

Rewriting Unions of General Conjunctive Queries Using Views^{*}

Junhu Wang^{1,2}, Michael Maher^{1,3}, and Rodney Topor¹

¹ CIT, Griffith University, Brisbane, Australia 4111

{jwang, rwt, mjm}@cit.gu.edu.au

² GSCIT, Monash University, Churchill, Australia 3842

John.Wang@infotech.monash.edu.au

³ DMCS, Loyola University, Chicago, USA

mjm@math.luc.edu

Abstract. The problem of finding contained rewritings of queries using views is of great importance in mediated data integration systems. In this paper, we first present a general approach for finding contained rewritings of unions of conjunctive queries with arbitrary built-in predicates. Our approach is based on an improved method for testing conjunctive query containment in this context. Although conceptually simple, our approach generalizes previous methods for finding contained rewritings of conjunctive queries and is more powerful in the sense that many rewritings that can not be found using existing methods can be found by our approach. Furthermore, nullity-generating dependencies over the base relations can be easily handled. We then present a simplified approach which is less complete, but is much faster than the general approach, and it still finds maximum rewritings in several special cases. Our approaches compare favorably with existing methods.

1 Introduction

The problem of rewriting queries using views (aka query folding [1]) is of key importance in mediated data integration systems. In such systems, users are usually presented with a uniform interface through which queries are submitted. The uniform interface, also called the *global schema*, consists of a set of *virtual relations* (aka *base relations*) which may not be physically stored. The actual data sources (i.e, the stored data) are regarded as logical views defined on the virtual relations [2]. Thus in order to answer a user query, the system must first rewrite the query into one that is defined on the views only. In other words, given a query Q defined on the base relations we need to find a query Q_r defined on the view relations such that Q_r gives correct answers to Q . If so, Q_r is called a rewriting of Q . Usually two types of rewritings are sought: equivalent rewritings and contained rewritings. Equivalent rewritings are those that give exactly the same set of answers to the original query. Contained rewritings are those that

^{*} This work is partially supported by the Australian Research Council.

give possibly only part of the answers to the original query. In this paper, we will focus on the latter. More specifically, we will study the problem of finding contained rewritings using views when the views are conjunctive queries with arbitrary built-in predicates (which we call *general conjunctive queries (GCQs)*) and the user query is a union of general conjunctive queries (referred to as a *union query*). The rewritings we obtain are union queries. Our attention is focused on how to quickly find rewritings that give as many correct answers as possible, rather than on how efficiently the rewritings can be evaluated.

1.1 Previous Work

Apparently the first papers dealing with query rewriting using views are [3] and [4], in which equivalent rewritings of conjunctive queries are discussed. Over the past few years, the problem has received intensive attention mainly because of its relevance to data integration and query optimization. For a comprehensive survey, see [5]. Here we only mention a few papers that are closely related to our work. Among the early algorithms, the *U-join* algorithm [1] and the *Bucket* algorithm [6] are used to find rewritings of conjunctive queries using conjunctive views (in the Bucket algorithm, the views and the query may contain comparison predicates such as $x < a$, $y \neq x$), and the *Inverse-rule* algorithm [7] was proposed for rewriting Datalog programs using Datalog views. More recently, the *MiniCon* algorithm [8] and the *Shared-Variable-Bucket* (hereafter abbreviated as *SVB*) algorithm [9] have been developed as faster (but less powerful) versions of the Bucket algorithm. Algorithms for finding contained rewritings in the presence of functional dependencies, inclusion dependencies, and full dependencies are studied in [10, 11].

1.2 The Problem and Our Contribution

The problem we study is the rewriting of unions of general conjunctive queries. We also consider the case where *nullity-generating dependencies (NGDs)* (see the next section for definition) over the base relations are present. As mentioned above, several previous algorithms consider rewritings of conjunctive queries with or without built-in predicates. However, for a union query such as $Q_u = Q_1 \cup Q_2$, it is not enough to find the rewritings for each of the conjunctive queries and then union them. It is possible that a conjunctive query defined on the views is not a rewriting of any of the conjunctive queries in the union, but it is a rewriting of the union. For instance, the rewriting of Q_u in the next example is not a rewriting of either Q_1 or Q_2 .

Example 1. Let $Q_u = Q_1 \cup Q_2$, where Q_1 and Q_2 are

$$\begin{aligned} q(y) &:- p(x, y), p'(x', y), x > y \text{ and} \\ q(y) &:- p(x, y), r(y, z), x \leq y \text{ respectively.} \end{aligned}$$

Let the views be

$$\begin{aligned} v_1(y) &:- p(x, y), \\ v_2(y) &:- r(y, x), \text{ and} \end{aligned}$$

$$v_3(y) :- p'(x, y).$$

Then $q(y) :- v_1(y), v_2(y), v_3(y)$ is a contained rewriting of Q_u , but not of Q_1 , nor Q_2 . The rewriting is not a union of the rewritings of Q_1 and Q_2 .

Existing algorithms may fail to find all contained rewritings even if the query is a conjunctive query, as shown in the next example.

Example 2. Suppose all relation attributes in this example are from the integers. Let the query Q be $q(x) :- p_1(x, y), p_2(x, y)$.

Let the views be

$$v_1(x) :- p_1(x, y), y > 0, y < 3,$$

$$v_2(x) :- p_2(x, 1), \text{ and}$$

$$v_3(x) :- p_2(x, 2).$$

It can be verified that $q(x) :- v_1(x), v_2(x), v_3(x)$ is a contained rewriting of Q , but this rewriting can not be found by existing algorithms.

Although the Inverse-rule algorithm in [7] considers rewritings of Datalog queries, it does not consider the case where built-in predicates are present. The presence of general-form NGDs was not considered by previous work.

In this paper, we first revise a previous result on query containment to one that does not require the queries to be in any normal form. This revised result is used extensively in the analysis of our rewritings. We then present a general approach for rewriting union queries using views which, for instance, can find the rewritings in Examples 1 and 2. Furthermore, NGDs can be handled easily. We then present a simplified version of the general approach (referred to as the *simplified approach*) which is less complete but significantly more efficient. When used in rewriting GCQs, our simplified approach finds strictly more rewritings than the MiniCon and the SVB algorithms when built-in predicates exists in the query, and it finds maximum rewritings (see the next section for definition) in several special cases. Also in these special cases, the simplified approach compares favorably with the MiniCon and the SVB algorithms in terms of efficiency. When there are no built-in predicates in the query and the views, the simplified approach finds the same rewritings as the U-join algorithm with similar costs.

