Rewriting General Conjunctive Queries Using Views*

Junhu Wang[1,2] **Michael Maher**[1,3] **Rodney Topor**[1]

[1]CIT, Griffith University, Brisbane, Australia
{jwang,mjm,rwt}@cit.gu.edu.au
[2]GSCIT, Monash University, Churchill, Australia
John.Wang@infotech.monash.edu.au
[3]DMCS, Loyola University, Chicago, USA
mjm@cs.luc.edu

Abstract

The problem of rewriting queries using views has important applications in data integration, query optimization, and physical data independence maintenance. Previous researchers have proposed rewriting algorithms for queries and views that are Datalog programs or conjunctive queries with arithmetic comparisons such as $x < y$ and $y \geq 10$. We present a method for finding rewritings of general conjunctive queries, i.e, conjunctive queries with *arbitrary* built-in predicates, using views. Our method also has advantages over previous algorithms when there are no built-in predicates or when the built-in predicates are conjunctions of arithmetic comparisons. In particular, our method finds strictly more rewritings than the MiniCon [PL00] and the Shared-Variable-Bucket [Mit01] algorithms and tends to be more efficient when the built-in predicates of the query involve only distinguished variables. It finds all rewritings that can be found by the Bucket [LRO96] algorithm in most practical cases, and more efficiently. It finds maximum rewritings in several special cases.

Keywords: data integration, conjunctive query, built-in predicate, query containment, view, contained rewriting.

1 Introduction

The problem of rewriting queries using views (aka query folding [Qia96]) is of great importance in data integration, query optimization and physical data independence maintenance [Lev00]. For example, in a mediated data integration system, users are usually presented with a uniform interface through which queries are to be 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 [Ull00]. 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. In this paper, we will focus on the latter. More specifically, we will study the problem of finding contained rewritings using views when the user query and the views are all in the form of conjunctive queries with arbitrary built-in predicates, which we call *general conjunctive queries*. Arbitrary built-in predicates are necessary in many practical applications [MW00, Top91, HS93]. The rewritings we obtain are unions of general conjunctive queries. Our attention is focused on how to find all possible rewritings quickly, rather than on how efficient the rewritings can be evaluated.

There has been intensive research on the problem of rewriting queries using views, see [Lev00] for a survey. Among previous work on finding contained rewritings, the Bucket algorithm [LRO96] is used for conjunctive queries and conjunctive views where the views and the query may contain comparison predicates such as $x < a$, $y \neq x$. The Inverse-rule algorithm [DG97b, DGL00] was proposed for Datalog programs and Datalog views. More recently, the MiniCon algorithm [PL00] and the Shared-Variable-Bucket algorithm (hereafter referred to as the SVB algorithm) [Mit01] were developed as improved versions of the Bucket algorithm. A resolution-based algorithm was proposed in [Min00] as a generalization of earlier methods.

The problem with the Bucket algorithm is its inefficiency, as will be seen in Section 5.1. The MiniCon and the SVB algorithms are more efficient, but they miss too many rewritings when built-in predicates are present. Also, they do not handle constants properly (see Example 5.1). The Inverse-rule algorithm does not handle built-in predicates. The resolution-based algorithm [Min00] does not handle built-in predicates well either. As a result, it fails too often to find rewritings.

We present a destination-based method which (1) considers arbitrary built-in predicates in the conjunctive query and the views; (2) extends the optimization in [PL00] and finds strictly more contained rewritings than the MiniCon and the SVB algorithms; (3) finds most rewritings which can be found by the Bucket algorithm, and much more efficiently; (4) finds maximum rewritings in several special cases; (5) tends to be more efficient than the MiniCon and the SVB algorithms in the special case where the constraint of the query involves only distinguished variables, especially when the query has many subgoals. In passing, we point out an error in the MiniCon and the SVB algorithms.

The rest of the paper is organized as follows. Section 2 provides the technical background. Section 3 presents our method for rewriting queries using views.

*This work is partly supported by the Australian Research Council.

Section 4 lists properties of the rewritings found by our method. Section 5 compares our method with the Bucket, the MiniCon and the SVB algorithms. Section 6 concludes the paper with a summary and a discussion about further research.

2 Preliminaries

2.1 General conjunctive queries

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

$$q(X) :- \ p_1(X_1), ..., p_n(X_n), C \qquad (*)$$

where $q, p_1, \ldots, p_n$ are relation names, $X, X_1, \ldots, X_n$ are tuples of variables and constants, C is a conjunction of atomic constraints over the variables in $X \cup X_1 \cup \cdots \cup X_n$. We call $q(X)$ the *head*, $p_1(X_1), ..., p_n(X_n), C$ the *body*, and C the *constraint*[1] (or *built-in predicates*) of the query. Each atom $p_i(X_i)$ $(i = 1, \ldots, n)$ is called a *subgoal*. 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 variable or constant 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 the head either appears explicitly in a subgoal, or is (implicitly or explicitly) equated, by the constraint C, to a variable in at least one of the subgoals or to a constant.

A GCQ is said to be in *normal form*, if the arguments in every atom (head or body) 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 its constraint does not contain explicit or implicit non-tautological equalities between two variables or between a variable and a constant. Clearly, every GCQ can be put in normal form, and it can be put in compact form provided we can find all the implicit equalities in the constraint.

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.

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.

The following lemma relates GCQ containment to the existence of some particular containment mappings [Mah93].

