

# Linearly Bounded Reformulations of Conjunctive Databases

Rada Chirkova and Michael R. Genesereth

Stanford University, Stanford CA 94305, USA  
{rada,genesereth}@cs.stanford.edu

**Abstract.** *Database reformulation* is the process of rewriting the data and rules of a deductive database in a functionally equivalent manner. We focus on the problem of automatically reformulating a database in a way that reduces query processing time while satisfying strong storage space constraints. In previous work we have investigated database reformulation for the case of unary databases. In this paper we extend this work to arbitrary arity, while concentrating on databases with conjunctive rules. The main result of the paper is that the database reformulation problem is decidable for conjunctive databases.

## 1 Introduction

In the lifecycle of a database system, there are recurring problems whose solution involves transformations of the database schema and/or queries defined on the schema. Prominent examples are database design, data model translation, schema (de)composition, view materialization, and multidatabase integration. Interestingly, nearly all these problems can be regarded as aspects of the same problem in a theoretical framework that we proceed to describe.

Consider an abstract database transformation problem. Suppose the input to the problem comprises the schema and rules of a deductive database and a set of elementary queries which, together with some algebra, forms a query language on the database. Suppose the objective of database transformation is to build an “optimal” structure of the database with respect to the requirements and constraints that are also provided in the input.

Generally, the transformations of the given database schema and rules need to be performed in such a way that the resulting database satisfies three conditions. First, it should be possible to extract from the transformed database, by means of the input query language, exactly the same information as from the original database. Second, the result should satisfy the input requirements, such as minimizing query processing costs. Finally, the result should satisfy the input constraints; one pervasive constraint is a guarantee of a (low) upper bound on the disk space for storing the transformed database. Notice that since the input does not include a specific database instance, all three conditions must hold for *all* instances of the input database and of the transformed database.

We call this problem *database reformulation* and consider logic-based approaches to its solution; a formal definition of the problem is the first contribution of this paper. Database reformulation is the process of rewriting the data and rules of a deductive database in a functionally equivalent manner: it takes as input the schema and rules of a database and a characterization of a query language, and produces as output new schema and rules, as well as a rewriting of (all elementary queries of) the query language in terms of the schema and rules. Notice that database reformulation, unlike query optimization, rewrites multiple rules, thus amortizing over many queries. By specifying various input requirements and constraints, the database reformulation problem translates into any of the database schema/query transformation problems mentioned above.

We focus on database reformulations whose input requirement is to minimize the computational costs of processing the given elementary queries, under strong storage space constraints that guarantee no more than linear increase in database size. In this formulation, the database reformulation framework is most suitable for dealing with the problems of view materialization and multidatabase integration.

This paper treats the base case where all rules in the reformulation input are conjunctive. We show that for such inputs, the database reformulation problem, in the narrow sense described above, is decidable: we describe an algorithm which outputs a solution if there exists at least one satisfactory reformulation of the

input. This result is the second contribution of this paper. In previous work we have proposed a solution to the reformulation problem for unary databases; notice that the solution described in this paper works for deductive databases that contain relations of arbitrary arity. Our long-term objective is to extend research in database reformulation to deductive databases whose queries and views are formulated in progressively more general standard query languages (datalog and its extensions), as well as to databases with integrity constraints.

In the main text, all proofs have been omitted. The proof of the main result and additional examples can be found in the appendix.

## 2 Preliminaries and Terminology

Our representation of the domain includes a set of *relations*; the set of attributes for a relation is called a *relation schema*. A *database schema*, for a given database  $D$ , is a collection of relation schemas for all stored relations in  $D$ .

A function-free *Horn rule* is an expression of the form

$$p(\bar{X}) : - p_1(\bar{Y}), \dots, p_n(\bar{Z}), \quad (1)$$

where  $p$  and  $p_1, \dots, p_n$  are relation names, and  $\bar{X}, \bar{Y}, \dots, \bar{Z}$  are tuples of variables and constants such that any variable appearing in  $\bar{X}$  appears also in  $\bar{Y} \cup \dots \cup \bar{Z}$ .

