are

[†] We do not assume view completetness here [1].

divided into three disjoint groups: base predicates, which refer to base relations in a database; view predicates, which refer to views, and query predicates, which refer to users' queries. An atomic formula whose arguments are variables or constants is a positive *literal* (atom). A *conjunctive query* Q has the form:

$$q : -p_1, \ldots, p_n.$$

where q is a query atom, whose arguments are distinct variables, $p_1, \ldots, p_n$ are atoms. The atom q is called the *head*, and $p_1, \ldots, p_n$, the *body*. The variables in the head are called *distinguished*. Sets of attributes are denoted by $\mathcal{A}, \mathcal{B}, \ldots$. When it is clear from context we use 'p' to refer to the atom $p(X_1, ..., X_n)$ or simply to the predicate p. An atom $p(X_1, ..., X_n)$ is associated with a relation P with attributes $A_1, ..., A_n$. We use the notation $p[\mathcal{A}]$ to refer to the values of the arguments of the atom p representing attributes in $\mathcal{A}$, for example $p[A_i] = X_i$. The SQL query in Example 1 is equivalent to the conjunctive query:

$$\begin{array}{ll} q(X,Y) : - & \text{patient}(X, Y, W_1, W_2), \\ & \text{procedure}(X, W_3, \text{“}MRI'', W_4). \end{array}$$

We use several standard notions from relational database theory [16]. An n-ary *inclusion dependency* is a formula of the form $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n]$ where R and S are relation schemes and $A_1, ..., A_n$ and $B_1, ..., B_n$ are attributes in R and S respectively. R and S satisfy this dependency if $\pi_{A_1,...,A_n}(R)$ is a subset of $\pi_{B_1,...,B_n}(S)$. For sets of attributes $\mathcal{A}$ and $\mathcal{B}$ in a relation scheme R, $R : \mathcal{A} \to \mathcal{B}$ denotes a functional dependency from $\mathcal{A}$ to $\mathcal{B}$. When clear from the context, the name of the relation scheme in a dependency is omitted. Sets of INDs are denoted by Δ; sets of FDs are denoted by Σ.

The notions of *containment* and *equivalence* relates queries with respect to their answer sets. We are assuming set semantics for queries throughout this paper (that is, queries' answer sets do not contain duplicates).

Definition 1 A conjunctive query Q is *contained* in another conjunctive query Q', denoted as $Q \subseteq Q'$, if the answer set of Q is a subset of the answer set of Q' in all possible databases. Q and Q' are *equivalent*, denoted as $Q \equiv Q'$, if $Q \subseteq Q'$ and $Q' \subseteq Q$.

These notions can be extended to databases with integrity constraints (dependencies).

Definition 2 A conjunctive query Q is Δ-*contained* in another conjunctive query Q', denoted $Q \subseteq_\Delta Q'$, if the answer set of Q is a subset of the answer set of Q' in all databases obeying the set of dependencies Δ. Q and Q' are Δ-*equivalent*, denoted $Q \equiv_\Delta Q'$, if $Q \subseteq_\Delta Q'$ and $Q' \subseteq_\Delta Q$.

Containment of queries in the presence of FDs, $\subseteq_\Sigma$, is defined similarly.

A *view* is expressed as a conjunctive query whose head is a view atom and whose body consists of base atoms only. We assume that no two views contain the same view predicate. Hence, every view completely defines its view predicate.

Next we define the notion of a *query rewriting* with respect to the set of inclusion dependencies Δ.

Definition 3 Let Q be a query, $\mathcal{V}$ be a set of views, and Δ be a (possibly empty) set of dependencies.

A Δ-*rewriting* of Q using $\mathcal{V}$ is a conjunctive query Q' such that $Q' \subseteq_\Delta Q$ and the body of Q' contains one or more view predicates defined in $\mathcal{V}$. A *complete* Δ-*rewriting* does not contain base predicates.

A Δ-rewriting of Q using $\mathcal{V}$ is *maximal* if there are no other Δ-rewriting of Q using $\mathcal{V}$ that contains it.

A *equivalent* Δ-*rewriting* of Q using $\mathcal{V}$ is a Δ-rewriting of Q using $\mathcal{V}$ that is Δ-equivalent to Q.

Σ-rewritings are defined similarly. When it is clear from the context, we will sometimes use the term *rewriting* for Δ-*rewriting* (Σ-*rewriting*), with Δ (Σ) understood. We assume throughout this paper that we need to find only maximal rewritings of a query. This assumption is justified by the fact that an answer set of a non-maximal rewriting of Q is a subset of an answer set of some maximal rewriting of Q.

Example 4 Consider again the view **Clinical_History** from Example 1, now expressed as conjunctive queries in Datalog [20]:

$$\begin{aligned} \text{clinical\_history}(X_1, X_2, X_3, X_4) :- \quad & \text{patient}(X_1, X_2, U_1, U_2), \\ & \text{procedure}(X_1, U_3, X_3, X_4) \end{aligned}$$

and the query from Example 1:

$$\begin{aligned} q(X, Y) :- \quad & \text{patient}(X, Y, W_1, W_2), \\ & \text{procedure}(X, W_3, \text{“}MRI'', W_4). \end{aligned}$$

The following is a complete, maximal, and equivalent rewriting of the query:

$$q(X, Y) :- \text{clinical\_history}(X, Y, \text{“}MRI'', V).$$

3. QUERY REWRITING WITH FDS ALONE

An algorithm for for generating query rewritings in the presence of FDs has been presented in [7]. We sketch the algorithm using the query of Example 2 which can be expressed in Datalog as:

$$q(X, Y, Z) :- \text{patient}(X, Y, Z, W)$$

The first step of the algorithm is the generation of *inverse rules* (also called *folding rules* in [17]) for all available views.