Lemma 2.1 *Let Q_i $(i = 1, 2)$ be GCQs. Let C_i be the constraints in Q_i.*

(1) If there are containment mappings $\delta_1, \ldots, \delta_k$ from Q_2 to Q_1 such that $C_1 \rightarrow \vee_{i=1}^{k} \delta_i(C_2)$, then $Q_1 \sqsubseteq Q_2$.

(2) If Q_2 is in normal form, C_1 is satisfiable, and $Q_1 \sqsubseteq Q_2$, then there must be containment mappings $\delta_1, \ldots, \delta_k$ from Q_2 to Q_1 such that $C_1 \rightarrow \vee_{i=1}^{k} \delta_i(C_2)$.

The union of a finite number of GCQs is called a *union query*.

2.2 Rewritings and maximum rewritings

We assume the existence of a set of base relations and a set $\mathcal{W}$ 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 [AD98, ALU01]: 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 [MLF00, DG97a, LRO96]. 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 $\mathcal{W}(\mathcal{D})$ to denote a *view instance* $\mathcal{W}(\mathcal{D})$ (with respect to $\mathcal{D}$) consisting of instances of the view relations. Since the open world assumption is used, each relation instance in $\mathcal{W}(\mathcal{D})$ may contain only part of the answers computed to the corresponding view using $\mathcal{D}$.

Given a GCQ Q 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 the answers to Q_r computed using any view instance $\mathcal{W}(\mathcal{D})$ are correct answers to Q. We call such a query Q_r a *contained rewriting* or simply a *rewriting*. When Q_r is a general conjunctive (or union) query, we call it a *general conjunctive (or union) rewriting*. 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, we need the expansion of Q_r, as defined below.

Definition 2.1 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 $v(x_1, \ldots, x_k)$ of Q_r, suppose $v(y_1, \ldots, y_k)$ 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 $i = 1, \ldots, k$, and maps every non-head variable in V to a distinct new variable, then

(1) replace $v(x_1, \ldots, x_k)$ with the body of $\sigma(V)$,

(2) now if a variable x_i $(1 \leq i \leq k)$ appears only in the constraint (not in the head or any subgoals), then replace x_i with the variable in the body 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 $q(x) :- \ p(z), x = z$.

Clearly, if the views are safe GCQs and Q_r is a safe GCQ defined on the view relations, then Q_r^{exp} is a safe GCQ defined on the base relations. Furthermore, Q_r is a rewriting of Q if and only if $Q_r^{exp} \sqsubseteq Q$, and a

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

rewriting Q_r is a non-empty rewriting if and only if Q_r^{exp} is a non-empty query.

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

Definition 2.2 A rewriting Q_1 of Q is said to be *maximum* 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 $\mathcal{W}(\mathcal{D})$ 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 to show that $Q_2^{exp} \sqsubseteq Q_1^{exp}$ holds for any other rewriting Q_2.

Example 2.1 Let the views V_1 and V_2 be

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

$v_2(x, y) :- \ p(x, y)$ respectively.

Let the query Q be

$q(x) :- \ p(x, y)$.

Clearly $Q_1 : \ q(x) :- \ v_1(x, y)$ and

$Q_2 : \ q(x) :- \ v_2(x, y)$

are two conjunctive rewritings, and $Q_1^{exp} = Q_2^{exp}$.

But neither Q_1 nor Q_2 is a maximum rewriting with respect to conjunctive queries or union queries because although V_1 and V_2 are defined to be equivalent, v_1 and v_2 may contain different tuples in a view instance. However, $Q_1 \cup Q_2$ is a maximum rewriting with respect to union queries. □

Note also that the condition for a maximum rewriting is stronger than that for a *maximally contained rewriting* in [PL00, Mit01], and that for a *maximally contained retrievable program* in [DG97b], and that for a *maximally contained query plan* in [DGL00].

2.3 Inverse rules and inferred constraints

Given a view V:

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

we can compute a set of *inverse rules* [DG97b]: 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 variable to the corresponding Skolem functions, then the inverse rules are

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

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 2.2 For the view

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

there are two inverse rules:

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

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

where $f(x, z)$ is a Skolem function. In the first inverse rule, x is a free variable but z is not. In the second inverse rule, z is a free variable but x is not. □

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.

Intuitively, an inverse rule has the potential to "link" a subgoal of the query to a view when the head of the rule and the subgoal share the same relation name.

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 2.3 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 A destination-based rewriting method

Our method for query rewriting 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 a rewriting; in the second step, we combine these p-formulas to see whether we can obtain correct rewritings.

3.1 Generating p-formulas

We assume the compact sets of inverse rules have been computed in advance.

Let Q be as given in (*). Roughly speaking, a p-formula of Q is a formula that has the "shape" of a rewriting: it has a head which is the same as that of Q, and it has a body consisting of only view atoms and a constraint, but the constraint might involve Skolem functions (so it may not be a correct GCQ). Furthermore, if we replace the view atoms by the Skolemized view body, we will get a GCQ which is contained in Q.

In order to find a p-formula of Q, we need to find a sequence of inverse rule heads which are "compatible" with the subgoals of Q. We call such a sequence a *destination*[2] of Q.

We assume Q is in compact form so that the distinguished variables appear explicitly in the head.

Definition 3.1 Given the GCQ Q as in (*) and a set of compact inverse rules IR, a *destination* of Q wrt to IR is a sequence DS of n atoms $DS = p_1(Y_1), \ldots, p_n(Y_n)$ 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 ith subgoal of Q.

[2] We can imagine that the subgoals of Q are moving towards the atoms in the sequence. Hence the name.