A *conjunctive query (view)* is a single non-recursive Horn rule. A *conjunctive query (view) relation* is the relation defined by a conjunctive query (view).

Given a query  $q$ , a query  $q'$  is called a *rewriting* of  $q$  in terms of a set  $\mathcal{V}$  of view relations if  $q$  and  $q'$  are equivalent and  $q'$  contains only literals of  $\mathcal{V}$ , i.e.,  $q'$  is defined in terms of  $\mathcal{V}$ . If a query relation  $q_{\mathcal{V}}$  is defined in terms of a set  $\mathcal{V}$ , and if  $\mathcal{R}_{\mathcal{V}}$  is a set of definitions of view relations in  $\mathcal{V}$ , an *expansion*  $q_{\mathcal{R}_{\mathcal{V}}}$  of  $q_{\mathcal{V}}$ , in terms of  $\mathcal{R}_{\mathcal{V}}$ , is a query obtained by replacing, in the body of  $q_{\mathcal{V}}$ , every occurrence of a view literal  $v_i$  ( $v_i \in \mathcal{V}$ ) by its body in  $\mathcal{R}_{\mathcal{V}}$ , with suitable variable renamings.

In this paper we use the notion of a *substitution*, and in this respect generally follow the terminology of [24]. The substitutions we consider are of the form  $\{ v_1 \leftarrow t_1, \dots, v_n \leftarrow t_n \}$  where all  $v_i$ 's are distinct variables, and each  $t_i$  is either a variable from among  $v_1, v_2, \dots, v_n$  or a constant.

A *containment mapping* [9] from a query  $q_1$  to a query  $q_2$  is a mapping from the variables of  $q_1$  into the variables of  $q_2$ , such that every literal in the body of  $q_1$  is mapped to a literal in  $q_2$ , and the head of  $q_1$  is mapped into the head of  $q_2$ . If both  $q_1$  and  $q_2$  are conjunctive and neither contains built-in predicates, the existence of a containment mapping from  $q_1$  to  $q_2$  is a necessary and sufficient condition for  $q_1$  to contain  $q_2$ ; this result is called the *containment mapping theorem*.

A conjunctive query  $q'$  is a *minimal equivalent* to a conjunctive query  $q$  if it has as few subgoals as any query equivalent to  $q$  [34]. Both minimization and equivalence of conjunctive queries are shown to be NP-complete in [4, 9].

## 3 Database Reformulation

This section gives a definition and a formal specification of the database reformulation problem for the general case where rules are not necessarily conjunctive.

*Database reformulation* is the process of rewriting the data and rules of a deductive database in a functionally equivalent manner; we use the word “reformulation” both for the process and for its output. We focus on the problem of automatically reformulating a database in a way that reduces the processing time for a prespecified set of queries while satisfying strong storage space constraints. The prespecified set of queries is the set of elementary queries in the input query language; see the Introduction for details.

Let us describe the input and the output of the database reformulation process. Consider a set  $\mathcal{P}$  of relation names. Let  $\mathcal{S}$  consist of schemas for some relation names in  $\mathcal{P}$ ;  $\mathcal{S}$  is the input database schema. Let  $\mathcal{R}_{\mathcal{S}}$  be a set of definitions, in terms of  $\mathcal{S}$ , for some relations whose names are in  $\mathcal{P}$ ;  $\mathcal{R}_{\mathcal{S}}$  is the set of rules in the input. Let  $\mathcal{Q}$  be a set of names of all elementary query relations in the input, such that  $\mathcal{Q} \subseteq \mathcal{P}$  and that  $\mathcal{R}_{\mathcal{S}}$  contains definitions of all relations in  $\mathcal{Q}$ .

Now, let  $\mathcal{V}$  consist of schemas for some relation names in  $\mathcal{P}$ ;  $\mathcal{V}$  is the output database schema, i.e., the set of schemas of new stored relations which are materialized in the process of database reformulation. Finally, let  $\mathcal{R}_{\mathcal{V}}$  be a set of views defined in terms of  $\mathcal{V}$ ;  $\mathcal{R}_{\mathcal{V}}$  is the set of rules in the output.