Consider a view of the form

$$r :- p_1, \ldots, p_n.$$

Let $X_1, \ldots, X_l$ be the distinguished variables, and $Y_1, \ldots, Y_m$ be the remaining variables. Since the view predicate in r is completely defined by the body of the view, the view implies the formula (through Clark's completion [15]):

$$(\forall X_1, \ldots, X_l)(r \rightarrow (\exists Y_1, \ldots, Y_m)(p_1 \wedge \cdots \wedge p_n)).$$

After skolemization, we derive n *inverse rules*:

$$\begin{gathered} p_1\theta :- r \\ \vdots \\ p_n\theta :- r \end{gathered}$$

where $\theta = \{Y_i \leftarrow f_i(X_1, \ldots, X_l)\}_{1 \le i \le m}$ is a substitution and $f_1, \ldots, f_m$ are l-ary skolem functions with arguments $X_1, \ldots, X_l$. Notice that, for an inverse rule, the head is a base atom, the body is a view atom, and every variable in the head also appears in the body. For example, from the views in Example 4, we derive the following inverse rules.

$$\begin{aligned} \text{patient}(X_1, X_2, U_1, U_2)\theta_1 &:- \text{clinical\_history}(X_1, X_2, X_3, X_4) \\ \text{procedure}(X_1, U_3, X_3, X_4)\theta_1 &:- \text{clinical\_history}(X_1, X_2, X_3, X_4) \\ \text{patient}(Y_1, V_1, Y_2, V_2)\theta_2 &:- \text{billing}(Y_1, Y_2, Y_3) \\ \text{insurer}(V_2, Y_3, V_3)\theta_2 &:- \text{billing}(Y_1, Y_2, Y_3) \end{aligned}$$

where $\theta_1 = \{U_i \leftarrow f_i(X_1, X_2, X_3, X_4)\}_{1 \le i \le 3}$ and $\theta_2 = \{V_i \leftarrow g_i(Y_1, Y_2, Y_3)\}_{1 \le i \le 3}$.

If no information about FDs is available, inverse rules can be used to replace original atoms of a query with view atoms ([17] presents such an algorithm). Note, that for the query of Example 2 it would not be possible, since there is no inverse rule that allows replacing the atom *patient*(X, Y, Z, W) in such a way that the values of its first three attributes could be projected.

To generate the rewriting of the query, we need to be able to use the semantic information about the key. This is done by means of *chase rules*. A new binary predicate $e(c_1, c_2)$ is introduced with the intended meaning that $e(c_1, c_2)$ holds if c_1 and c_2 must be equal under the given FDs. For the relation **Patient**, the following chase rules can be stated:

$$
\begin{aligned}
e(Y_1, Y_2) &:- \text{patient}(X_1, Y_1, Z_1, W_1), \text{patient}(X_2, Y_2, Z_2, W_2), e(X_1, X_2). \\
e(Z_1, Z_2) &:- \text{patient}(X_1, Y_1, Z_1, W_1), \text{patient}(X_2, Y_2, Z_2, W_2), e(X_1, X_2). \\
e(W_1, W_2) &:- \text{patient}(X_1, Y_1, Z_1, W_1), \text{patient}(X_2, Y_2, Z_2, W_2), e(X_1, X_2).
\end{aligned}
$$

The next step of the construction is to rewrite the query in such a way that all implicit equalities between the variables as well as between the variables and the constants in the query are explicitly represented by means of the predicate e. Thus, a the body of a given query $\mathcal{Q}$ is modified as follows. If c is a constant in one of the atoms of $\mathcal{Q}$, we replace it by a new variable Z, and add an atom $e(Z, c)$. If X is a variable in the head of $\mathcal{Q}$, we replace X in the body of $\mathcal{Q}$ by a new variable X', and add the atom $e(X, X')$. If a variable Y that is not in the head of $\mathcal{Q}$ appears in two atoms of $\mathcal{Q}$, we replace one of its occurences by Y', and add an atom $e(Y, Y')$. We continue until we cannot apply this rule anymore. Our example query would be rewritten to:

$$q(X, Y, Z) : -\text{patient}(X', Y', Z', W), e(X, X'), e(Y, Y'), e(Z, Z').$$

Assuming that e is reflexive and transitive, we can derive the intended rewriting of the original query of Example 2:

$$q(X, Y, Z) : -\text{clinical\_history}(X, Y, U_1, U_2), \text{billing}(X, Z, U_3)$$

It has been shown in [7] that the query plan (that is, the set of inverse rules and chase rules) generated by the algorithm sketched above is *maximally-contained* in the original query, that is, the rewriting it produces for a given query, is maximal. It is also shown that generating such a plan is polynomial in the size of the query, set of resources and set of FDs.

4. QUERY REWRITING WITH INDS ALONE

4.1. Preliminaries

In the algorithms that follow, we need to be able to refer not only to the set Δ of INDs as stated for a given database, but to its closure Δ^* which is defined as the set of all INDs implied by Δ. Henceforth, we will assume that the set Δ is closed under its consequences (that is, $\Delta = \Delta^*$).

Before we describe rewriting algorithms, we introduce the concept *chase*$_\Delta$ and *chase*$_\Sigma$ (the latter will be used in Section 5), which describe procedures applied to atoms in the body of a query.

Definition 4 [9]

The Chase$_\Delta$ Procedure

An IND $R[A_1, ..., A_n] \subseteq S[B_1, ..., B_n] \in \Delta$ is applicable to an atom $r(X_1, ..., X_m)$ where $m \geq n$ if r refers to the relation R. Each step of a chase procedure consists of an application of the chase rule to a given IND and an atom $r(X_1, ..., X_m)$ to which it is applicable.

The Chase$_\Delta$ Rule

Let an IND and an atom be as above. Add a new atom $s(Y_1, ..., Y_k)$ to the query, where $s[B_i] = r[A_i]$ for $1 \leq i \leq n$ and where $s[C]$ for $C \neq B_i,\ 1 \leq i \leq n$, is a new variable that does not appear in the query.

Definition 5 [9] **The Chase$_\Sigma$ Procedure**

Let P be a relation with attributes $A_1, ..., A_n$. If $P : \mathcal{A} \to \mathcal{A}_|$, ($\mathcal{A} \subseteq \{\mathcal{A}_\infty, ..., \mathcal{A}_\backslash\}$), is an FD in Σ and $p_1(Z_1, ..., Z_n)$ and $p_2(W_1, ..., W_n)$ are atoms on the same relation P (where $Z_1, ..., Z_n, W_1, ..., W_n$ are either variables or constants) such that $p_1[\mathcal{A}] = \surd_\infty[\mathcal{A}]$ and $Z_j \neq W_j$, we say that the *chase rule* is applicable to $p_1(Z_1, ..., Z_n)$ and $p_2(W_1, ..., W_n)$. Each step of a chase procedure consists of an application of the chase rule to a given FD and a pair $p_1(Z_1, ..., Z_n)$ and $p_2(W_1, ..., W_n)$ to which it is applicable.

The Chase$_\Sigma$ Rule

Let $p_1(Z_1, ..., Z_n)$ and $p_2(W_1, ..., W_n)$ be atoms and $\mathcal{A} \to \mathcal{A}_|$ be an FD as described above.