2. Each constant in $p_i(X_i)$ corresponds[3] to the *same* constant or a free variable in $p_i(Y_i)$.

3. Each distinguished variable in $p_i(X_i)$ corresponds to a constant or a free variable in $p_i(Y_i)$.

4. Different occurrences of the same non-distinguished variable in Q correspond either all to free variables and constants, or all to the same Skolem function, in DS.

5. No two occurrences of the same variable in Q correspond to two different constants in DS, and no two occurrences of the same variable *in the same rule head* correspond to two different constants in Q.

□

Intuitively a destination "links" the subgoals of Q to the view atoms in a rewriting. Once we have found a destination DS of Q, we can use it to construct a p-formula as follows:

1. For each occurrence of a constant α in DS, if the corresponding argument in Q is a distinguished variable x, then add an equality [4] $x = \alpha$ to the constraint C of Q; if the corresponding argument in Q is a non-distinguished variable, then replace all occurrences of the variable with α. If C becomes unsatisfiable, then stop with failure.

2. Define a relation $\sim$ among the atoms in DS such that two atoms p and q satisfy $p \sim q$ if they share a Skolem function $f(Z)$ and the two occurrences of $f(Z)$ in p and q correspond to the same non-distinguished variable of Q, or if there are some other atoms $r_1, \ldots, r_t$ in DS such that $p \sim r_1, r_1 \sim r_2, \ldots, r_{t-1} \sim r_t, r_t \sim q$. The relation $\sim$ is an equivalence relation. Divide the atoms in DS into disjoint groups according to $\sim$.

   Clearly, the atoms in the same group are the heads of rules generated from the same view, that is, the rules have the identical body.

3. For each group G, do the following:

   Let G' be the set of subgoals of Q corresponding to the atoms in G. Suppose the body of the rule corresponding to each atom in this group[5] is $v(Z)$.

   (a) Define a mapping ϕ and a constraint E as follows: Initially, $E = True$.

   For each free variable y in G, suppose X' is the set of arguments in G' that correspond to y. If there is a constant or distinguished variable x in X', then let ϕ map y to x and, for any other distinguished variable or constant x' in X', let $E = E \wedge (x' = x)$, for any non-distinguished variable x'' in X', replace all occurrences of x'' in Q with x. If all of the arguments in X' are non-distinguished variables, then choose one of them, for example x, let ϕ map y to x, and for any other variable $x' \in X'$, replace all occurrences of x' in Q with x.

   If E is not satisfiable (e.g, when E contains an equality between two different constants) or it is not consistent with the constraint C of Q, then stop with failure.

   For each variable $z \in Z$ that does not appear as a free variable in G, let ϕ map z to a distinct new variable not in Q.

   (b) In Q, replace G' with $\phi(v(Z))$ and add (conjoin) E to the constraint C.

   (c) For each non-distinguished variable x of Q that appears in C, if x corresponds to a Skolem function $f(Z)$ in G, then, in C, replace x with $\phi(f(Z))$.

4. Remove duplicate atoms if necessary. Output the resulting formula (i.e, the formula modified from Q). It is a p-formula.

Example 3.1 Let the query Q be

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

Let the views be

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

The compact inverse rules are:

R1: $p'(u) \leftarrow v_1(u)$

R2: $p(f(y,z),y) \leftarrow v_2(y,z)$

R3: $p(y,z) \leftarrow v_2(y,z)$

R4: $r(g(y,z),y) \leftarrow v_3(y,z)$

R5: $r(y,z) \leftarrow v_3(y,z)$

There are only two destinations:

(1) $p'(u), p(f(y,z),y), r(y,z)$

(2) $p'(u), p(y,z), r(y,z)$

For destination (1), since the three atoms do not share any Skolem functions, we put them into separate groups. The three groups correspond to rules R1, R2 and R5 respectively. So we rename u to u in R1, y to y in R2, and y to y, z to v in R5. Finally we replace the subgoals of the query with the bodies of R1, R2 and R5, and replace x in Q with the Skolem function $f(y,z)$ to get the p-formula

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

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

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

□

Note the p-formula may have Skolem functions in the constraint part.

If F is a p-formula, we will use F^{exp} to denote the formula obtained by expanding F using the Skolemized views the same way we expand a GCQ, and call it the *expansion* of F. Treating the Skolem functions in F^{exp} as variables, it is easy to show the following theorem.

Theorem 3.1 *The expansion of a p-formula of Q is a GCQ contained in Q.*

Thus if there happen to be no Skolem functions in the p-formula, then the p-formula is a rewriting. In particular, if the query Q do not have constraint, or its

[3] An argument in $p_i(Y_i)$ and an argument in $p_i(X_i)$ are said to correspond to each other iff they are in the corresponding positions of the two atoms.

[4] The reason we do not replace x with α is to make the head of the rewriting identical to that of Q.

[5] If there are rules that have the same head but different bodies, then choose one of them in turn to generate different p-formulas.

constraint involves only distinguished variables, then every p-formula is a rewriting.

Corollary 3.1 *If Q has no constraint, or the constraint of Q involves only distinguished variables, then any p-formula of Q is a rewriting of Q.*

However, when a p-formula contains Skolem functions, it is not a correct GCQ because these Skolem functions do not appear in the view atoms or in the head, and their values can not be determined[6]. So we need to find some way to combine the p-formulas and remove the Skolem functions.

3.2 Removing Skolem functions

We note that the Skolem functions in a p-formula only appear in the constraint part, and every p-formula has exactly the same head.