**Definition 1.** For a given triple  $(\mathcal{S}, \mathcal{R}_{\mathcal{S}}, \mathcal{Q})$ , a triple  $(\mathcal{V}, \mathcal{R}_{\mathcal{V}}, \mathcal{Q})$  is a reformulation of  $(\mathcal{S}, \mathcal{R}_{\mathcal{S}}, \mathcal{Q})$  if for each query relation in  $\mathcal{Q}$  with a definition  $q_{\mathcal{S}}$  in  $\mathcal{R}_{\mathcal{S}}$ ,  $\mathcal{R}_{\mathcal{V}}$  contains a rewriting of  $q_{\mathcal{S}}$ .

Let  $D_{\mathcal{S}}$  be an arbitrary database with schema  $\mathcal{S}$ ; let  $D_{\mathcal{V}}$  be a database that consists of the tables for all and only those (materialized, starting from  $D_{\mathcal{S}}$ ) view relations in  $\mathcal{V}$  that are used to define  $\mathcal{Q}$ . For a fixed database schema  $\mathcal{S}$  and for a fixed set of definitions of view relations in  $\mathcal{V}$  in terms of  $\mathcal{S}$ , consider all possible databases  $D_{\mathcal{S}}$  and all corresponding databases  $D_{\mathcal{V}}$ , with sizes (in bytes)  $|D_{\mathcal{S}}|$  and  $|D_{\mathcal{V}}|$  respectively.

**Definition 2.** A reformulation  $(\mathcal{V}, \mathcal{R}_{\mathcal{V}}, \mathcal{Q})$  of an input  $(\mathcal{S}, \mathcal{R}_{\mathcal{S}}, \mathcal{Q})$  is linearly bounded with parameter  $t$ , where  $t$  is a positive constant, if for all pairs  $(D_{\mathcal{S}}, D_{\mathcal{V}})$  the storage space  $|D_{\mathcal{V}}|$  taken up by  $D_{\mathcal{V}}$  is no more than linear in both  $|D_{\mathcal{S}}|$  and  $t$ :

$$|D_{\mathcal{V}}| \leq t |D_{\mathcal{S}}|. \quad (2)$$

One example of a linear bound is the *no-growth bound*; there,  $t = 1$ . The no-growth bound may be too restrictive for some applications.

## 4 Views with Bounded Definition Length

Our objective is to automate the database reformulation process for as general query languages as possible; in other words, we strive to design *reformulation algorithms*. To that end, it is first necessary to understand, for each class of query languages, whether the potentially infinite, for each input, search space of reformulations can be transformed in such a way that it becomes finite but still contains valuable reformulations. One way of making the search space of reformulations more tractable is to restrict the number of view relations that are used to generate rewritings of the input queries.

In this paper we focus on reformulating, in terms of conjunctive views, deductive databases where all rules are conjunctive; in the remainder of the paper we will denote conjunctive queries (views) simply by “queries” (“views”). In the conjunctive case, one might be able to restrict the number of views under consideration by setting an upper bound on the number of subgoals in views. Suppose we could show that in any “good” rewriting of an arbitrary conjunctive input, all participating view relations can be defined by conjunctions of up to a fixed number of subgoals. If this hypothesis were true, the problem of finding all “good” reformulations of the given input would be reduced to the clearly feasible problem of enumerating and combining all “short” views, thereby yielding an *enumeration procedure*. Notice that we do not use the number of subgoals as a cost measure for executing the query.