If Z_j and W_j are different constants, delete all atoms from the query and halt (this query cannot be chased to an equivalent query obeying the given FD).

If one, say Z_j, is a constant, set W_j to be that constant as well.

If one, say Z_j, is a distinguished variable, set W_j to be that distinguished variable as well.

If both Z_j and W_j are non-distinguished variables, we choose the lexicographically first of the two as the value for the new symbol.

We illustrate definitions 4 and 5 on the following example.

Example 5 Let $q(X, Y) : -\text{procedure}(X, Y)$ be the query and let **Procedure**(procedure_name, patient_name) $\subseteq$ **Event**(event_name, patient_name) be the IND as described in Example 3. Then, the chase$_\Delta$ procedure results in the query: $q(X, Y) : -\text{procedure}(X, Y), \text{event}(X, Z, Y, W)$.

Let $q(X, Y) : -\text{patient}(X, Y, W_1, W_2), \text{patient}(X, Y_2, W_3, W_4)$. be a query and the first attribute of the relation **Patient** be a key. An application of chase$_\Sigma$ procedure results in the query: $q(X, Y) : -\text{patient}(X, Y, W_1, W_2), \text{patient}(X, Y, W_1, W_2)$.

We also define the following relation between queries.

Definition 6 Let Q, Q' be queries. Q is *chase$_\Delta$ reachable* from Q' if $Q \equiv Q'$ or $\exists Q_1, ..., Q_n$, such that:

- $Q' \equiv Q_1$
- $Q \equiv Q_n$
- Q_{i+1} is derived from Q_i, $1 \leq i \leq n-1$, by applying a single chase$_\Delta$ step to it.

Next we define the notion of a *scope* of an atom. Informally a scope is the set of "interesting" attributes of an atom.

Definition 7 Let Q: $q : -p_1, ..., p_n$ be a query. The *scope* of p_i for Q is the set of attributes in P_i (relation for p_i) corresponding to distinguished variables, constants, and join variables (that is, variables shared by at least two atoms in the body of Q), of the atom p_i in Q.

Example 6 For the query of Example 4:

$$\begin{aligned} q(X, Y) : - \quad & \text{patient}(X, Y, W_1, W_2), \\ & \text{procedure}(X, W_3, \text{“}MRI''\text{”}, W_4). \end{aligned}$$

The scope of *patient*(X, Y, W_1, W_2) is the set of the first two attributes, $\{$*name, dob*$\}$, of the relation **Patient**. The scope of the atom *procedure*$(X, W_3, \text{“MRI”}, W_4)$ is the set with the first and the third attribute of the relation **Procedure**, that is, $\{$*patient_name, procedure_name*$\}$.

The notion of a scope is used to define *atom containment.*

Definition 8 Let p be an atom in the query Q and p' be an atom in the query Q' with *scope*$(p') = \mathcal{A}$. Then p is *contained* in p' (we also say that p' contains p) with respect to the set of INDs Δ iff the IND $P[\mathcal{B}] \subseteq P'[\mathcal{A}] \in \Delta$ and $p[\mathcal{B}] = p'[\mathcal{A}]$.

An atom p is equivalent to an atom p' if they are mutually contained in one another.

The algorithm for generating Δ-rewritings of a query consists of two steps. The second step of the algorithm, which we do not present in detail here, implements one of the existing procedures for finding rewritings of a query (for example, the one presented in [17]). We concentrate here on the first step and present it in two different versions. The first version of our algorithm generates queries Δ-equivalent to the original query. The second version generates queries which are Δ-contained in the original query.

4.2. The Strategy

The procedure for generating all queries Δ-equivalent to a given query is based on a notion of chase$_\Delta$ as described above. The following theorem provides the link between the notion of Δ-*containment* and the notion of $chase_\Delta$.

Theorem 1 *[16]*
Let Q and Q' be queries. Then $Q \subseteq_\Delta Q'$ iff $chase_\Delta(Q) \subseteq chase_\Delta(Q')$.

Consider an input query Q_I. Our task is to find all queries that are Δ-equivalent to Q_I, that is, we need to find elements of the set $\mathcal{Q}$ defined as follows:

$$\mathcal{Q} = \{Q \mid chase_\Delta(Q) \equiv chase_\Delta(Q_I)\}$$

Another way of describing the set $\mathcal{Q}$ is:

$$\mathcal{Q} = \{Q \mid chase_\Delta(Q_I) \text{ is chase}_\Delta \text{ reachable from } Q\}$$

The set $\mathcal{Q}$ can be depicted graphically as shown in Figure 1.

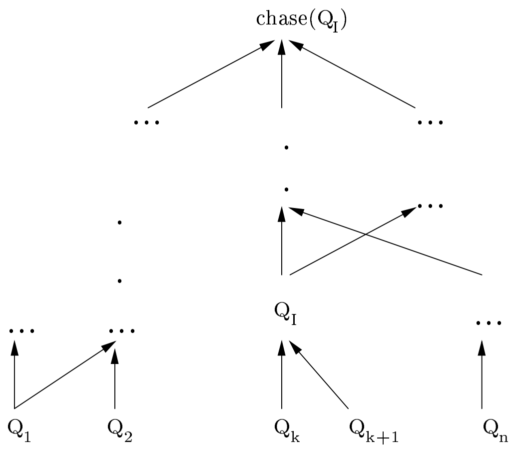

Fig. 1: The set $\mathcal{Q}$ of queries related by the relation of chase reachability.

An arrow from Q_i to Q_j indicates that Q_j is chase$_\Delta$ reachable from Q_i. Thus, the query $chase_\Delta(Q_I)$ is chase$_\Delta$ reachable from all queries in the set $\mathcal{Q}$ (this is a convergence point for the relation of chase reachability). It can be shown that all Δ-rewriting of Q_I can be generated by computing rewritings of all elements of $\mathcal{Q}$. Clearly, this is not practical, since $\mathcal{Q}$ can be large (for some types of INDs $\mathcal{Q}$ can be infinite). Our goal then, is to find the smallest possible subset of $\mathcal{Q}$ that contains all queries necessary to generate all Δ-rewritings of Q_I. We identify such a subset below and call its elements *minimal* queries; they are represented in Figure 1 as $Q_1, ..., Q_n$. Given Q_I, we find the set of minimal queries of $\mathcal{Q}$ in two steps. First, we find one minimal query by undoing (if possible) all chase$_\Delta$ steps from Q_I. In Figure 1 this process would result in Q_k or Q_{k+1}. The second step, involves replacing (when possible) all atoms of such a query with atoms equivalent with respect to Δ. These two steps are described as procedures **minimize** and **find_equiv_atoms** in Section 4.3.