Given some p-formulas that involve Skolem functions, there are several possible cases where we can combine these p-formulas and remove the Skolem functions, making what is left a rewriting of Q. In general, we need to utilize the power of a constraint solver. We summarize these cases into the following steps.

1. For each p-formula
$$q(X) :- \ v_1(Z_1), \ldots, v_k(Z_k), C$$
that contains Skolem functions,
   (a) Split the constraint C into C' and C'', where C' is the conjunction of atomic constraints involving at least one Skolem function, and C'' is the conjunction of atomic constraints involving no Skolem functions.
   (b) Attach the inferred constraints of the view atoms to the p-formula. To distinguish these inferred constraints from the constraint in the p-formula, we put them in a bracket. If the inferred constraint of $v_i(Z_i)$ is D_i for $i = 1, \ldots, k$, then we will get
$$q(X) :- \ v_1(Z_1), \ldots, v_k(Z_k), C' \wedge C'' \ [D]$$
where $D = D_1 \wedge \cdots \wedge D_k$.
2. If there is a constraint D' which involves no Skolem functions such that $D \wedge C'' \wedge D'$ is satisfiable and $D \wedge C'' \wedge D' \rightarrow C'$, then output
$$q(X) :- \ v_1(Z_1), \ldots, v_k(Z_k), C'' \wedge D'$$
Note that D' may be *TRUE*.
3. Combine p-formulas which are identical (modulo renaming of non-head variables and re-ordering of the view atoms) except their constraint parts as follows:
   (a) Make the "non-constraint" parts of the p-formulas identical by appropriate variable renamings. Apply the same renamings to the constraint parts.
   (b) Check the disjunction of the constraints in those p-formulas: Suppose it is $(C'_1 \wedge C''_1) \vee \cdots \vee (C'_t \wedge C''_t)$, where C'_i $(i = 1, \ldots, t)$ involves Skolem functions, but C''_i does not. Rewrite it to the equivalent constraint $(C''_1 \vee \cdots \vee C''_t) \wedge C'''$.
   (c) Suppose the common non-constraint part of the p-formulas is
$$q(X) :- \ v_1(Z_1), \ldots, v_k(Z_k).$$
Denote the conjunction of inferred constraints of $v_1(Z_1), \ldots, v_k(Z_k)$ by D. If there is a constraint D' involving no Skolem functions such that $D \wedge D' \wedge (C''_1 \vee \cdots \vee C''_t)$ is satisfiable and $D \wedge D' \wedge (C''_1 \vee \cdots \vee C''_t) \rightarrow C'''$, then output the query
$$\begin{aligned} q(X) \ :- \ & v_1(Z_1), \ldots, v_k(Z_k), \\ & D' \wedge (C''_1 \vee \cdots \vee C''_t). \end{aligned}$$

It is easy to see that, if F is the set of input p-formulas and O is an output query of the above process, then O^{exp} is contained in the union of the expansions of the p-formulas in F. Thus we have

Theorem 3.2 *The above process of removing Skolem functions generates correct contained rewritings.*

Example 3.2 Continuing with Example 3.1, the two p-formulas are
$$q(u) :- \ v_1(u), v_2(y, z), v_3(y, v), f(y, z) < y, y < v$$
and
$$q(u) :- \ v_1(u), v_2(x, y), v_3(y, v), x < y, y < v.$$
The second p-formula is a rewriting because it has no Skolem functions.

Attaching the inferred constraints to the first p-formula, we get
$$\begin{aligned} q(u) \ :- \ & v_1(u), v_2(y, z), v_3(y, v), f(y, z) < y, \\ & y < v \ [f(y, z) < z, g(y, v) < v] \end{aligned}$$
Since $(z \leq y) \wedge (f(y, z) < z) \rightarrow f(y, z) < y$, we can replace $f(y, z) < y$ with $z \leq y$ (Step 2 in the above process) and get the rewriting
$$q(u) :- \ v_1(u), v_2(y, z), v_3(y, v), z \leq y, y < v.$$
□

The next example demonstrates the case where two p-formulas can be combined.

Example 3.3 Let the query Q be
$$q(x) :- \ p(x, y, z, w), z < w.$$
Let the view be
$$\begin{aligned} v(x) :- \ & p(x, y_1, z_1, w), p(x, y_2, w, w_2), \\ & y_1 < y_2, z_1 < w_2. \end{aligned}$$
The compact inverse rules are
R1: $p(x, f_1(x), f_2(x), f_3(x)) \leftarrow v(x)$
R2: $p(x, g_1(x), f_3(x), g_2(x)) \leftarrow v(x)$
There are two destinations of Q:

[6] It is also incorrect to regard the Skolem functions as usual existential variables in the constraint (if we allow for existential variables in the constraint), since they represent special existential variables that appear in some view subgoals. For example, if $v(x) :- p(x, y)$ is the definition of view v, $q(x) :- p(x, y), x > y$ is a query, and $f(x)$ is a Skolem function in the rule $p(x, f(x)) \leftarrow v(x)$, then $q(x) :- v(x), x > f(x)$ is not equivalent to $q(x) :- v(x), \exists y \ x > y$. This is the main reason we use Skolem functions.

$p(x, f_1(x), f_2(x), f_3(x))$ and

$p(x, g_1(x), f_3(x), g_2(x))$.

Thus we get two p-formulas:

$q(x) :- \ v(x), f_2(x) < f_3(x)$ and

$q(x) :- \ v(x), f_3(x) < g_2(x)$.