Consider a database schema  $\mathcal{S}$ , a set  $\mathcal{V}$  of view relations, and a set  $\mathcal{R}_{\mathcal{V}}$  of definitions of relations in  $\mathcal{V}$  in terms of  $\mathcal{S}$ . Consider a query  $q_{\mathcal{S}}$  in terms of  $\mathcal{S}$  and a query  $q_{\mathcal{V}}$  in terms of  $\mathcal{V}$ , with the corresponding expansion  $q_{\mathcal{R}_{\mathcal{V}}}$  in terms of  $\mathcal{R}_{\mathcal{V}}$ . For each view literal  $v_i$  in the body of  $q_{\mathcal{V}}$  let us denote by  $r_i$  the body of  $v_i$  in  $q_{\mathcal{R}_{\mathcal{V}}}$ . Suppose there is a containment mapping from  $q_{\mathcal{R}_{\mathcal{V}}}$  to  $q_{\mathcal{S}}$ ; such a mapping is called *noninterleaving* if no two  $r_i$ s in  $q_{\mathcal{R}_{\mathcal{V}}}$  map into the same subset of the body of  $q_{\mathcal{S}}$ , and if the mapping preserves head variables of all the views involved.

**Theorem 1 (Restricted: New Definitions with Fewer Subgoals).** *If  $q_{\mathcal{S}}$  is minimal,  $q_{\mathcal{S}}$  and  $q_{\mathcal{V}}$  are equivalent, and there is a noninterleaving containment mapping from  $q_{\mathcal{R}_{\mathcal{V}}}$  to  $q_{\mathcal{S}}$ , then there exists a set  $\mathcal{R}'_{\mathcal{V}}$  of (alternative) definitions of the view relations in  $\mathcal{V}$ , such that each definition in  $\mathcal{R}'_{\mathcal{V}}$  has no more subgoals than  $q_{\mathcal{S}}$ .*

Intuitively, the theorem states that for each view used to define  $q_{\mathcal{V}}$ , there exists a “short” definition of the view that has no more subgoals than the original query  $q_{\mathcal{S}}$ ; thus, in the setting of the theorem, one can apply the enumeration procedure described above to generate all views that can be used in any rewriting of the input query.

Unfortunately, the result stated in Theorem 1 holds in an extremely limited setting and thus is not very useful. It would be desirable to make a similar claim for a more general case. Suppose we could show that

if the queries  $q_S$  and  $q_V$  are equivalent then there exists a set  $\mathcal{R}'_V$  of (alternative) definitions of the view relations in  $\mathcal{V}$ , such that each definition in  $\mathcal{R}'_V$  has no more subgoals than  $q_S$ . Notice that this formulation is similar to that of Theorem 1, except that we no longer require that  $q_S$  be minimal or that there exist a noninterleaving mapping from  $q_{\mathcal{R}_V}$  to  $q_S$ .

This conjecture, however desirable, is not true: removing the noninterleaving mapping requirement invalidates the claim. Consider a counterexample.

*Example 1.* Consider a database schema  $\mathcal{S}$  that consists of one binary relation schema  $s$ , and consider a query  $q_S$ :

$$q_S(X, Y) : - s(X, Y), s(Y, X); \quad (3)$$

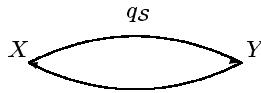
for a graphic depiction of  $q_S$ , see Figure 1.

Consider a set  $\mathcal{V}$  of two view relations  $v_1$  and  $v_2$  with the following definitions:

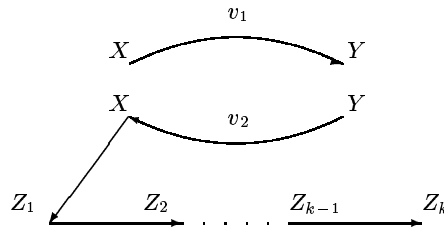
$$v_1(X, Y) : - s(X, Y); \quad (4)$$

$$v_2(Y, X) : - s(Y, X), s(X, Z_1), s(Z_1, Z_2), \dots, s(Z_{k-1}, Z_k); \quad (5)$$

let these definitions of  $v_1$  and of  $v_2$  form a set  $\mathcal{R}_V$ ; for a graphic depiction of the views see Figure 2.



**Fig. 1.** Graphic depiction of the query in Example 1.



**Fig. 2.** Graphic depiction of the views in Example 1.

It is easy to show that the query

$$q_V(X, Y) : - v_1(X, Y), v_2(Y, X) \quad (6)$$

is equivalent to  $q_S$ . At the same time, it is not possible to construct a noninterleaving containment mapping from  $q_{\mathcal{R}_V}$  (the expansion of  $q_V$  in terms of  $\mathcal{V}$ ) to  $q_S$ .