Next, we define formally minimal queries and prove that they are sufficient to generate all rewritings of a given query.

Definition 9 Q is *minimal* (w.r.t. Δ) iff there is no Q' such that Q is chase$_\Delta$ reachable from Q'.

The following corollary follows immediately from Definition 9 and the notion of atom containment.

Corollary 2 *The query q :- $p_1, ..., p_n$ is not minimal iff $\exists\ p_i, p_j,\ i \neq j$, such that p_i is contained in p_j.*

Finally, the following lemma allows us to restrict our attention to minimal queries when looking for rewritings of a query.

Lemma 1 *Let $\mathcal{Q}$ be the set of queries Δ-equivalent to Q_I (i.e. $\mathcal{Q} = \{Q \mid chase_\Delta(Q) \equiv chase_\Delta(Q_I)\}$). Then if Q is a rewriting of Q_I, then there is $Q' \in \mathcal{Q}$ which is minimal and Q is a rewriting of Q'.*

Proof. Q_I is either minimal or it is chase$_\Delta$ reachable from another query $Q' \in \mathcal{Q}$. If it is chase$_\Delta$ reachable from Q', then $Q_I \subseteq Q$ (chase$_\Delta$ steps do not change the query except adding extra atoms to it). Thus, if Q is a rewriting of Q_I, then Q is also a rewriting of Q'.

□

4.3. Algorithm for Equivalent Rewritings

Our strategy for finding queries that would generate all rewritings equivalent to a given query is as follows.

Algorithm 1, **new_queries**, is the top-level procedure for generating a set of queries which are minimal and Δ-equivalent to a given query Q_I. The first step of the algorithm produces a single arbitrary minimal query which is Δ-equivalent to the input query Q_I; this is achieved through procedure **minimize**. Then for each atom in that query, all atoms equivalent to it are found using the subprocedure **find_equiv_atoms**. Last, all Δ-equivalent queries are generated by combining equivalent atoms from the sets Atoms$_1$,...,Atoms$_n$.

Procedure **minimize** removes redundant atoms (with respect to Δ) from the query. This means reversing the chase$_\Delta$ steps that may have been applied to some minimal query to produce Q_I. This is done by testing for each atom p_i in the body of Q_I, whether p_i can be removed from Q_I. The removal is possible if there is another atom p_j such that there is an IND between relations P_i and P_j which renders p_i redundant. If this is the case, p_i is removed.[†] The purpose of the last step of **minimize**, removing redundant atoms, is to remove the atoms from the query that might have lost connections (through shared variables) with other atoms in the query (see Lemma 2 for justification of this step).

Procedure **find_equiv_atoms** takes an atom p and a set Δ as input and generates a set of all atoms equivalent to p. Function *new_var* returns a new variable that has not occured before.

Algorithm 1

new_queries(Q_I, Δ)
    Q := **minimize**(Q_I)
    $p_1, ..., p_n$:= body(Q)
    for $i = 1$ to n
        Atoms$_i$:= **find_equiv_atoms**(p_i, Q, Δ)
    Return all queries Q' s.t.
        head(Q') := head(Q_I)
        body$(Q') := r_1, ..., r_n$ where $r_i \in$ Atoms$_i$

[†]The notion of a *scope* (referred to in the algorithm **minimize**) is defined in Definition 7.

Algorithm 2

```
minimize(Q, Δ)
        p_1, ..., p_n := body(Q)
        for i = 1 to n
                A := scope(p_i)
                if P_j[B] ⊆ P_i[A] ∈ Δ, i ≠ j
                and p_j[B] = p_i[A] then
                    body(Q):= p_1, ..., p_{i-1}, p_{i+1}, ..., p_n
        Remove redundant atoms from Q
        Return Q
```

Algorithm 3

```
find_equiv_atoms (p, Q, Δ)
    A := scope(p)
    Atoms := {p}
    for each R[B] ≡ P[A] ∈ Δ
            r[B] := p[A]
            for each attribute C of R s.t.  C ∉ B
                r[C]:= new_var
            Atoms := Atoms ∪{r}
    Return Atoms
```

We trace the algorithm **new_queries** on the following example.

Example 7 Let the query be:

$$q(X, Z) : -p(X, Y), r(Y, Z, W), s(W, U)$$

where predicates p, r, s, t refer to relation schemes $P[A_1, A_2]$, $R[B_1, B_2, B_3]$, $S[C_1, C_2]$, $T[D_1, D_2, D_3]$ respectively. The set Δ contains the following INDs: IND_1: $R[B_3] \subseteq S[C_1]$, IND_2: $P[A_1, A_2] \subseteq T[D_1, D_3]$, IND_3: $T[D_1, D_3] \subseteq P[A_1, A_2]$.

The first step of **new_queries** calls the procedure **minimize**. Because of IND_1, the atom $s(W, U)$ is removed from the body of the query. For each of the remaining two atoms of the modified query, the procedure **find_equiv_atoms** is called. For the atom $p(X, Y)$, it returns the set $\text{Atoms}_1 := \{p(X, Y), t(X, V, Y)\}$ where V is a new variable that does not appear anywhere else in the query. For the atom $r(Y, Z, W)$, the set consisting of that atom only is returned, that is, $\text{Atoms}_2 := \{r(Y, Z, W)\}$.

The last step of **new_queries** generates the queries:

$$q(X, Z) : -p(X, Y), r(Y, Z, W)$$
$$q(X, Z) : -t(X, V, Y), r(Y, Z, W)$$

A standard QRV algorithm then can be applied to each of these queries to generate all Δ-rewritings equivalent to the original query.

4.4. Extension of the Algorithm to all Rewritings

The algorithm for finding all rewritings, that is, rewritings not necessarily equivalent to the original query, differs little from the algorithm descibed above. The only difference occurs in the procedure **find_equiv_atoms**, which we now call **find_contained_atoms**. The procedure is shown as Algorithm 4. Instead of generating atoms equivalent to a given atom, the procedure now finds all atoms *contained* in a given atom.

Algorithm 4

find_contained_atoms(p, Δ)

```
    𝒜 := scope(p)
    Atoms := {p}
    for each R[ℬ] ⊆ P[𝒜] ∈ Δ
        r[ℬ] := p[𝒜]
        for each attribute C of R s.t.  C ∉ ℬ
            r[C]:= new_var
        Atoms:=Atoms∪{r}
    Return Atoms
```

4.5. Correctness and Complexity of the Algorithms

We describe first the proof of correctness of the algorithm to find queries equivalent to the original query.