Combining them, we will get the formula

$$q(x) \ :- \ v(x), f_2(x) < f_3(x) \vee f_3(x) < g_2(x) \\ [f_1(x) < g_1(x) \wedge f_2(x) < g_2(x)].$$

Since

$$f_2(x) < g_2(x) \rightarrow f_2(x) < f_3(x) \vee f_3(x) < g_2(x),$$

we can get a rewriting $q(x) :- \ v(x)$. □

3.3 Finding all destinations

There are different ways for finding all destinations. Here we present two of them.

Let us divide the rules into groups such that two rules are in the same group if and only if their heads have the same name. Let RS_i be the group of rules whose heads have the same name as that of $p_i(X_i)$ for $i = 1, \ldots, n$.

Our first method is to build a destination from an initially empty set DS by adding to it the head of a rule from each of $RS_1, \ldots, RS_n$ one by one. The head of a rule from RS_i *can be added* to DS iff adding it to DS leaves DS consistent. Here by *consistent* we mean that if we denote the subgoals of Q corresponding to the atoms in DS by DS', then (1) every constant in DS' corresponds to the *same* constant or to a free variable in DS; (2) every distinguished variable in DS' corresponds to a free variable or to a constant in DS; (3) all occurrences of the same non-distinguished variable in DS' correspond either all to free variables and constants, or all to the same Skolem function; (4) no two occurrences of the same variable in DS' correspond to two different constants in DS, and no two occurrences of the same variable in the same rule correspond to two different constants in DS'.

Our second method is to use the procedure *FindAllD* as listed in Figure 1. The procedure calls a function *FindPD* to find some partial destinations, where a *partial destination* is a set P of rule heads such that (1) each rule is from a distinct group among $RS_1, \ldots, RS_n$ (let us call the atom from RS_i the *mirror* of $p_i(X_i)$); (2) P is consistent; (3) every subgoal of Q involving a non-distinguished variable which corresponds to a Skolem function in P has a mirror in P. The empty set is a partial destination. A partial destination that contains a mirror for every subgoal of Q is a destination.

When called on a subgoal p_i of Q, the function *FindPD* finds all *minimum partial destinations* (a minimum partial destination is one such that removing any atom from it will make it no longer a partial destination) containing a mirror for p_i, it can be implemented in different ways. For instance, one implementation is to use enumeration, as listed in Figure 2.

Procedure *FindAllD*
Input: The query $Q \equiv q(X) :- \ p_1, \ldots, p_n, C$
The groups $RS_1, \ldots, RS_n$ of rules
Output: The set $allD$ of all destinations of Q

Initially, $allD = \{\emptyset\}$, i.e, $allD$ contains only the empty partial destination.
while there is a partial destination DS in $allD$ which is not a destination
Choose a subgoal p_i of Q which does not have a mirror in DS;
if PD_i has not been computed
Let PD_i=*FindPD*(p_i);
if $PD_i = \emptyset$
Discard DS from $allD$;
else
Let$PD_{i,1}, \ldots, PD_{i,k_i}$ be all those minimum partial destinations in PD_i that do not contain a mirror for any subgoal of Q which already has a mirror in DS.
Add $DS \cup PD_{i,1}, \ldots, DS \cup PD_{i,k_i}$ to $allD$;
Remove DS from $allD$;
Return $allD$;

Figure 1: Procedure **FindAllD**

Function *FindPD()*
Input: The query $Q \equiv q(X) :- \ p_1, \ldots, p_n, C$,
The groups $RS_1, \ldots, RS_n$ of rules,
A subgoal p_i of Q.
Output: The set $allPD$ of all partial destinations containing a mirror of p_i.

Initially, $allPD = \emptyset$.
Choose every rule in RS_i whose head p' can be added to the empty set;
Put the set $\{p'\}$ to $allPD$;
for every set PD in $allPD$
if there is a non-distinguished variable x in Q which corresponds to a Skolem function $f(Z)$ in PD, and there is a subgoal p_j of Q in which x appears and p_j has no mirror in PD
Choose all rules in RS_j whose head can be added to PD;
Suppose the heads of such rules are $h_1, \ldots, h_s$;
Add the sets $PD \cup \{h_1\}, \ldots, PD \cup \{h_s\}$ to $allPD$;
Remove PD from $allPD$.

Figure 2: Function **FindPD**

4 Properties of the destination-based method

In this section, we list some special cases where the destination-based method finds maximum rewritings. We assume the attributes of all relations are from infinite domains.

First, when the query and views do not have constraints, the union of all p-formulas is a maximum rewriting with respect to the language of union queries. This is implied by the next theorem.

Theorem 4.1 *Suppose the relation attributes are all from infinite domains. If the query Q and the views do not have constraints, then for any general conjunctive rewriting Q_r, there are some p-formulas $Q_1, \ldots, Q_s$ defined on the view relations mentioned in Q_r such that $Q_r \sqsubseteq Q_1 \cup \cdots \cup Q_s$.*

In Theorem 4.1 the condition that the variables are from infinite domains is necessary, as shown by the next example.

Example 4.1 Let $p(x, y, z)$ be a relation, where x is from the reals and y, z are from the domain $\{\texttt{ON}, \texttt{OFF}\}$. Consider the query

Q: $q(x) :- \ p(x, y, y)$

and the view

V: $v(x) :- \ p(x, y, \texttt{ON}), p(x, y, \texttt{OFF})$.