The rest of the paper is organized as follows. Section 2 provides the technical background. In Section 3 we introduce the concept of *relevance mappings* and present the improved method for testing query containment. In Section 4 we present our general approach on rewriting union queries. The simplified approach is described in Section 5. Section 6 compares our approaches with the Bucket, the U-Join, the MiniCon and the SVB algorithms. Section 7 concludes the paper with a summary and a discussion about further research.

2 Preliminaries

2.1 General Conjunctive Queries and Union Queries

A *general conjunctive query (GCQ)* is of the form

$$q(X) :- p_1(X_1), \dots, p_n(X_n), C \tag{1}$$

where q, p_1, \dots, p_n are relation names (q is distinct from p_1, \dots, p_n), X, X_1, \dots, X_n are tuples of variables and constants, C is a conjunction of atomic constraints over the variables in $X \cup X_1 \cup \dots \cup X_n$. We call $q(X)$ the *head*, $p_1(X_1), \dots, p_n(X_n), C$ the *body*, and C the *constraint*¹ of the query. Each atom $p_i(X_i)$ ($i = 1, \dots, n$) is called a *subgoal*, and each atomic constraint in C is called a *built-in predicate*. The tuple X is called the *output* of the query. The variables in the head and those that are equated, by C , to some head variables or constants are called *distinguished variables*.

We make the following safety assumptions about the GCQs

1. There is at least one atom in the body.
2. Every variable in X either appears explicitly as an argument in a subgoal, or is (implicitly or explicitly) equated, by C , to a variable in at least one of the subgoals or to a constant.

Two GCQs are said to be *comparable* if they have the same number of output arguments and the corresponding output arguments are from the same domains. A *union query* is a finite union of comparable GCQs. Clearly, a GCQ can be regarded as a union query. In what follows, when we say a *query*, we mean a GCQ or a union query.

Query containment and equivalence are defined in the usual way. We will use $Q_1 \sqsubseteq Q_2$ and $Q_1 = Q_2$ to denote Q_1 is contained in Q_2 and Q_1 is equivalent to Q_2 respectively. We will use *empty query* to refer to any query whose answer set is empty for any database instance. Clearly, a GCQ is empty if and only if its constraint is unsatisfiable.

A GCQ is said to be in *normal form*, if the arguments in every atom (head or subgoal) are distinct variables only, and the sets of variables in different atoms are pairwise disjoint. A GCQ is said to be in *compact form* if there are no explicit or implicit *basic equalities* (i.e, non-tautological equalities between two variables or between a variable and a constant, e.g, $x \geq y \wedge x \leq y$, where $x = y$ is not a tautology) in the constraint. Clearly, every GCQ can be put into normal form. It can be put into compact form provided we can find all the implicit equalities in the constraint.

2.2 Nullity-Generating Dependencies

A *nullity-generating dependency (NGD)* is a formula of the form

$$r_1(X_1), \dots, r_m(X_m), D \rightarrow FALSE \quad (2)$$

where r_1, \dots, r_m are relation names, X_1, \dots, X_m are tuples of variables and constants, and D is a constraint over the variables in $X_1 \cup \dots \cup X_m$.

Functional dependencies and equality-generating dependencies are special cases of NGDs.

¹ Note that the constraint of a GCQ refers to built-in predicates, rather than integrity constraints.

Let Δ be a set of NGDs. If for any database instance \mathcal{I} which satisfies the NGDs in Δ , the answer set of Q_1 is a subset of that of Q_2 , then we say Q_1 is *contained in Q_2 under Δ* , denoted $Q_1 \sqsubseteq_{\Delta} Q_2$.

2.3 Rewritings and Maximum Rewritings

We assume the existence of a set of base relations and a set W of *views*. A *view* is a GCQ defined on the base relations. Without loss of generality, we assume the arguments in the head of a view are *distinct variables* only. We refer to the relation in the head of the view as the *view relation*.

There are two world assumptions [12]: under the *closed world assumption*, the view relation stores all of the answers to the view; under the *open world assumption*, the view relation stores possibly only part of the answers to the view. The open world assumption is usually used in data integration [5] because it is usually not known that all answers of the views are actually stored. In this paper, we will use the open world assumption.

For any *base instance* \mathcal{D} consisting of instances of the base relations, we use $W(\mathcal{D})$ to denote a *view instance* (with respect to \mathcal{D}) consisting of instances of the view relations. Since the open world assumption is used, each relation instance in $W(\mathcal{D})$ may contain only part of the answers computed to the corresponding view using \mathcal{D} .

Given a union query Q_u defined on the base relations, our task is to find a query Q_r defined solely on the view relations such that, for any base instance \mathcal{D} , all of the answers to Q_r computed using any view instance $W(\mathcal{D})$ are correct answers to Q_u . We call such a query Q_r a *contained rewriting* or simply a *rewriting* of Q_u . If Q_r does not always give the empty answer set, we call it a *non-empty* rewriting.

To check whether a query Q_r is a rewriting of Q_u , we need the *expansion* of Q_r , as defined below.

Definition 1. *If Q_r is a GCQ defined on the view relations, then the expansion Q_r^{exp} of Q_r is the GCQ obtained as follows: For each subgoal $v(x_1, \dots, x_k)$ of Q_r , suppose $v(y_1, \dots, y_k)$ is the head of the corresponding view V , and σ is a mapping that maps the variable y_i to the argument x_i for $i = 1, \dots, k$, and maps every non-head variable in V to a distinct new variable, then*

- (1) *replace $v(x_1, \dots, x_k)$ with the body of $\sigma(V)$,*
- (2) *now if a variable x_i ($1 \leq i \leq k$) appears only in the constraint, then replace x_i with the variable in the subgoals of $\sigma(V)$ (or the constant) to which x_i is equated by the constraint of $\sigma(V)$.*

The expansion Q_u^{exp} of a union query Q_u is the union of the expansions of the GCQs in Q_u .

For example, if Q_r is $q(x) :- v(y), x = y$, and V is $v(y) :- p(z), y = z$, then Q_r^{exp} is $q(x) :- p(z), x = z$.

There may be many different rewritings of a query. To compare them, we define maximum rewritings.

Definition 2. A rewriting Q_1 of Q is said to be a maximum rewriting with respect to a query language \mathcal{L} if for any rewriting Q_2 of Q in \mathcal{L} , every answer to Q_2 is an answer to Q_1 for any view instance $W(\mathcal{D})$ with respect to any base instance \mathcal{D} .