The following lemma describes a strategy for finding *a* minimal element of $\mathcal{Q}$. It shows that the chase steps may be undone in a random order. This is what the procedure **minimize** implements.

Lemma 2 *Let Q be a minimal query of $\mathcal{Q}$ and Δ be its own closure. Then, if Q' is chase$_\Delta$ reachable from Q, then for each atom p of Q'*

- *p does not contain any variables occuring in Q, or*
- *p can be generated from some $r \in Q$ by a single chase$_\Delta$ step using Δ.*

Proof. : By induction on the number of chase steps from Q to Q'.

1. $n = 1$; trivial.

2. Assume the theorem holds for n. Let Q_1 be a query which is chase$_\Delta$ reachable from Q in n steps, and such that Q' is chase$_\Delta$ reachable from Q_1 in one chase$_\Delta$ step. Consider an atom p of Q'. p is added to Q_1 by an application of some $\delta \in \Delta$ to an atom s of Q_1, where $\delta = S[\mathcal{A}] \subseteq \mathcal{P}[\mathcal{B}]$. By the induction hypothesis one of the following cases must hold:

- s does not contain any variables of Q. Then p cannot contain them either.

- s can be generated from $r \in Q$ by a single chase$_\Delta$ step using $\delta_1 \in \Delta$, where $\delta_1 = R[\mathcal{C}] \subseteq \mathcal{S}[\mathcal{D}]$. There are three subcases here.

  1. $s[\mathcal{A}]$ does not contain any variables from Q. Then p cannot contain them either.

  2. $s[\mathcal{A}]$ contains *only* variables from Q. Then $\mathcal{A} \subseteq \mathcal{D}$. Let $\mathcal{D} = [A_1, ..., A_m]$ and $\mathcal{A} = [A_{i_1}, ..., A_{i_k}]$. Let also $\mathcal{C} = [B_1, ..., B_m]$. Define $\mathcal{E} = [B_{i_1}, ..., B_{i_k}]$. Since $R[\mathcal{C}] \subseteq \mathcal{S}[\mathcal{D}]$, then by permutation and projection axiom (see [2] for axiomatization of INDs) $R[\mathcal{E}] \subseteq \mathcal{S}[\mathcal{A}]$ is in Δ. By transitivity from $R[\mathcal{E}] \subseteq \mathcal{S}[\mathcal{A}]$ and $S[\mathcal{A}] \subseteq \mathcal{P}[\mathcal{B}]$ we also get the inclusion $R[\mathcal{A}] \subseteq \mathcal{P}[\mathcal{B}]$. Hence p can be generated from r by a single application of this inclusion dependency.

  3. $s[\mathcal{A}]$ contains some variables from Q, that is, there are attributes in the set $\mathcal{A}_1 \subseteq \mathcal{A}$ such that $s[\mathcal{A}_1]$ contains only variables from Q. By reasoning similar to the one above we can show that there is a dependency in Δ which allows deriving p by a single chase$_\Delta$ step from r.

□

Once *one* minimal element of $\mathcal{Q}$ is generated, we find the remaining minimal elements of $\mathcal{Q}$. For each atom in the body of Q, we identify atoms Δ-equivalent to it and replace it (with the appropriate substitution) with each of these atoms. This is what procedure **find_equiv_atoms** implements. We state this fact as the following lemma.

Lemma 3 *Let $Q :- p_1, ..., p_n$ and $Q' :- r_1, ..., r_m$ be any two minimal queries in $\mathcal{Q}$. Then there is a one-to-one correspondence between atoms in the two queries, such that p_i is contained in r_i and r_i is contained in p_i.*

Proof. All atoms in $\text{chase}_\Delta(Q)$ contain atoms in Q and all atoms in $\text{chase}_\Delta(Q')$ contain atoms in Q'. Also, since $\text{chase}_\Delta(Q) \equiv \text{chase}_\Delta(Q')$, then all atoms in $\text{chase}_\Delta(Q')$ contain atoms in Q and all atoms in $\text{chase}_\Delta(Q)$ contain atoms in Q'. Consider the atom p_i. Clearly, p_i is among the atoms of $\text{chase}_\Delta(Q)$. Hence, p_i contains some atom r_j from Q'. Assume that r_j does not contain p_i. Then, there must be another atom p_k s.t. r_j contains p_k. But then, by transitivity of atom containment, p_i contains p_k, so, by Corollary 2, Q is not minimal, a contradiction. □

Note that the correspondence described in Lemma 3 holds between atoms of the two minimal queries in Example 7:

$$q(X, Z) :- p(X, Y), r(Y, Z, W)$$

$$q(X, Z) :- t(X, V, Y), r(Y, Z, W)$$

It is so, because $r(Y, Z, W)$ is trivially contained in $r(Y, Z, W)$ and $p(X, Y)$ is contained in $t(X, V, Y)$ (and vice versa) due to inclusion dependencies IND_2 and IND_3.

We can now state the main theorem and prove it in one direction (proving the other direction, that is, soundnes, is straightforward). The theorem says that if Q' is an equivalent Δ-rewriting of Q, it will be generated by running the algorithm **new_queries** followed by an application of standard rewriting algorithm (such as the one described in [17]).

Theorem 3 *Given a conjunctive query Q and a set Δ of INDs on the base relations of predicates in Q, if Q' is a equivalent Δ-rewriting of Q, then there is Q'' such that Q'' is returned by the algorithm* **new_queries** *and Q' is a equivalent rewriting of Q''.*

Proof.

By definition of equivalent Δ-rewriting we have $Q' \equiv_\Delta Q$. Since $\text{unfold}(Q') \equiv Q'$, we also have† $\text{unfold}(Q') \equiv_\Delta Q$. By Theorem 1, we have $\text{chase}_\Delta(\text{unfold}(Q')) \equiv \text{chase}_\Delta(Q)$. Let $Q'' = \text{unfold}(Q')$. By definition of rewriting we obtain: $\exists Q''$ $\text{chase}_\Delta(Q'') \equiv \text{chase}_\Delta(Q)$ and Q' is an equivalent rewriting of Q''. By Lemma 1, we have $\exists Q''$ where Q'' is minimal and $\text{chase}_\Delta(Q'') \equiv \text{chase}_\Delta(Q)$ and Q' is an equivalent rewriting of Q''. By Lemmas 2 and 3, the result follows.

□

Next we prove the following Lemma which is used in the proof of correctness of the algorithm extended to all rewriting (Theorem 4).