It should be clear that $V \sqsubseteq Q$. Thus $q(x) :- \ v(x)$ is a rewriting of Q. However, there are no destinations and hence we can not find any p-formulas. □

Second, we consider the case where the query has no constraints, and the constraints of the views are conjunctions of linear arithmetic constraints involving only distinguished variables (resp. basic comparisons), assuming all attributes are from the reals. A *linear arithmetic constraint* is a constraint of the form

$$a_1x_1 + a_2x_2 + \cdots + a_lx_l \ op \ b,$$

where $a_1, \ldots, a_l$ and b are constants, $x_1, \ldots, x_l$ are variables, *op* is one of $<, \leq, >, \geq, =, \neq$. A *basic comparison* is a constraint of the form $x \ op \ y$, where x, y are variables or constants, *op* is one of $<, \leq, >, \geq, =, \neq$. We claim that in this case, the union of all p-formulas is a maximum rewriting with respect to the language of unions of conjunctive queries with linear arithmetic constraints (resp. basic comparisons). This is due to the next theorem.

Theorem 4.2 *Suppose the relation attributes are all from the reals. If the query does not have constraint, then*

(1) If the constraints of the views are conjunctions of linear arithmetic constraints involving only distinguished variables, then for every general conjunctive rewriting Q_r of Q whose constraint is a conjunction of linear arithmetic constraints, there is a p-formula Q_p of Q defined on the view relations mentioned in Q_r such that $Q_r^{exp} \sqsubseteq Q_p^{exp}$.

(2) If the constraints in the views are conjunctions of basic comparisons, then for every general conjunctive rewriting Q_r of Q whose constraint is a conjunction of basic comparisons, there is a p-formula Q_p of Q defined on the view relations mentioned in Q_r such that $Q_r^{exp} \sqsubseteq Q_p^{exp}$.

Note the assumption that the attributes are from the reals are necessary[7].

Example 4.2 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

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

$v_2(x) :- \ p_2(x, 1), p_2(x, 2)$.

The inverse rules are

$p_1(x, f(x)) \leftarrow v_1(x)$,

$p_2(x, 1) \leftarrow v_2(x)$,

$p_2(x, 2) \leftarrow v_2(x)$.

Thus there is no destination of Q, and hence no rewriting that can be obtained by the destination-based method. But the query $q(x) :- \ v_1(x), v_2(x)$ is a rewriting of Q because its expansion

$q(x) :- \ p_1(x, y), p_2(x, 1), p_2(x, 2), y > 0, y < 3$

is contained in Q (this can be verified by Lemma 2.1). □

The above example also shows that the claim in [PL00] that "if the views contain comparison predicates but the query does not, then the MiniCon algorithm finds maximally contained rewritings" is not strictly correct, because we will see in Section 5 that any rewritings that can be found by the MiniCon Algorithm can also be found by our destination-based method.

In *(1)* of Theorem 4.2, if we do not assume the constraints in the views involve only distinguished variables, then the result will not hold.

Example 4.3 let $p_1(x, y, z)$ and $p_2(x, z)$ be two relations, where x, y and z are all from the reals.

Let the query Q be:

$q(z) :- \ p_1(0, 0, z), p_2(0, z)$.

Let the views V_1, V_2 be

$v_1(x, z) :- \ p_1(x, y, z), x + y = 0$ and

$v_2(x, z) :- \ p_2(x, z), x = 0$, respectively.

According to definition, y is a non-distinguished variable in V_1. The compact inverse rules are

$p_1(x, f(x), z) \leftarrow v_1(x, z)$

$p_2(0, z) \leftarrow v_2(0, z)$

Thus we can not find a destination of Q. So there are no p-formulas or rewritings found.

But $q(z) :- \ v_1(0, z), v_2(0, z)$ is a rewriting, because the expansion of the above query is

$q(z) :- \ p_1(0, y, z), p_2(0, z), 0 + y = 0, x = 0$

which is equivalent to Q. □

The problem shown in the above example is caused by the definitions of safety and distinguished variables. In an extended version of the destination-based method, we revise those definitions so that our method does find the rewriting in the above example.

[7] In fact, we only need all variables that appear in the constraints are from the reals.

5 Comparison with related work

In this section, we compare our destination-based method with three most closely related algorithms: The Bucket algorithm, the MiniCon algorithm, and the SVB algorithm. These algorithms assume that different views and the query use disjoint sets of variables.

5.1 The Bucket algorithm

The Bucket algorithm [LRO96] 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.

The problem with the Bucket algorithm is its inefficiency (compared to our method) for the following reasons. First, it considers every possible combinations of the atoms from the buckets rather than only those that are potentially useful. For instance, if we apply the Bucket algorithm to the query and the views in Example 3.1, we will get three buckets $\{v_1(u)\}$, $\{v_2(y, z_2), v_2(x, y)\}$ and $\{v_3(v, z_3), v_3(y, v)\}$ (where we add a subscript i to the variables in the ith view for $i = 1, 2, 3$ so that the variables in different views and the query are different). Thus it has to consider four queries

$Q_1:\ q(u) :-\ v_1(u), v_2(y, z_2), v_3(y, v)$

$Q_2:\ q(u) :-\ v_1(u), v_2(x, y), v_3(y, v)$

$Q_3:\ q(u) :-\ v_1(u), v_2(y, z_2), v_3(v, z_3)$

$Q_4:\ q(u) :-\ v_1(u), v_2(x, y), v_3(v, z_3)$

rather than only the first two. Second, for each query Q' resulting from a combination, it needs to do a containment test and find the constraint C'. This entails the computation of all possible containment mappings from Q to Q'^{exp}. For instance, for the query Q_1 above, it has to compute all the containment mappings from Q to Q_1^{exp}, and then use a constraint solver to find C'. Recall our method does not need to compute containment mappings when we have the p-formulas.