Note that for a rewriting Q_1 to be maximum under the open world assumption, it is not enough that $Q_2^{exp} \sqsubseteq Q_1^{exp}$ holds for any other rewriting Q_2 . Note also that the condition for a maximum rewriting is slightly stronger than that for a *maximally contained rewriting* in [8, 9], and that for a *maximally contained retrievable program* in [7], and that for a *maximally contained query plan* in [11].

The above definition of rewritings extends straightforwardly to the case where NGDs on the base relations exist. In this case, the rewriting is called a *semantic rewriting*.

Definition 3. Let Δ be a set of NGDs on the base relations, Q be a query defined on the base relations, and W be a set of views. If Q_r is a query defined on the view relations in W such that $Q_r^{exp} \sqsubseteq_{\Delta} Q$, then we say Q_r is a semantic rewriting of Q wrt W and Δ .

2.4 Inverse Rules and Inferred Constraints

Given a view V :

$$v(X) :- p_1(X_1), \dots, p_n(X_n), C$$

we can compute a set of *inverse rules* [7]: First, replace each non-distinguished variable in the body with a distinct Skolem function. The resulting view is said to be *Skolemized*. Suppose ρ is the mapping that maps the non-distinguished variables to the corresponding Skolem functions, then the inverse rules are

$$\rho(p_i(X_i)) \leftarrow v(X) \text{ (for } i = 1, \dots, n).$$

The left side of an inverse rule is called the *head*, and the right side is called the *body*. A variable in an inverse rule is said to be *free* if it appears as an independent argument of the head, that is, it appears in the head, and appears not only inside the Skolem functions. In addition, we will call $\rho(C)$ the *inferred constraint* of the atom $v(X)$.

Example 3. For the view

$$v(x, z) :- p_1(x, y), p_2(y, z), x > y,$$

there are two inverse rules:

$$p_1(x, f(x, z)) \leftarrow v(x, z) \text{ and}$$

$$p_2(f(x, z), z) \leftarrow v(x, z).$$

In the first inverse rule x is a free variable, but z is not. In the second one, z is a free variable, but x is not. The inferred constraint of $v(x, z)$ is $x > f(x, z)$.

If we have more than one view, we can generate a set of inverse rules from each of them. In this case, the inverse rules generated from different views must use different Skolem functions.

In the sequel, when we say a *rule*, we mean an inverse rule. For simplicity, we also assume the rules are compact as defined below.

Definition 4. *The set of rules generated from a view is said to be compact, if the inferred constraint does not imply a non-tautological equality between a constant and a Skolem function, or between a constant and a variable, or between a variable and a Skolem function, or between two Skolem functions, or between two variables.*

Clearly, if the views are in compact form, then the rules generated will be compact.

3 An Improved Method for Testing Query Containment

Let us use $Var(Q)$ (resp. $Arg(Q)$) to denote the set of variables (resp. the set of variables and constants) in a GCQ Q . A *containment mapping* from a GCQ Q_2 to another GCQ Q_1 is a mapping from $Var(Q_2)$ to $Arg(Q_1)$ such that it maps the output of Q_2 to the output of Q_1 , and maps each subgoal of Q_2 to a subgoal of Q_1 .

The following lemma relates query containment to the existence of some particular containment mappings [13].

Lemma 1. *Let Q_i ($i = 0, \dots, s$) be GCQs. Let C_i be the constraints in Q_i .*

(1) *If there are containment mappings $\delta_{i,1}, \dots, \delta_{i,k_i}$ from Q_i to Q_0 such that $C_0 \rightarrow \bigvee_{i=1}^s \bigvee_{j=1}^{k_i} \delta_{i,j}(C_i)$, then $Q_0 \sqsubseteq \bigcup_{i=1}^s Q_i$.*

(2) *If Q_1, \dots, Q_s are in normal form, C_0 is satisfiable, and $Q_0 \sqsubseteq \bigcup_{i=1}^s Q_i$, then there must be containment mappings $\delta_{i,1}, \dots, \delta_{i,k_i}$ from Q_i to Q_0 such that $C_0 \rightarrow \bigvee_{i=1}^s \bigvee_{j=1}^{k_i} \delta_{i,j}(C_i)$.*

The condition that Q_i ($i = 1, \dots, s$) are in normal form in (2) of Lemma 1 is necessary even if C_0 is a tautology. This is demonstrated in the next example.

Example 4. Let Q_1 and Q_2 be

$h(w) :- q(w), p(x, y, 2, 1, u, u), p(1, 2, x, y, u, u), p(1, 2, 2, 1, x, y)$ and

$h(w) :- q(w), p(x, y, z, z', u, u), x < y, z > z'$, respectively. Suppose all relation attributes are from the reals. There are only two containment mappings from Q_2 to Q_1 :

$\delta_1 : w \rightarrow w, x \rightarrow x, y \rightarrow y, z \rightarrow 2, z' \rightarrow 1, u \rightarrow u$ and

$\delta_2 : w \rightarrow w, x \rightarrow 1, y \rightarrow 2, z \rightarrow x, z' \rightarrow y, u \rightarrow u$

Clearly $TRUE \not\rightarrow \delta_1(x < y, z > z') \vee \delta_2(x < y, z > z')$, but $Q_1 \sqsubseteq Q_2$.

Thus in order to use Lemma 1, we need to put all of the queries Q_1, \dots, Q_s into normal form. This sometimes makes the application of Lemma 1 inconvenient. Next, we present a revised method for testing query containment using *relevance mappings*.

Before introducing relevance mappings, we need the concept of *targets*.

Definition 5. *Let Q_2 be the GCQ $q(X) :- p_1(X_1), \dots, p_n(X_n), C$, and Q_1 be a GCQ comparable to Q_2 .*

A target T of Q_2 in Q_1 is a formula $q'(Y) :- p'_1(Y_1), \dots, p'_n(Y_n)$ such that

1. $q'(Y)$ is the head of Q_1 , and for each $i \in \{1, \dots, n\}$, $p'_i(Y_i)$ is a subgoal of Q_1 with the same relation name as that of $p_i(X_i)$.
2. If we denote the sequence of all arguments in Q_2 by $S_2 = (x_1, x_2, \dots, x_m)$ and denote the sequence of all arguments in T by $S_1 = (y_1, y_2, \dots, y_m)$, then none of the following holds:
 - (a) There is a position i such that x_i and y_i are two different constants.
 - (b) There are two positions i and j such that x_i and x_j are two different constants, but y_i and y_j are the same variable.
 - (c) There are two positions i and j such that y_i and y_j are two different constants, but x_i and x_j are the same variable.

We now give a constructive definition of relevance mappings and their associated equalities.