Lemma 4 *Let Q_2 be a query. If $\exists Q_1$ s.t. $\text{chase}_\Delta(Q_1) \subseteq \text{chase}_\Delta(Q_2)$ and Q is a rewriting of Q_1, then $\exists Q_3$ such that atoms($\text{chase}_\Delta(Q_2)$) $\subseteq$ atoms($\text{chase}_\Delta(Q_3)$) and Q is a rewriting of Q_3.*

Proof. We will construct Q_3 as follows. Since $\text{chase}_\Delta(Q_1) \subseteq \text{chase}_\Delta(Q_2)$, then there is a mapping, h, from atoms of $\text{chase}_\Delta(Q_2)$ to atoms of $\text{chase}_\Delta(Q_1)$. In particular, all atoms in Q_2 are mapped, according to h, to atoms in $\text{chase}_\Delta(Q_1)$. If all atoms in Q_2 are mapped to atoms in Q_1, then we are done (i.e. $Q_3 := Q_2$). Else, there is an atom p in Q_2 that maps to an atom p' in $\text{chase}_\Delta(Q_1)$, but not to an atom in Q_1. Then there must be an atom r' in Q_1 s.t. r' is contained in p'. Replace p in Q_2 with an atom r s.t. $h(r) = r'$ and r is contained in p (this must be possible since $h(p) = p'$). Repeat this process until all atoms in Q_2 are mapped to atoms in Q_1, call the

†The functor *unfold* is the reverse of rewriting, i.e. $Q = \text{unfold}(Q')$ if Q' is a rewriting of Q.

resulting query Q_3. Note that all original atoms in Q_3 are contained in atoms in Q_2. Hence atoms(chase$_\Delta(Q_2)$) $\subseteq$atoms(chase$_\Delta(Q_3)$). Also, since $Q_1 \subseteq Q_3$, any rewriting of Q_1 is a rewriting of Q_3. □

Finally, in Theorem 4 we prove the correctness of the algorithm for finding all rewritings of a query. Theorem 4 states that if Q' is a Δ-rewriting of Q, then Q' can be generated by applying a standard rewriting algorithm to a query which is returned by the algorithm **new_queries** (with the procedure **find_equiv_atoms** replaced by the procedure **find_contained_atoms**).

Theorem 4 *Given a conjunctive query Q and a set Δ of inclusion dependencies on the base relations of predicates in Q, and an arbitrary query Q_{min} which is minimally equivalent to Q, Q' is a Δ-rewriting of Q iff $\exists Q''$ such that Q' is a rewriting of Q'' and each atom in Q'' is contained in an atom in Q_{min}.*

Proof. Again, we prove the completeness only ($\Rightarrow$).

By definition of Δ-rewriting we have $Q' \subseteq_\Delta Q$. Since unfold$(Q') \equiv Q'$, we also have unfold$(Q') \subseteq_\Delta Q$. By Lemma 1, we have chase$_\Delta$(unfold(Q')) $\subseteq$ chase$_\Delta(Q)$. Let Q'' = unfold(Q'). By definition of rewriting we obtain: $\exists Q''$ chase$_\Delta(Q'') \subseteq$ chase$_\Delta(Q)$ and Q' is a rewriting of Q''. Since chase$_\Delta(Q) \equiv$ chase$_\Delta(Q_{min})$, then $\exists Q''$ chase$_\Delta(Q'') \subseteq$ chase$_\Delta(Q_{min})$ and Q' is a rewriting of Q''. By Lemma 4, we have $\exists Q''$ atoms(chase$_\Delta(Q_{min})$) $\subseteq$ atoms(chase$_\Delta(Q'')$) and Q' is a rewriting of Q''. Then, all atoms in Q_{min} are contained in atoms in Q'' and Q' is a rewriting of Q''.

□

Finally, we describe the complexity of the algorithm.

Theorem 5 *The complexity of Algorithm 1,* **new_queries**, *is $O(n * |\Delta^*|)$, where n is the size of the query and $|\Delta^*|$ is the number of INDs in the closure of Δ.*

Proof. Algorithm 1, **new_queries**, consists of two steps (the first one is implicit):

1. computing the closure Δ^* of Δ
2. running the algorithm **new_queries**

A brute force approach to the first step would require testing for each IND that can be expressed in the database language, whether it is implied by the set Δ. In general, this test is expensive: it has been shown in [2] to be PSPACE-complete. However, in practical applications in which INDs are at most k-ary for some fixed k (that is, INDs of the form $R[A_1, ..., A_r] \subseteq S[B_1, ..., B_r]$ where $r \leq k$), there is a polynomial time algorithm for this test. More important, the computation of Δ^* need not be done at run time, since the set Δ is not likely to change frequently. Thus, the computation of Δ^* can be done off-line, amortizing the cost over many queries. Hence, we ignore this cost in the remainder of our discussion.

The second step, the algorithm **new_queries** implements two subprocedures: **minimize** and **find_equiv_atoms**. **minimize** tests for each atom p_i of the query, whether there is an IND such that p_i is redundant. Hence, in the worst case the complexity of this procedure is $n * |\Delta^*|$, where n is the size of the query and $|\Delta^*|$ is the number of INDs in Δ^*. This brute force approach can be easily optimized by indexing INDs so that only a part of Δ^* has to be searched for INDs applicable to a given atom. The procedure **find_equiv_atoms** is called n times, testing each time existence of equivalences applicable to a given atom. Thus, similarly to **minimize** its complexity is $n * |\Delta^*|$ (optimization similar to the one suggested for **minimize** can be applied here also). Note that in the worst case, the complexity of the procedure **find_contained_atoms** (used to find *all* Δ-rewritings of a query) is identical to the complexity of **find_equiv_atoms**. The complexity of these two procedures are likely to differ only for the average case, since the number of INDs applicable to a given atom (line 3 of **find_contained_atoms**) is larger than the number of equivalences applicable to an atom (line 3 of **find_equiv_atoms**).

□

We note here that the cost of computing Δ-rewritings increases little over the cost of computing rewritings (that is, without INDs). The problem of determining whether there exists a rewriting of a conjunctive query $\mathcal{Q}$ that uses a set of conjunctive views $\mathcal{V}$ is NP-complete [12]. Thus, the total cost of computing Δ-rewritings will be dominated by that cost.

5. QUERY FOLDING WITH FDS AND INDS

5.1. *General Case*

Although INDs and FDs have a simple complete axiomatization, the interaction of INDs and FDs is complex. Indeed, it has been shown [2] that there is no finite complete axiomatization for INDs and FDs taken together. The following example shows how INDs and FDs combined can generate new dependencies not implied by INDs and FDs alone.