The number of subgoals of  $v_2$  can be made arbitrarily large by varying the parameter  $k$ ; it is clear that neither  $v_1$  nor  $v_2$  is redundant in  $q_V$ , and therefore no upper bound — one that could be related to the length of  $q_S$  — can be established on the number of subgoals in the definition of  $v_2$ .  $\square$

This example allows us to formulate

**Theorem 2 (General: New Definitions with Fewer Subgoals).** *It is not true for all conjunctive queries  $q_S$  and all their rewritings  $q_V$  that each view used in  $q_V$  has an alternative (equivalent) definition which has no more subgoals than  $q_S$ .*

The failure of the conjecture formulated above makes one wonder whether it is possible at all to generalize the claim of Theorem 1. Fortunately, the answer is yes: a much more general result is described in the next section.

## 5 Shorter Views Contained in Longer Views

The main result of the previous section is that it is not possible to rewrite views in an arbitrary formulation of a query, in such a way that each view has no more subgoals than the query itself. However, in this section we show that it is possible to “reformulate” the conjunctive views in an arbitrary rewriting of an arbitrary conjunctive query. This reformulation process outputs views that, although not necessarily equivalent to the original views, constitute an equivalent rewriting of the query and, at the same time, have each no more subgoals than the query itself. Moreover, we show that if, in addition, the input query is linearly bounded with some parameter  $t$  (see Definition 2 in Section 3) then the reformulated query is also linearly bounded with the same parameter  $t$ .

*Example 2.* Recall  $\mathcal{S}$ ,  $q_{\mathcal{S}}$ ,  $\mathcal{V} = \{v_1, v_2\}$ ,  $\mathcal{R}_{\mathcal{V}}$ , and  $q_{\mathcal{V}}$  introduced in Example 1; we choose  $k = 2$  for  $v_2$  which will thus be defined as follows:

$$v_2(Y, X) : - s(Y, X), s(X, Z_1), s(Z_1, Z_2). \quad (7)$$

We define a new view relation  $v'_1$  ( $v'_2$ ) by applying a substitution to the body of  $v_1$  ( $v_2$ ). For  $v'_1$ , consider the trivial substitution applied to the only subgoal  $s(X, Y)$  of  $v_1$ :

$$v'_1(X, Y) : - s(X, Y). \quad (8)$$

For  $v'_2$ , consider a substitution  $\{X \leftarrow X, Y \leftarrow Y, Z_1 \leftarrow Y, Z_2 \leftarrow X\}$  applied to the body of  $v_2$ :

$$v'_2(Y, X) : - s(Y, X), s(X, Y). \quad (9)$$

Notice that  $v'_2$  is *not* equivalent to  $v_2$ .

Now the query  $q_{\mathcal{V}'}$ :

$$q_{\mathcal{V}'}(X, Y) : - v'_1(X, Y), v'_2(Y, X) \quad (10)$$

is equivalent to both  $q_{\mathcal{S}}$  and  $q_{\mathcal{V}}$ , since the expansion  $q_{\mathcal{R}_{\mathcal{V}'}}$  of  $q_{\mathcal{V}'}$  is isomorphic to the query  $q_{\mathcal{S}}$  (after removing duplicate subgoals). It is easy to see that when  $\{v'_1, v'_2\}$  is the schema of the reformulated database (i.e., when both  $v'_1$  and  $v'_2$  are materialized), the query of interest is computed faster than in databases with schema  $\mathcal{S}$ , as well as in databases with schema  $\mathcal{V}$ .