Definition 6. Let T be a target of Q_2 in Q_1 . Let $S_2 = (x_1, x_2, \dots, x_m)$ and $S_1 = (y_1, y_2, \dots, y_m)$ be the sequences of arguments in Q_2 and T respectively. The relevance mapping δ from Q_2 to Q_1 wrt T and its associated equality E_δ are constructed as follows: Initially, $E_\delta = \text{TRUE}$. For $i = 1$ to m

1. If x_i is a constant α , but y_i is a variable y , then let $E_\delta = E_\delta \wedge (y = \alpha)$.
2. If x_i is a variable x , and x appears the first time in position i , then let δ map x to y_i . If x appears again in a later position j ($> i$) of S_2 , and $y_j \neq y_i$, then let $E_\delta = E_\delta \wedge (y_j = y_i)$.

Relevance mappings are closely related to containment mappings. Any containment mapping is a relevance mapping with the associated equality being a tautology, and a relevance mapping is a containment mapping iff its associated equality is a tautology.

If δ is a relevance mapping from Q_2 to Q_1 wrt T , then we will use $\delta(Q_2)$ to denote the query obtained by applying δ to Q_2 , and use $T \wedge E_\delta \wedge \delta(C_2)$ to denote the query $q'(Y) :- p'_1(Y_1), \dots, p'_n(Y_n), E_\delta \wedge \delta(C_2)$, where C_2 is the constraint of Q_2 . Clearly $\delta(Q_2)$ is equivalent to $T \wedge E_\delta \wedge \delta(C_2)$, and it is contained in Q_2 . Let us call $\delta(Q_2)$ an *image* of Q_2 in Q_1 . The next lemma implies that Q_1 is contained in Q_2 if and only if Q_1 is contained in the union of all of the images of Q_2 in Q_1 .

Lemma 2. Let C_i be the constraint in the GCQ Q_i ($i = 0, 1, \dots, s$). Suppose C_0 is satisfiable. Then $Q_0 \sqsubseteq \cup_{i=1}^s Q_i$ iff there are relevance mappings $\delta_{i,1}, \dots, \delta_{i,k_i}$ from Q_i to Q_0 such that $C_0 \rightarrow \bigvee_{i=1}^s \bigvee_{j=1}^{k_i} (\delta_{i,j}(C_i) \wedge E_{\delta_{i,j}})$, where $E_{\delta_{i,j}}$ is the associated equality of $\delta_{i,j}$.

Note there is no need to put Q_1, \dots, Q_s in any normal form. The next example demonstrates the application of Lemma 2.

Example 5. For the queries in Example 4, there is a third relevance mapping δ_3 from Q_2 to Q_1 in addition to δ_1 and δ_2 :

$\delta_3 : w \rightarrow w, x \rightarrow 1, y \rightarrow 2, z \rightarrow 2, z' \rightarrow 1, u \rightarrow x$, with $E_{\delta_3} = (x = y)$.
Since $\delta_1(x < y, z > z') = x < y, \delta_2(x < y, z > z') = x > y$ and $\delta_3(x < y, z > z') = \text{TRUE}$, and $x < y \vee x > y \vee x = y$ is always true, we know $Q_1 \sqsubseteq Q_2$.

Lemma 2 has an additional advantage over Lemma 1: the number of mappings we have to consider can be drastically reduced in some cases. The next example shows one of such cases.

Example 6. Let Q_2 and Q_1 be

$$q(x, y) :- p(x, y, 0), p(x, y, z), p(x, y', z), x \leq y, z < 10 \text{ and}$$

$$q(x, y) :- p(x, y, 1), p(x, y, 2), \dots, p(x, y, N), x < y, z < 1 \text{ respectively.}$$

If we use Lemma 1 to test whether $Q_1 \sqsubseteq Q_2$, then we need to put Q_2 into normal form, and consider 3^N containment mappings from Q_2 to Q_1 . If we use Lemma 2, we can see $Q_1 \not\sqsubseteq Q_2$ immediately because there are obviously no relevance mappings from Q_2 to Q_1 .

Similarly, query containment under NGDs can be characterized by relevance mappings. Given a NGD ic as in (2) and a GCQ Q as in (1), we can construct *relevance mappings* and the *associated equalities* from ic to Q in a way similar to what we use in constructing relevance mappings from one GCQ to another. The only difference is that a *target* of ic in Q is defined to be a sequence of m subgoals p'_1, \dots, p'_m in Q such that (1) p'_i and $r_i(X_i)$ ($i = 1, \dots, m$) have the same relation name, (2) a constant in $r_i(X_i)$ corresponds either to a variable or the same constant in p'_i , (3) no two occurrences of the same variable in p'_1, \dots, p'_m correspond to two different constants in $r_1(X_1), \dots, r_m(X_m)$ and vice versa.

The next lemma is revised from a result in [14].

Lemma 3. *Let $\Delta = \{ic_i \equiv P_i, D_i \rightarrow FALSE \mid i = 1, \dots, t\}$ be a set of NGDs, and Q_1, \dots, Q_s be GCQs. Suppose the constraint of Q_i is C_i , and C_0 is satisfiable. Then $Q_0 \sqsubseteq_{\Delta} \cup_{i=1}^s Q_i$ iff there are relevance mappings $\delta_{i,1}, \dots, \delta_{i,m_i}$ from Q_i to Q_0 ($i = 1, \dots, s$) and relevance mappings $\rho_{i,1}, \dots, \rho_{i,k_i}$ from ic_i to Q_0 ($i = 1, \dots, t, m_1 + \dots + m_s + k_1 + \dots + k_t > 0$) such that*

$$C_0 \rightarrow \bigvee_{i=1}^s \bigvee_{j=1}^{m_i} (\delta_{i,j}(C_i) \wedge E_{\delta_{i,j}}) \bigvee \bigvee_{i=1}^t \bigvee_{j=1}^{k_i} (\rho_{i,j}(D_i) \wedge E_{\rho_{i,j}}).$$

4 The General Approach for Rewriting Union Queries

Let Q_u be the union query to be rewritten. Without loss of generality, we assume all the GCQs in Q_u have the same head. Our method for rewriting Q_u consists of two major steps. In the first step, we generate a set of *potential formulas* (or *p-formulas* for short) which may or may not be rewritings; these p-formulas are generated separately for every GCQ in Q_u . In the second step, we combine all these p-formulas to see whether we can obtain correct rewritings.

4.1 Generating p-formulas

We assume the compact set IR of inverse rules has been computed in advance. To generate a p-formula for a GCQ, we need to find a *destination* first.