Example 8 The following two dependencies: $R[A, B] \subseteq S[C, D]$ and $S : C \rightarrow D$ imply a new functional dependency $R : A \rightarrow B$.

The following three dependencies: $R[A, B] \subseteq S[C, D]$, $R[A, E] \subseteq S[C, F]$, and $C \rightarrow D$ imply a new inclusion dependency: $R[A, B, E] \subseteq S[C, D, F]$.

It has been shown [3] that the inference problem for INDs and FDs is undecidable even if we consider only binary INDs. The following observation is a consequence of that result.

Theorem 6 *There does not exist a* complete *algorithm for query rewriting with FDs and INDs which are at least binary.*

Proof. We use in the proof the concept of $\text{chase}_{\Delta\cup\Sigma}$. This is a procedure in which chase_{Δ} steps and chase_{Σ} steps are applied to the query in an arbitrary order.

We prove the theorem by contradiction. Assume that there is such an algorithm, call it **Test**. We design a procedure that will return YES when a given dependency σ is *not* implied by $\Delta \cup \Sigma$. The procedure is as follows: Consider a query Q to which a single $\text{chase}_{\Delta\cup\Sigma}$ step using σ is applicable. Let Q' be the result of chasing Q with respect to σ. Run **Test** on Q' with respect to dependencies in $\Delta \cup \Sigma$. If Q is not among the rewritings of Q', σ is not implied by $\Delta \cup \Sigma$.

Given the fact that the unrestricted implication problem for FDs and INDs is r.e. (this is a consequence of the fact that they can be expressed as first-order logic formulas, for which implication is r.e.), we are able to decide whether or not σ is implied by $\Delta \cup \Sigma$, a contradiction.

$\square$

Thus, in the general case, we can at best produce *some* rewritings of a query. A natural approach in this case would be to generate all rewritings of a query with respect to $\Delta^* \cup \Sigma^*$.[†] This can be achieved as shown in Algorithm 5. The procedure $\textbf{rewrite}_{\Sigma}$ is the algorithm for finding Σ-rewritings of a query sketched in Section 3, and $\mathcal{V}$ is a set of available views.

Algorithm 5

$\textbf{rewrite}_{\Sigma\cup\Delta}(Q, \mathcal{V}, \Sigma, \Delta)$
    $\mathcal{Q}$:=$\textbf{new\_queries}(Q, \Delta)$
    `For each` $Q \in \mathcal{Q}$
        `Return` $\textbf{rewrite}_{\Sigma}(Q, \mathcal{V})$

In the following two subsections we present two special cases for which complete algorithms for query rewriting with INDs and FDs are possible.

5.2. *Unary Inclusion Dependencies*

It has been shown [10] that the inference problem for UINDs (unary inclusion dependencies) and FDs is decidable. In fact, this result is the strongest possible in some sense, because - as we stated above - the inference problem for INDs and FDs remains undecidable even if we consider only binary INDs. The only interaction between UINDs and FDs is the one due to non-standard FDs of the form $\emptyset \rightarrow Y$. These FDs force a column to have at most one element. The following example illustrates this type of interaction.

[†] To guarantee completeness of the algorithm, it would have been necessary to find all rewritings with respect to $(\Delta \cup \Sigma)^*$ which, as we stated above, is not computable. Note that $(\Delta^* \cup \Sigma^*) \subseteq (\Delta \cup \Sigma)^*$.

Example 9 Let $\emptyset \to A$ and $B \subseteq A$ be the set of dependencies. Then, $\emptyset \to B$ and $A \subseteq B$ also hold.

[10] provides a sound and complete axiomatization for UINDs and FDs. Hence, the closure of the set of UINDs and FDs can be computed and is finite.

We show that Algorithm 5 generates all rewritings$_{\Delta\cup\Sigma}$ of a query where all INDs in Δ are unary. First, we state a lemma that relates query containment with respect to $\Delta \cup \Sigma$ with the notion of chase$_{\Delta\cup\Sigma}$ described in the proof of Theorem 6.

Lemma 5 *Let Q, Q' be queries, chase$_{\Delta\cup\Sigma}$ be as described above, and Δ be a set of UINDs. Then,*

$$Q \subseteq_{\Delta\cup\Sigma} Q' \text{ iff } chase_{\Delta\cup\Sigma}(Q) \subseteq chase_{\Delta\cup\Sigma}(Q')$$

Proof. Similar to the proof of Theorem 1 of [9]. □

Next, we show that applying all chase$_\Sigma$ steps before any chase$_\Delta$ steps generates a query equivalent to the query which results from applying chase$_\Delta$ and chase$_\Sigma$ steps in an arbitrary order.

Theorem 7

$$chase_{\Delta\cup\Sigma}(Q) \equiv chase_\Delta(chase_\Sigma(Q))$$

Proof. We show that after all chase steps with respect Σ have been applied, followed by all chase steps with respect to Δ, then there is no need to apply again any of the chase steps with respect to Σ. Let $Q_1 = \text{chase}_\Sigma(Q)$ and $Q_2 = \text{chase}_\Delta(Q_1)$. There are two cases:

1. An FD is applicable to some two atoms which are in Q_2, but not in Q_1. Then, these atoms must have been generated by an application of a chase$_\Delta$ rule and are of the form $p(X, Y_1, ..., Y_n)$ and $p(X, Z_1, ..., Z_n)$, where none of the variables $Y_1, ..., Y_n$ and $Z_1, ..., Z_n$ appear anywhere else in the query. But then $p(X, Y_1, ..., Y_n)$ and $p(X, Z_1, ..., Z_n)$ are identical modulo variable renaming, so one of them can be removed from the query and FD is no longer applicable.

2. An FD is applicable to some two atoms, one of which is in Q_1 and the other one is in Q_2, but not in Q_1. (Note that they both cannot be in Q_1, since this would mean that chase$_\Sigma$ has not been completed.) Let these two atoms be respectively $p(X, Y_1, ..., Y_n)$, where some of $Y_1, ..., Y_n$ can be constants and $p(X, Z_1, ..., Z_n)$, where none of the variables $Z_1, ..., Z_n$ appear anywhere else in the query. But then $p(X, Z_1, ..., Z_n)$ is redundant, because it is subsumed by $p(X, Y_1, ..., Y_n)$, hence it can be removed from Q and the FD is no longer applicable.