**Table 1.** The stored relation  $s$  and view relations  $v_2$  and  $v'_2$  in Example 2.

$s$	
$a$	$b$
$b$	$a$
$c$	$d$
$d$	$c$
$b$	$c$
$d$	$f$

$v_2$	
$a$	$b$
$b$	$a$
$c$	$d$
$d$	$c$
$b$	$c$

$v'_2$	
$a$	$b$
$b$	$a$
$c$	$d$
$d$	$c$

Consider a database instance  $D$  with schema  $\mathcal{S}$ . Suppose the data in the table for  $s$ , in that database instance  $D$ , is as in Table 1. Table 1 also shows the result of materializing view relations  $v_2$  and  $v'_2$  (both  $v_1$  and  $v'_1$  are the same as  $s$ ). Notice that the size of the table for each  $v'_i$  is no larger than the size of the table for its corresponding  $v_i$ ; this is also true for any other database instance with schema  $\mathcal{S}$ , as will be explained shortly. Thus, if a reformulation that uses the rewriting  $q_{\mathcal{V}}$  is linearly bounded with some parameter  $t$ , then the reformulation that uses the rewriting  $q_{\mathcal{V}'}$  is also guaranteed to be linearly bounded with the same parameter  $t$ .

Notice that  $v'_1$  is not needed to compute  $q_{\mathcal{V}'}$ , and thus the reformulation that contains only  $v'_2$ , in addition to improved computation time, satisfies the no-growth bound compared to the *original* (input) database schema.  $\square$

In formulating our main result and the reformulation algorithm, we will use the notion of a *unifier* of a query:

**Definition 3.** Given a query  $q$ , a unifier of  $q$  is a query  $q'$  such that the body of  $q'$  is the result of applying some substitution to some subset of the body of  $q$ .

Consider, as usual, a database schema  $\mathcal{S}$  and a set  $\mathcal{V}$  of view relations defined in terms of  $\mathcal{S}$ . Consider a query  $q_{\mathcal{S}}$  defined in terms of  $\mathcal{S}$  and a query  $q_{\mathcal{V}}$  defined in terms of  $\mathcal{V}$ . In this setting, the following main result holds.

**Theorem 3.** If  $q_{\mathcal{S}}$  and  $q_{\mathcal{V}}$  are equivalent then, for each view literal  $v_i$  in  $q_{\mathcal{V}}$ , there is a (not necessarily equivalent) view relation  $v'_i$  which is a unifier of  $q_{\mathcal{S}}$ , and a query  $q_{\mathcal{V}'}$  obtained by replacing, in  $q_{\mathcal{V}}$ , each  $v_i$  with its corresponding  $v'_i$ , is equivalent to  $q_{\mathcal{V}}$ ;

in addition, if both  $v_i$  and  $v'_i$  are materialized in any database with the schema  $\mathcal{S}$ , then the size of the table for  $v'_i$  is no larger than the size of the table for its  $v_i$ .

Intuitively, the theorem states that for any rewriting  $q_{\mathcal{V}}$  of a query  $q_{\mathcal{S}}$  we can always find an alternative rewriting  $q_{\mathcal{V}'}$  with two special properties. First,  $q_{\mathcal{V}'}$  is composed entirely of views with no more subgoals each than in  $q_{\mathcal{S}}$ . The second property is that in  $q_{\mathcal{V}'}$ , each  $v'_i$  is contained in its corresponding  $v_i$  (this becomes clear from the proof of the Theorem — see the appendix); therefore, when the view relations in both rewritings are materialized, the tables for the view relations that define  $q_{\mathcal{V}'}$  always “fit in” the space required to store the tables for the view relations that define  $q_{\mathcal{V}}$ .

Consider again Example 2. By the main result, the table for each  $v'_i$  is no larger than the table for its corresponding  $v_i$  in any database instance with schema  $\mathcal{S}$ .

Notice that the theorem is true independently of whether the input query  $q_{\mathcal{S}}$  is minimized.

## 6 A Database Reformulation Algorithm

In this section we describe a database reformulation algorithm for deductive databases where all rules are conjunctive. The algorithm is based on the result described in the previous section: for each space-satisfactory rewriting of a given query there exists another space-satisfactory rewriting, defined in terms of “short” views only, i.e., of those views that have no more subgoals than the query itself. Thus, the idea of the algorithm is, for each input query, to examine all “short” views (obviously there is only a finite number of them) and to construct rewritings of the query by combining such views. The successful rewritings then undergo a storage space check to decide whether they belong to a space-satisfactory reformulation. If such rewritings cannot be found by the algorithm, then, by our results, no rewritings of the query exist.