Definition 7. *Given the GCQ Q as in (1) and a set IR of compact inverse rules, a destination of Q wrt to IR is a sequence DS of n atoms $DS = p_1(Y_1), \dots, p_n(Y_n)$ such that*

1. Each atom $p_i(Y_i)$ is the head of some rule, and it has the same relation name as that of $p_i(X_i)$, the i th subgoal of Q .
2. There is no i such that a constant in $p_i(X_i)$ corresponds to a different constant in $p_i(Y_i)$.
3. No two occurrences of the same variable in Q correspond to two different constants in DS , and no two occurrences of the same variable or Skolem function in the same rule head correspond to two different constants in Q .

Intuitively, a destination “connects” the subgoals of the query to the view atoms in a rewriting. Once a destination DS of Q is found, we can use it to (try to) construct a p-formula as follows:

1. For each atom $p_i(Y_i)$ in DS , do the following:
 Suppose the arguments in $p_i(Y_i)$ are y_1, y_2, \dots, y_l , and the corresponding arguments in $p_i(X_i)$ are x_1, x_2, \dots, x_l . Suppose $p_i(Y_i)$ is the head of the rule $p_i(Y_i) \leftarrow v_i(Z_i)$ (If there are rules that have the same head but different bodies, then choose one of them in turn to generate different p-formulas).
 Define a variable mapping ϕ_i as follows: For each free variable $z \in Y_i$, if z first appears (checking the argument positions from left to right) at position i , then map z to x_i . For each variable z in $v(Z_i)$ which does not appear in $p_i(Y_i)$ as a free variable, let ϕ_i map z to a distinct new variable not occurring in Q or any other view atom $\phi_j(v_j(Z_j))$ ($j \neq i$).
2. Construct a formula $T: q(X) :- \phi_1(p_1(Y_1)), \dots, \phi_n(p_n(Y_n))$.
 Regard T as a target of Q and construct the relevance mapping δ from Q wrt to T (Skolem functions in the target are treated in the same way other arguments in the target are treated). We will get a GCQ

$$q(X) :- \phi_1(p_1(Y_1)), \dots, \phi_n(p_n(Y_n)) \wedge \delta(C) \wedge E_\delta \quad (F)$$

3. Replace $\phi_i(p_i(Y_i))$ with $\phi_i(v_i(Z_i))$ (for $i = 1, \dots, n$), in the above GCQ to get the formula

$$q(X) :- \phi_1(v_1(Z_1)), \dots, \phi_n(v_n(Z_n)), \delta(C) \wedge E_\delta \quad (PF)$$

4. Suppose the inferred constraints of $v_1(Z_1), \dots, v_n(Z_n)$ are C_1, \dots, C_n respectively. If the constraint $\delta(C) \wedge E_\delta \wedge \phi_1(C_1) \wedge \dots \wedge \phi_n(C_n)$ is satisfiable, then output the formula (PF) (remove duplicate atoms if possible).

The formula (PF) is the p -formula of Q we get. Any p-formula of a GCQ in the union Q_u is called a p -formula of Q_u . The p-formula (PF) has the following property: If we replace each view atom with the corresponding Skolemized view body and treat the Skolem functions as variables, then we will get a safe GCQ Q'' (hereafter referred to as the *expansion* of (PF) , denoted $(PF)^{exp}$) which is contained in Q . This is because (F) is a safe GCQ which is equivalent to $\delta(Q)$, and all subgoals and built-in predicates of (F) are in the body of Q'' .

Lemma 4. *The expansion of a p-formula of Q is a safe GCQ contained in Q .*

Thus if there happen to be no Skolem functions in the p-formula, then the formula is a rewriting of Q .

Theorem 1. *For any query Q , if there are no Skolem functions in a p-formula of Q , then the p-formula is a rewriting of Q .*

However, if there are Skolem functions in $E_\delta \wedge \delta(C)$, then the formula (PF) is not a correct GCQ because the Skolem functions appear only in the constraint part $\delta(C) \wedge E_\delta$, and their values can not be determined. So (PF) is not a rewriting if it contains Skolem functions.

Example 7. Let the query be

$$q(u) :- p'(u), p(x, y), r(y, v), x < y, y < v + 1.$$

Let the views be

$$\begin{aligned} v_1(u) &:- p'(u), \\ v_2(y, z) &:- p(x, y), p(y, z), x < z, \text{ and} \\ v_3(y, z) &:- r(x, y), r(y, z), x < z. \end{aligned}$$

The compact inverse rules are:

$$\begin{aligned} \text{R1: } &p'(u) \leftarrow v_1(u) \\ \text{R2: } &p(f(y, z), y) \leftarrow v_2(y, z) \\ \text{R3: } &p(y, z) \leftarrow v_2(y, z) \\ \text{R4: } &r(g(y, z), y) \leftarrow v_3(y, z) \\ \text{R5: } &r(y, z) \leftarrow v_3(y, z) \end{aligned}$$

There are four destinations:

- (1) $p'(u), p(f(y, z), y), r(y, z)$
- (2) $p'(u), p(y, z), r(y, z)$
- (3) $p'(u), p(f(y, z), y), r(g(y, z), y)$
- (4) $p'(u), p(y, z), r(g(y, z), y)$

For the first destination, we first define the mappings

$$\phi_1 : u \rightarrow u; \phi_2 : y \rightarrow y; \text{ and } \phi_3 : y \rightarrow y, z \rightarrow v.$$

and construct the target

$$q(u) :- p'(u), p(f(y, z), y), r(y, v).$$

Then we obtain the image of Q wrt the above target

$$q(u) :- p'(u), p(f(y, z), y), r(y, v), f(y, z) < y, y < v + 1.$$

Finally we replace $p'(u), p(f(y, z), y), r(y, v)$ with $v_1(u), v_2(y, z), v_3(y, v)$ to get the p-formula

$$q(u) :- v_1(u), v_2(y, z), v_3(y, v), f(y, z) < y, y < v + 1.$$

Similarly, for the second destination (2), we can get the p-formula

$$q(u) :- v_1(u), v_2(x, y), v_3(y, v), x < y, y < v + 1.$$

The second p-formula is a rewriting because it involves no Skolem functions.

4.2 Obtaining Rewritings from the p-formulas

As noted earlier, when there are Skolem functions in a p-formula, the p-formula is not a rewriting. However, it is possible that such p-formulas can be combined to obtain correct rewritings. Generally, the following two steps are needed.