□

Finally, we state the completeness result about Algorithm 5 for FDs and UINDs.

Theorem 8 *Let Q be a query, Δ be a set of UINDs and Σ a set of FDs. Then, Algorithm 5 generates all rewritings of Q with respect to Δ and Σ.*

Proof. By Theorem 5, $Q \equiv_{\Delta\cup\Sigma} Q'$ iff $\text{chase}_{\Delta\cup\Sigma}(Q) \equiv \text{chase}_{\Delta\cup\Sigma}(Q')$. Assume without loss of generality that the input query cannot be chased any further, that is, $Q' \equiv \text{chase}(Q')$. Thus, $Q \equiv_{\Delta\cup\Sigma} Q'$ iff $\text{chase}_{\Delta\cup\Sigma}(Q) \equiv Q'$. By Theorem 7, $\text{chase}_{\Delta\cup\Sigma}(Q) \equiv Q'$ iff $\text{chase}_\Delta(\text{chase}_\Sigma(Q)) \equiv Q'$. Let $Q'' \equiv \text{chase}_\Sigma(Q)$. By Theorem 1 $\text{chase}_\Delta(Q'') \equiv Q'$ iff $Q'' \equiv_\Delta Q'$. By Theorem 3 $Q'' \equiv_\Delta Q'$ iff Q'' is generated by the algorithm **new_queries** (line 2 of Algorithm 5). On the other hand, since $Q'' \equiv \text{chase}_\Sigma(Q)$ then $Q'' \equiv_\Sigma Q$. It is shown in [7] that $Q'' \equiv_\Sigma Q$ iff Q is generated by the algorithm **rewrite**$_\Sigma$ (line 4 of Algorithm 5).

□

We note that since UINDs are the most prevalent INDs in practical applications, the above completeness result applies to a majority of the possible queries.

5.3. Acyclic Inclusion Dependencies

Another class of INDs for which a complete rewriting algorithm can be devised, when INDs are considered together with FDs, is the class of acyclic INDs (AINDs). AINDs are defined as follows.

Definition 10 A set Δ of INDs is acyclic if there is no sequence $R_i[\mathcal{A}_i] \subseteq S_i[\mathcal{B}_i]$, $1 \leq i \leq n$, of INDs in Δ, where for $1 \leq i \leq n$, $R_{i+1} = S_i$ for $1 \leq i \leq n-1$ and $R_1 = S_n$.

It was shown in [5] that the implication problem for AINDs and FDs is decidable. Thus, the following theorem allows us to construct a complete algorithm for rewritings with FDs and AINDs.

Theorem 9

$$chase_{\Delta\cup\Sigma}(Q) \equiv chase_{\Sigma}(chase_{\Delta}(Q))$$

Proof. Note that since INDs are acyclic, chase$_\Delta$ will terminate (that is not the case, in general, for UINDs) after all AINDs were applied to Q. Chase$_\Sigma$ does not add any new atoms to Q, it may only change arguments of the exisiting atoms. Thus, applying chase$_\Delta$ to the modified atoms would generate atoms already in Q (a change of a variable through chase$_\Sigma$ is applied consistently to all atoms in Q). □

A version of Algorithm 5, in which the computation of Σ-equivalent queries (line 4) and Δ-equivalent queries (line 2) are reversed, can be shown to be complete for computing all queries $\Delta \cup \Sigma$-equivalent to a given query.

6. CONCLUSIONS

The results of this paper provide a solution to the problem of query rewriting for conjunctive queries and views in the presence of inclusion dependencies. This expands the practical scope of query rewriting, since the use of inclusion dependencies enables the generation of rewritings that might otherwise be overlooked. The paper also addresses the issue of query rewriting in the presence of both functional and inclusion dependencies, thus providing a solution to query rewriting with respect to the most ubiquitous integrity constraints enforced on databases.

Ongoing work involves desigining algorithms for other types of integrity constraints and combining them with the results from this paper.

REFERENCES

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. PODS*, pp. 254–263 (1998).

[2] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. Inclusion dependencies and their interaction with functional dependencies. *Journal of Computer and System Sciences*, **28**(1):29–59 (1984).

[3] A. Chandra and M. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM Journal of Computing*, **14**(3):671–677 (1985).

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proceedings of the 11th ICDE*, pp. 190–200 (1995).

[5] S. Cosmadakis and P. Kanellakis. Functional and inclusion dependencies: A graph theoretic approach. In P. Kanellakis and F. Preparata, editors, *Advances in Computing Research, vol 3: The Theory of Databases*, pp. 164–185. JAI Press, Greenwich, CT (1986).

[6] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proceedings of 22nd VLDB*, pp. 330–341 (1996).

[7] O. Duschka and A. Levy. Recursive plans for information gathering. In *Proc. of 15th IJCAI*, Nagoya, Japan (1997).

[8] J. Gryz. Query folding with inclusion dependencies. In *Proceedings of 14th ICDE*, pp. 126–133, Orlando (1998).

[9] 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).

[10] P. Kanellakis, S. Cosmadakis, and M. Vardi. Unary inclusion dependencies have polynomial-time inference problem. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pp. 246–277 (1983).

[11] A. M. Keller and J. Basu. A predicate-based scheme for client-server database architectures. In *Proceedings of the International Conference on Parallel and Distributed Information Systems* (1994).

[12] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS*, pp. 95–104 (1995).

[13] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query-answering algorithms for information agents. In *Proceedings of 13th AAAI Conference* (1996).

[14] A. Y. Levy, A. Rajaraman, and J.J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. 22nd VLDB*, pp. 251–262 (1996).

[15] J.W. Lloyd. *Foundations of Logic Programming.* Springer–Verlag, second edition (1987).

[16] D. Maier. *The theory of relational databases.* Computer Science Press, Rockville, Md (1983).

[17] X. Qian. Query folding. In *Proceedings of the* 12^{th} *ICDE*, pp. 48–55 (1996).

[18] A. Rajaraman, Y. Sagiv, and J. D. Ullman. Answering queries using templates with binding patterns. In *Proceedings of the 14th PODS*, pp. 105–112 (1995).

[19] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proceedings of 20th VLDB*, pp. 367–378 (1994).

[20] J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Principles of Computer Science Series. Computer Science Press, Rockville, Maryland 20850 (1988).

[21] J.D. Ullman. Information integration using logical views. *Proceedings of ICDT*, pp. 19–40 (1997).

[22] H. Z. Yang and P.-Å. Larson. Query transformation for PSJ-queries. In *Proceedings of 13th VLDB*, pp. 245–254 (1987).