The reformulation algorithm takes as input a database schema  $\mathcal{S}$ , a set  $\mathcal{R}_{\mathcal{S}}$  of view definitions in terms of  $\mathcal{S}$ , a set  $\mathcal{Q}$  of elementary query relations with definitions in  $\mathcal{R}_{\mathcal{S}}$ , and the value of parameter  $t$  for the linear bound. For each query relation in  $\mathcal{Q}$  with definition  $q_{\mathcal{S}}$  in  $\mathcal{R}_{\mathcal{S}}$ , the algorithm generates a set of rewritings of  $q_{\mathcal{S}}$  by, first, producing all unifiers of  $q_{\mathcal{S}}$  (see Definition 3 in Section 5) as views and combining the views in conjunctive definitions in all possible ways, and, second, by testing the resulting conjunctions for equivalence with  $q_{\mathcal{S}}$ . After all such rewritings of all input queries have been built, the algorithm generates candidate reformulations as all combinations where each candidate reformulation contains one rewriting per input query. Only those candidate reformulation that are linearly bounded for the given value of  $t$ , belong in the output of the algorithm.

**Theorem 4.** For a given input  $(\mathcal{S}, \mathcal{R}_{\mathcal{S}}, \mathcal{Q})$  and for a given positive number  $t$ , the reformulation algorithm outputs only those reformulations  $(\mathcal{V}, \mathcal{R}_{\mathcal{V}}, \mathcal{Q})$  that are linearly bounded with parameter  $t$ , and their  $\mathcal{R}_{\mathcal{V}}$  contain only definitions with no more subgoals in each than the number of subgoals in the query in  $\mathcal{R}_{\mathcal{S}}$  with the longest definition.

The longest query in  $\mathcal{R}_{\mathcal{S}}$  is the query with the most subgoals. This theorem is true by construction of the algorithm.

**Theorem 5.** If, for some input and for some positive number  $t$ , the output of the reformulation algorithm is empty then no conjunctive reformulation of the input is linearly bounded with parameter  $t$ .

This result follows immediately from Theorem 3 in Section 5.

The reformulation algorithm presented in this section is proof that the data-base reformulation problem is decidable for conjunctive databases. Notice that the algorithm is, in all probability, too expensive to be used to actually generate reformulations. To see why, it suffices to notice that one of the steps of the algorithm involves testing pairs of conjunctive queries for equivalence, i.e., solving a known NP-complete problem.

## 7 Related Work

Database schema evolution is an integral part of database design, data model translation, schema (de)composition, and multidatabase integration; fundamental to these problems is the notion of equivalence between database schemata.

The first definition of schema equivalence was proposed in [11]; schema equivalence was studied in [6, 8, 29]. Later, relative information capacity was introduced in [19] as a fundamental theoretical concept which encompasses schema equivalence and dominance. Other work on relative information capacity includes [5, 25, 26]. Tutorial [18] surveys a number of frameworks — relative information capacity among others — for dealing with the issue of semantic heterogeneity arising in database integration.

In practical database systems, database design frequently uses normalization, first introduced in [10] and described in detail in [33]. Papers [7, 20] survey methods and issues in multidatabase integration.

Query transformation is another aspect of database transformation tasks; query rewriting methods complement schema transformation methods in that they are applied to databases that are already operational. Query rewriting is important for query optimization, especially in deductive databases [27] where queries can be complex and the amount of data accessed can be overwhelming. [28] is a survey on implementation techniques and implemented projects in deductive databases.

There is an extensive body of work on theoretical aspects of query rewriting. For an overview of query rewriting methods in datalog and its extensions see [2, 33, 34]. In addition, applications such as data warehousing and multidatabase integration have promoted the study of views in databases. The paper [32] is a survey of containment and rewriting/optimization of queries using views. [1] discusses the complexity of answering queries using materialized views and contains references to major results in the areas of query containment and view materialization. Papers [16, 17, 21, 30] describe approaches to view materialization. [3, 12, 13, 23] treat the problem of using available materialized views for query evaluation. Nearly all results described in the literature concern rewriting of single queries; notice that the database reformulation approach we propose involves simultaneous rewriting of *sets* of queries.