First, we choose some p-formulas and combine them into a single formula. Suppose we have chosen k p-formulas PF_1, \dots, PF_k , where PF_i is of the form

$$q(X) :- v_{i,1}(Z_{i,1}), \dots, v_{i,n_i}(Z_{i,n_i}), C_i.$$

Then the combined formula (CF) is

$$q(X) :- v_{1,1}(Z_{1,1}), \dots, v_{1,n_1}(Z_{1,n_1}), \dots, v_{k,1}(Z_{k,1}), \dots, v_{k,n_k}(Z_{k,n_k}), C_1 \vee \dots \vee C_k$$

where the variables which appear only in the view atoms (not in $q(X)$) of different p-formulas should be renamed to different variables.

Second, for the above combined formula (CF), we try to remove those constraints that involve Skolem functions, or to replace them with another constraint (over the variables in the ordinary atoms of the formula) that does not involve Skolem functions. Generally, we need to utilize the inferred constraints of view atoms as follows.

Let D be the conjunction of the inferred constraints of the view atoms in (CF). Write the constraint $C_1 \vee \dots \vee C_k$ into conjunctive normal form, and divide it into the conjunction of C' and C'' , where C' involves Skolem functions, but C'' does not. If there exists a constraint D' over the variables in $X, Z_{1,1}, \dots, Z_{k,n_k}$ such that $D \wedge D' \wedge C''$ is satisfiable and $D \wedge D' \wedge C'' \rightarrow C'$, then output the following query (CR):

$$q(X) :- v_{1,1}(Z_{1,1}), \dots, v_{1,n_1}(Z_{1,n_1}), \dots, v_{k,1}(Z_{k,1}), \dots, V_{k,n_k}(Z_{k,n_k}), C'' \wedge D'$$

The following theorem is straightforward.

Theorem 2. *The query (CR) computed above, if safe, is a rewriting of Q_u .*

In order to get more rewritings, we should check all possible combinations of the p-formulas. In particular, we should check whether it is possible to get a rewriting from a single p-formula. In addition, the constraint D' should be as weak as possible (for example, when it is possible, choose D' to be *TRUE*), so that the rewriting we obtain is as general as possible.

Let us look at some examples. In Example 8, we get a rewriting from a single p-formula.

Example 8. For the first p-formula in Example 7, the conjunction of the inferred constraints of the view atoms is $f(y, z) < z \wedge g(y, v) < v$. Since $(z \leq y) \wedge (f(y, z) < z) \rightarrow f(y, z) < y$, we can replace $f(y, z) < y$ with $z \leq y$ and get the rewriting

$$q(u) :- v_1(u), v_2(y, z), v_3(y, v), z \leq y, y < v + 1.$$

In the next example, we combine two p-formulas by conjoining their view atoms and “disjuncting” their constraints.

Example 9. For the views in Example 1, the inverse rules are

$$\begin{aligned} p(f_1(y), y) &\leftarrow v_1(y), \\ r(y, f_2(y)) &\leftarrow v_2(y), \text{ and} \\ p'(f_3(y), y) &\leftarrow v_3(y). \end{aligned}$$

For Q_1 , we find the destination $p(f_1(y), y), p'(f_3(y), y)$ and then the p-formula $q(y) :- v_1(y), v_3(y), f_1(y) > y$.

For Q_2 , we find the destination $p(f_1(y), y), r(y, f_2(y))$ and then the p-formula $q(y) :- v_1(y), v_2(y), f_1(y) \leq y$.

Combining the above two p-formulas, we will get the rewriting

$$q(y) :- v_1(y), v_2(y), v_3(y).$$

The rewriting in Example 2 can be found similarly.

There are two remarks about the above general approach.

First, about the completeness of the rewritings. Due to the inherent complexity of the problem (see [12]), we do not expect to find a rewriting which can produce all possible correct answers to the original query using the views. However, we do have the following theorem, which shows that p-formulas are an appropriate basis for performing rewritings.

Theorem 3. *For any non-empty union rewriting Q_r of Q_u , there are p-formulas PF_1, \dots, PF_s of Q_u such that $Q_r^{exp} \sqsubseteq \cup_{i=1}^s PF_i^{exp}$.*

Second, about the complexity of the general approach. The above approach is exponential in the number of views. The step of combining the p-formulas is particularly expensive because the number of possible combinations may grow explosively and the constraint implication problem is intractable in general.

In practice, one can use various simplified versions of the general approach. For example, we may choose to consider some, rather than all of the p-formulas. In Section 5, we focus on such a version which is practically much more efficient, yet it still produces maximum rewritings in some special cases. We can also consider only some, rather than all of combinations (e.g., only combinations of p-formulas which involve common Skolem functions). We can also use a more systematic way for combining the p-formulas. For example, as a rule of thumb, we should check each single p-formula first to see whether we can get rewritings; then combine p-formulas with the same non-constraint part (modulo variable renaming); and finally conjoin the view atoms of chosen p-formulas as stated before.

4.3 Handling Nullity-Generating Dependencies

Suppose there is a set $\Delta = \{ic_i \equiv P_i, D_i \rightarrow FALSE \mid i = 1, \dots, s\}$ of NGDs. Our general method for handling these CGDs is as follows: First, regard every CGD ic_i as an empty query $\emptyset :- P_i, D_1$ (let us call it the *induced query* of ic_i), and find the p-formulas of this query. Second, combine the p-formulas of these induced queries and the p-formulas of Q_u in the same way as before, except that the p-formulas of the induced queries should be combined with at least one p-formula of Q_u and the resulting query should use the head $q(X)$, to find semantic rewritings of Q_u under Δ .

The correctness of the above method is clear from Lemma 3.

The next example is modified from Example 1.

Example 10. Let $Q_u = Q_1 \cup Q_2$, where Q_1 and Q_2 are

$q(y) :- p(x, y), p_1(x_1, y), x > y$ and

$q(y) :- p(x, y), r(y, z), x < y$ respectively.

Let Δ contain $p(x, y), p_2(y), x = y \rightarrow FALSE$ only.

The induced query is $\emptyset :- p(x, y), p_2(y), x = y$.

Let the views be

$v_1(y) :- p(x, y),$

$v_2(y) :- r(y, x),$

$v_3(y) :- p'(x, y),$ and

$v_4(y) :- p_2(y).$

For Q_1 , we find the p-formula $q(y) :- v_1(y), v_3(y), f_1(y) > y$.

For Q_2 , we find the p-formula $q(y) :- v_1(y), v_2(y), f_1(y) < y$.

For the induced query, we find the p-formula $\emptyset :- v_1(y), v_4(y), f_1(y) = y$.