5.2 The MiniCon and the SVB algorithms

The MiniCon algorithm [PL00] proceeds as follows:

Step 1. For every subgoal p of Q and every subgoal p' of view V (suppose $v(Z)$ is the head of V), find a least restrictive mapping h from Z to Z such that there exists a mapping ϕ s.t. $\phi(p) = h(p')$; If h and ϕ exist, then extend the domain of ϕ to the variables in a minimum set G of subgoals of Q such that

(1) every subgoal in G is mapped to a subgoal of $h(V)$ by ϕ;

(2) every distinguished variable in G is mapped to a distinguished variable in $h(V)$;

(3) if a non-distinguished variable x in G is mapped to a non-distinguished variable of $h(V)$, then (i) every subgoal of Q involving x is in G; (ii) all variables in the comparison predicates of Q that involve x are in the domain of ϕ, and $h(d)$ and $\phi(C)$ are consistent, where d is the constraint of V;

(4) $h(d) \rightarrow \phi(C')$, where C' is the conjunction of comparisons in Q involving only variables in the domain of ϕ, and C' involves at least one non-distinguished variable of Q;

The tuple $(h, v(h(Z)), \phi_c, G)$ (where ϕ_c is the extended mapping) is called a *minimum MCD* in [PL00]. Among those minimum MCDs formed from the same h and ϕ, the algorithm only retains those that have the fewest number of subgoals in the G component.

Step 2. If there are minimum MCDs

$$(h_1, v_1(Y_1), \phi_1, G_1), \ldots, (h_k, v_k(Y_k), \phi_k, G_k)$$

such that $G_1, \ldots, G_k$ are pairwise disjoint and $G_1, \ldots, G_k$ together cover all the subgoals of Q, then

(1) If ϕ_i maps two or more variables $x_1, \ldots, x_s$ in G_i to the same argument, then choose one of the variables as a representative, denote the representative variable of x_j by $EC_i(x_j)$. Thus $EC_i(x_1) = \cdots = EC_i(x_s)$. For every variable x in Q, if $EC_i(x) \neq EC_j(x)$ $(1 \leq i, j \leq k)$, then let $EC(x)$ be one of them but consistently across all y for which $EC_i(y) = EC_i(x)$;

(2) For each $y \in Y_i$, if exists x such that $\phi_i(x) = y$, then let $\psi_i(y) = x$, otherwise let $\psi(y)$ be a distinct new variable;

(3) Create the rewriting

$$\begin{aligned} q(EC(X)) \ :-\ & v_1(EC(\psi_1(Y_1))), \ldots, \\ & v_k(EC(\psi_k(Y_k))), EC(C'') \end{aligned}$$

where C'' is the set of comparisons in Q which are not implied by the inferred constraints of the view atoms.

The SVB algorithm [Mit01] greatly resembles the MiniCon algorithm. It constructs two types of buckets: the *single-subgoal buckets* and the *shared-variable buckets*. The single-subgoal buckets correspond to the MCDs which have a single subgoal in their G components, and the shared-variable buckets correspond to the minimum MCDs having two or more subgoals in their G components. The algorithm then constructs a rewriting by combining atoms from some buckets which represent disjoint sets of subgoals and which together respresent all subgoals of Q. Comparison predicates are handled in a way similar to the way they are by MiniCon.

Therefore, in the following discussion, we will only compare our algorithm with the MiniCon algorithm.

5.2.1 Similarities and differences

There are some similarities between our method and the MiniCon algorithm. The minimum MCDs in MiniCon correspond to the minimum partial destinations, and the union of pairwise disjoint MCDs which cover all subgoals corresponds to a destination in our method.

However, our method is superior to MiniCon in the following respects.

First, the MiniCon algorithm does 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. This is demonstrated in the next example.

Example 5.1 Let the query Q be

$q(u) :- p_1(x, u), p_2(u, x).$

Let the views be $v_1(y) :- p_1(1, y)$ and

$v_2(z) :- p_2(z, 2).$

The minimum MCDs are shown below

h	$v(Y)$	ϕ	G
I	$v_1(y)$	$u \to y,\ x \to 1$	$p_1(x, u)$
I	$v_2(z)$	$u \to z,\ x \to 2$	$p_2(u, x)$

where I is the identity mapping. Using MiniCon, we would generate a query $q(x) :- v_1(u), v_2(u)$. But clearly the expansion of the query is not contained in Q. □

Second, the handling of constraints by MiniCon is too restrictive, thus it may miss more contained rewritings than our method. For instance, the rewriting in Example 3.3 can not be found by MiniCon. On the other hand, any *correct* rewritings that can be found by the MiniCon algorithm can also be found by our method, because of the above-mentioned similarities between the two. One can verify the same rewritings can be found (with no loss of efficiency) if we apply the destination-based method to the examples in [PL00] and [Mit01]. Here we use the running example in [PL00] as a demonstration.