Combine the three p-formulas, we get a combined formula

$q(y) :- v_1(y), v_2(y), v_3(y), v_1(z), v_4(z), f_1(y) > y \vee f_1(y) < y \vee f_1(z) = z$.

Clearly $z = y \rightarrow f_1(y) > y \vee f_1(y) < y \vee f_1(z) = z$, therefore, we can get the semantic rewriting $q(y) :- v_1(y), v_2(y), v_3(y), v_4(y)$.

5 A Simplified Approach

As mentioned earlier, the most difficult part of the general approach is in the p-formula combination step. The simplified approach imposes extra conditions on p-formulas, so that the combination step is simplified.

Naturally, we would like to get as many rewritings as possible from the single p-formulas. We start with the simple case where there are no built-in predicates in the query or the views. In this case, the p-formula (PF) becomes

$$q(X) :- \phi_1(v_1(Z_1)), \dots, \phi_n(v_n(Z_n)), E_\delta.$$

That is, the constraint in the p-formula is E_δ . To make sure that there are no Skolem functions in E_δ , we first require that every distinguished variable x or constant α in the GCQ does not correspond to a Skolem function in the destination DS , otherwise there will be the equality $x = \phi_i(f(Z))$ or $\alpha = \phi_i(f(Z))$ (where $f(Z)$ is the Skolem function in $p_i(Y_i)$ corresponding to x or α) because the GCQ and the target have the same head; we then require that no variable in the GCQ corresponds to two different Skolem functions, otherwise there will be an equality between two different Skolem functions. Even when the query and views do have built-in predicates, we may still want the above requirements for DS in order to reduce the number of p-formulas.

Based on the above analysis, we can define *valid destinations* and *valid p-formulas*. We assume Q is in compact form so that we know all of the distinguished variables and non-distinguished variables.

Definition 8. Given the GCQ Q as in (1) and a set IR of compact rules, a destination DS of Q wrt to IR is said to be a valid destination if

1. Each distinguished variable or constant in $p_i(X_i)$ corresponds to a free variable or to a constant.
2. All occurrences of the same non-distinguished variable in Q correspond either all to free variables and constants, or all to the same Skolem function.

For instance, among the four destinations in Example 7, only the first two are valid destinations.

Once we have found a valid destination DS of Q , we can generate a p-formula as before. Note that even with a valid destination, there may still be equalities of the form $f(Z_1) \equiv \phi_i(f(Z)) = \phi_j(f(Z)) \equiv f(Z_2)$ in the E_δ part of the p-formula. In this case, we can remove the equality $f(Z_1) = f(Z_2)$ by replacing it with $Z_1 = Z_2$. The resulting formula is called a *valid p-formula*.

Example 11. Let the query Q be $q(x, x') :- p(x, y), p(x', y)$.

Let the view V be $v(x) :- p(x, y)$.

The inverse rule is $p(x, f(x)) \leftarrow v(x)$, and $p(x, f(x)), p(x, f(x))$ is a valid destination of Q . Therefore, we can construct a target

$q(x, x') :- p(x, f(x)), p(x', f(x'))$ and then get the p-formula
 $q(x, x') :- v(x), v(x'), f(x') = f(x)$.

Replacing $f(x') = f(x)$ with $x = x'$, we get a valid p-formula

$q(x, x') :- v(x), v(x'), x' = x$

which is equivalent to $q(x, x') :- v(x), x' = x$. This is a rewriting of Q .

Note the valid p-formula may still have Skolem functions if Q has a constraint involving non-distinguished variables. However, if Q does not have constraints, or if the constraint of the GCQ involves distinguished variables only, then it is impossible for any valid p-formula to have Skolem functions. Thus every valid p-formula will be a rewriting.

Theorem 4. *If Q is a GCQ without constraint, or the constraint of Q involves only distinguished variables, then every valid p-formula of Q is a rewriting of Q .*

Obviously, if we limit the destinations to valid destinations, and p-formulas to valid p-formulas in the destination-based approach, then we will achieve a simplification of the rewriting process. We call this simplified version of our approach the *simplified approach*. This approach is less powerful than the general approach in the sense that it finds less rewritings. However, the simplified approach still finds maximum rewritings in some special cases. Before summarizing these cases in Theorem 5, we need to define *linear arithmetic constraints* and *basic comparisons*. A linear arithmetic constraint is a constraint of the form

$$a_1x_1 + a_2x_2 + \dots + a_lx_l \text{ op } b,$$

where a_1, \dots, a_l and b are constants, x_1, \dots, x_l are variables, op is one of $<, \leq, >, \geq, =, \neq$. A basic comparison is a constraint of the form $x \text{ op } y$, where x, y are variables or constants, op is one of $<, \leq, >, \geq, =, \neq$.

Theorem 5. 1. *Suppose the relation attributes are all from infinite domains. If the GCQs in the union query Q_u and views do not have constraints, then the union of all valid p-formulas is a maximum rewriting wrt to the language of union queries.*

2. *Suppose the relation attributes are all from the reals, and the GCQs in the union query Q_u do not have constraints. If the constraints of the views are conjunctions of basic comparisons (resp. linear arithmetic constraints involving only distinguished variables), then the union of all valid p-formulas of Q_u is a maximum rewriting with respect to the language of unions of conjunctive queries with conjunctions of basic comparisons (resp. linear arithmetic constraints).*

Note the assumption that the attributes are from infinite domains (resp. the reals) is necessary in 1 (resp. 2) of the theorem. For instance, the rewriting in Example 2 can not be found by the simplified approach.

6 Comparison With Related Work

In this section, we compare our approaches with some most related work, namely the Bucket, U-join, MiniCon and SVB algorithms. The Bucket algorithm is the most powerful (albeit the slowest) among these previous algorithms.

6.1 The Bucket Algorithm and the General Approach

The Bucket algorithm [6] is as follows: For each subgoal p_i of the query Q , a bucket B_i is created. If a view V has a subgoal p'_i which is unifiable with p_i , then let ϕ map every distinguished variable in p'_i to the corresponding argument in p_i . If $C \wedge \phi(C_V)$ is satisfiable (where C and C_V are the constraints of Q and V respectively), then put the view atom $\phi(v)$ in B_i . Then one view atom is taken from each of the buckets to form the body of a query Q' which has the head identical to that of Q . Then the algorithm tries to find a constraint C' such that $Q'^{exp} \wedge C' \sqsubseteq Q$. If C' can be found, then return $Q' \wedge C'$ as a rewriting.

As seen earlier, our general approach can find rewritings that can not be found by the Bucket algorithm, e.g., the rewritings in Examples 1 and 2. It is not difficult to see that any rewriting found by the Bucket algorithm can also be found by our general approach. In terms of efficiency, the Bucket algorithm does not need to combine p-formulas as in our general approach. However, for each query Q' resulting from a combination of the atoms in the buckets, it needs to do a containment test and find the constraint C' , which is expensive.

6.2 The U-join Algorithm and the Simplified Approach

The U-join algorithm [1] can be used to find contained rewritings of conjunctive queries using conjunctive views when neither the query nor the views have constraints. It proceeds as follows. First a set of inverse rules is obtained in the same way as in this paper. Then for each subgoal p_i in the user query, a “label”

L_i is created. Define $attr(L_i) = Arg(p_i)$. If $r :- v$ is an inverse rule, and r and p_i are unifiable, then the pair $(\sigma(Arg(p_i)), \sigma(v))$ is inserted into L_i provided that $\sigma(q)$ does not contain any Skolem functions. Here, σ is the most general unifier of p_i and r , and q is the head of the user query. The U-join of two labels L_1 and L_2 , denoted $L_1 \overset{u}{\bowtie} L_2$, is defined as follows. Let $Y = attr(L_1) \cap attr(L_2)$, and $Z = attr(L_2) - attr(L_1)$. Define $attr(L_1 \overset{u}{\bowtie} L_2) = attr(L_1) \cup Z$. If L_1 contains a pair (t_1, u_1) and L_2 contains a pair (t_2, u_2) , then $L_1 \overset{u}{\bowtie} L_2$ contains the pair $(\sigma(t_1, t_2[Z]), \sigma(u_1 \wedge u_2))$ where σ is a most general unifier of $t_1[Y]$ and $t_2[Y]$ such that $\sigma(u_1 \wedge u_2)$ does not contain Skolem functions, provided such σ exists. If $(\sigma, v_{i_1} \wedge \dots \wedge v_{i_n})$ is in the U-join of all labels corresponding to the subgoals of the query, and the head of the query is q , then $q\sigma :- v_{i_1}, \dots, v_{i_n}$ is a conjunctive rewriting of Q . The union of all such conjunctive rewritings is returned by the U-join algorithm.

It is not difficult to see that the U-join algorithm and our simplified approach generate the same rewritings. This is because the condition in generating the label L_i “ $\sigma(q)$ does not contain any Skolem functions” has the same effect as requiring that no distinguished variables of the query corresponds to a Skolem function in the valid destination, and the condition in U-joining L_1 and L_2 “ σ is a most general unifier of $t_1[Y]$ and $t_2[Y]$ such that $\sigma(u_1 \wedge u_2)$ does not contain Skolem functions” has the same effect as requiring that no argument in the query corresponds to two different Skolem functions or to both a free variable (constant) and a Skolem function in the valid destination. The efficiency of the two are similar because both need to do similar variable substitutions. One can use the simplified approach to the examples in [1] to get better understanding of our claim.

6.3 MiniCon, SVB and the Simplified Approach

We claim (for proof and more detailed comparisons, see [15]) that our simplified approach finds strictly more rewritings than the MiniCon and the SVB algorithms. For instance, the rewriting in Example 8 can not be found by MiniCon or SVB but it can be found by our simplified approach. Furthermore, if the query has many subgoals, our simplified approach tends to be faster. In addition, MiniCon and SVB do not handle constants properly. The authors do not say whether a constant should be treated like a distinguished variable or not. If not, the algorithm may fail to find a rewriting even for conjunctive queries without built-in predicates. For example, if $Q(x, y) :- p(x, y)$ is the query, and $V(x) :- p(x, 1)$ is the view, then no MCDs can be generated for Q and V , and no rewriting can be generated, but clearly $Q(x, 1) :- V(x)$ is a rewriting. On the other hand, if constants are treated like distinguished variables, the MiniCon algorithm may generate incorrect rewritings. For example, if the query is $q(u) :- p_1(x, u), p_2(u, x)$, and the views are $v_1(y) :- p_1(1, y)$ and $v_2(z) :- p_2(z, 2)$, then MiniCon will produce an incorrect “rewriting” $q(u) :- v_1(u), v_2(u)$.

7 Conclusion and Further Research

We presented a destination-based general approach for rewriting union queries using views. When used to rewrite GCQs, our approach is more powerful than existing algorithms. Furthermore, it can exploit NGDs to find semantic rewritings. A simplified version of the approach is less complete, but is faster and can still find maximum rewritings in some special cases. Our approaches generalize existing algorithms for rewriting GCQs using general conjunctive views.

Currently we are trying to identify more classes of built-in predicates with which there is a more efficient way of combining the p-formulas and with which we can obtain the maximum rewritings. We plan to investigate the effect of more complex integrity constraints on the rewriting. We also plan to implement our approaches so as to get empirical evaluation of their performance.

References

- [1] X. Qian. Query folding. In *Proc. of 12th ICDE*, pages 48–55, 1996.
- [2] J.D. Ullman. Information integration using logical views. *TCS: Theoretical Computer Science*, 239(2):189–210, 2000.
- [3] P.-A. Larson and H. Z. Yang. Computing queries from derived relations. In *Proc. of VLDB*, pages 259–269, 1985.
- [4] H. Z. Yang and P.-A. Larson. Query transformation for PSJ-queries. In *VLDB*, pages 245–254, 1987.
- [5] A. Levy. Answering queries using views: a survey. Technical report, Computer Science Dept, Washington Univ., 2000.
- [6] A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.
- [7] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. 16th PODS*, pages 109–116, 1997.
- [8] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, pages 484–495, 2000.
- [9] P. Mitra. An algorithm for answering queries efficiently using views. In *Proc. of the 12th Australasian database conference*, 2001.
- [10] J. Gryz. Query rewriting using views in the presence of functional and inclusion dependencies. *Information Systems*, 24(7):597–612, 1999.
- [11] O. Duschka, M. Genesereth, and A. Levy. Recursive query plans for data integration. *Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems*, pages 778–784, 2000.
- [12] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, 1998.
- [13] M. J. Maher. A logic programming view of CLP. In *Proc. 10th International Conference on Logic Programming*, pages 737–753, 1993.
- [14] M. Maher and J. Wang. Optimizing queries in extended relational databases. In *Proc. of the 11th DEXA conference, LNCS 1873*, pages 386–396, 2000.
- [15] J. Wang, M. Maher, and R. Topor. Rewriting general conjunctive queries using views. In *Proc. of the 13th Australasian Database Conference*, Australia, 28 January–1 February 2002.