Example 5.2 In the running example of [PL00], the query Q is

$q_1(x) :- cites(x, y), cites(y, x), sameTopic(x, y).$

The views are

$v4(u) :- cite(u, v), cites(v, u)$

$v5(u, v) :- sameTopic(u, v)$

$v6(u, v) :- cites(u, w), cites(w, v), sameTopic(u, w)$

The reverse rules are:

R1: $cites(u, f_1(u)) \leftarrow v4(u)$

R2: $cites(f_1(u), u) \leftarrow v4(u)$

R3: $sameTopic(u, v) \leftarrow v5(u, v)$

R4: $cites(u, f_2(u, v)) \leftarrow v6(u, v)$

R5: $cites(f_2(u, v), v) \leftarrow v6(u, v)$

R6: $sameTopic(u, f_2(u, v)) \leftarrow v6(u, v)$

The only destination is

$cites(u, f_2), cites(f_2, v), sameTopic(u, f_2),$

where $f_2 \equiv f_2(u, v)$. All of the atoms belong to the same group, which corresponds to the view $v6(u, v)$. Therefore, we can first define the mapping $\phi : u \to x, v \to x$, and then replace the body of Q with $\phi(v6(u, v))$ to get the rewriting $q_1(x) :- v6(x, x)$. □

Third, in the special case that the constraint of the query involves only distinguished variables, the MiniCon algorithm is not the most efficient since it may generate many MCDs which are useless, while our method allows different choices of methods for finding destinations, which is the major contributor for complexity. In particular, our second method for finding destinations is more efficient than MiniCon, especially when the query has many subgoals, because MiniCon needs to find all minimum MCDs. In effect, it is like calling the function *FindPD* for every subgoal of Q; while our method calls that function only for some subgoals. The claimed efficiency of MiniCon comes partly from avoiding computing the minimum MCDs multiple times, but our algorithm also calls the function *FindPD at most* once for each subgoal of Q.

The next example shows this point.

Example 5.3 Let Q be

$q(x) :- p_1(x, y), p_2(y, z), p_3(z, u), p_4(x), p_5(y), p_6(z).$

Let the views V_1, V_2 be

$v_1(x_1, z_1) :- p_1(x_1, y_1), p_2(y_1, z_1), p_4(x_1)$, and

$v_2(y_2, z_2) :- p_2(y_2, z_2), p_5(y_2), p_6(z_2)$,

$v_3(y_3) :- p_2(y_3, z_3), p_3(z_3, u_3)$, respectively.

The inverse rules are

R1: $p_1(x, f_1(x_1, z_1)) \leftarrow v_1(x_1, z_1)$

R2: $p_2(f_1(x_1, z_1), z_1) \leftarrow v_1(x_1, z_1)$

R3: $p_4(x_1) \leftarrow v_1(x_1, z_1)$

R4: $p_2(y_2, z_2) \leftarrow v_2(y_2, z_2)$

R5: $p_5(y_2) \leftarrow v_2(y_2, z_2)$

R6: $p_6(z_2) \leftarrow v_2(y_2, z_2)$

R7: $p_2(y_3, f_2(y_3)) \leftarrow v_3(y_3)$

R8: $p_3(f_2(y_3), f_3(y_3)) \leftarrow v_3(y_3)$

The MiniCon algorithm considers every subgoal of Q against every subgoal of every view to find all minimum MCDs, and then finds there are no combinations that cover all subgoals of Q. In effect, it is like calling the function *FindPD* six times.

However, starting from the subgoal $p_1(x, y)$ and using our second alternative for finding destinations, our algorithm needs to call the function only twice (once on the subgoal p_1, and once on the subgoal p_3) before we find there are no destinations. □

6 Conclusion and future work

We presented a destination-based method for rewriting GCQs using views which is more efficient than the Bucket algorithm, finds strictly more rewritings than the MiniCon and the SVB algorithms, and tends to be more efficient than MiniCon and SVB in the special case where the constraints of the query involve only distinguished variables.

We plan to implement the destination-based method so as to evaluate its performance empirically. We plan to identify more classes of constraints for which there is a systematic and efficient way of combining the p-formulas so that a maximum rewriting can be found.

Currently we are investigating how to best extend our method to rewriting union queries and to cases where integrity constraints exist.

References

[AD98] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, 1998.

[ALU01] F. N. Afrati, C. Li, and J. D. Ullman. Generating efficient plans for queries using views. In *Proc. of SIGMOD*, pages 319–330, 2001.

[DG97a] O. M. Duschka and M. Genesereth. Query planning in infomaster. In *Proc. of ACM Symposium on Applied Computing*, 1997.

[DG97b] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. 16th PODS*, pages 109–116, 1997.

[DGL00] 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.

[HS93] J. M. Hellerstein and M. Stonebraker. Predicate migration: optimizing queries with expensive predicates. *SIGMOD Record*, 22(2):267–276, 1993.

[Lev00] A. Levy. Answering queries using views: a survey. Technical report, Computer Science Dept, Washington Univ., 2000.

[LRO96] A. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, 1996.

[Mah93] M. J. Maher. A logic programming view of CLP. In *Proc. 10th International Conference on Logic Programming*, pages 737–753, 1993.

[Min00] J. Minker. A logic-based approach to data integration. Technical report, CS-TR4179, University of Maryland at College Park,USA, 2000.

[Mit01] P. Mitra. An algorithm for answering queries efficiently using views. In *Proc. of the 12th Australasian database conference*, 2001.

[MLF00] T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. In *Proc. of 19th PODS*, pages 67–75, 2000.

[MW00] M. Maher and J. Wang. Optimizing queries in extended relational databases. In *LNCS 1873*, pages 386–396, 2000.

[PL00] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, pages 484–495, 2000.

[Qia96] X. Qian. Query folding. In *Proc. of 12th ICDE*, pages 48–55, 1996.

[Top91] R. W. Topor. Safe database queries with arithmetic relations. In *Proc of 14th Australian Computer Science Confnference*, 1991.

[Ull00] J.D. Ullman. Information integration using logical views. *TCS: Theoretical Computer Science*, 239(2):189–210, 2000.