Transformations of database schemas and queries can be considered together as reformulations of logical theories. [31] provides a theoretical foundation for theory reformulations, and [15, 22] contain work on general transformations of logical theories.

Descriptions of basic methods used in this paper can be found, e.g., in [14].

## 8 Conclusions and Future Work

The first contribution of this paper is that it introduces the notion of database reformulation which is the process of rewriting the data and rules of a deductive database in a functionally equivalent manner. We focus on the problem of automatically reformulating a database in a way that reduces query processing time while satisfying strong storage space constraints.

The second contribution of this paper is a proof that it is decidable to reformulate, using conjunctive views, deductive databases where all rules are conjunctive, in the presence of strong storage space constraints. We have shown that for any space-satisfactory rewriting of a conjunctive query there is a rewriting which is also space-satisfactory and is composed entirely of views that have no more subgoals than the original query. We have described a reformulation algorithm which returns space-satisfactory query rewritings composed of views that have no more subgoals than the longest query in the input. Notice that all the results automatically hold for select-project-join (SPJ) queries in SQL.

There are several directions of future research in database reformulation. A pressing research problem is the issue of update complexity in the reformulated database. On a more long-term scale, it is desirable to develop criteria for comparing different outputs produced by the reformulation algorithm, as well as

criteria for choosing the optimal output among the potentially multiple answers. Another challenge is to examine possible interactions between views chosen in support of different queries and to understand how such interactions might influence the quality of a reformulation, in particular its storage space requirements. We also plan to study the complexity of the reformulation problem. Finally, our long-term objective is to extend research in database reformulation to deductive databases whose queries and views are formulated in progressively more general standard query languages, for example include disjunction or negation, as well as to databases with integrity constraints.

## Acknowledgments

The authors would like to thank Jeffrey D. Ullman for pointing out the problem of query folding in the first place, which ultimately led to the database reformulation problem. We would also like to thank Vasilis Vassalos for the reading and extensive discussions of the paper.

## References

1. Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *PODS-98*, pages 254–263.
2. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, Reading, Mass., 1995.
3. F.N. Afrati, M. Gergatsoulis, and T.G. Kavalieros. Answering queries using materialized views with disjunctions. In *ICDT-99*, pages 435–452.
4. A.V. Aho, Y. Sagiv, and J.D. Ullman. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.
5. J. Albert, Y. Ioannidis, and R. Ramakrishnan. Conjunctive query equivalence of keyed relational schemas. In *PODS-97*, pages 44–50.
6. P. Atzeni, G. Ausiello, C. Batini, and M. Moscarini. Inclusion and equivalence between relational database schemata. *Theoretical Computer Science*, 19:267–285, 1982.
7. C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
8. C. Beeri, A.O. Mendelzon, Y. Sagiv, and J.D. Ullman. Equivalence of relational database schemes. *SIAM J. Comput.*, 10(2):352–370, 1981.
9. Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC-77*, pages 77–90.
10. E.F. Codd. A relational model of data for large shared data banks. *Comm. ACM*, 13(6):377–387, June 1970.
11. E.F. Codd. Further normalization of the data base relational model. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice Hall Inc., Englewood Cliffs, NJ, 1972.
12. Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *PODS-97*, pages 109–116.
13. Oliver M. Duschka and Michael R. Genesereth. Query planning with disjunctive sources. In *AAAI-98 Workshop on AI and Information Integration*, 1997.
14. Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, New York, 1972.
15. Fausto Giunchiglia and Toby Walsh. A theory of abstraction. *Artificial Intelligence*, 57(2-3):323–389, 1992.
16. Himanshu Gupta. Selection of views to materialize in a data warehouse. In *ICDT-97*, pages 98–112.
17. Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize under a maintenance cost constraint. In *ICDT-99*, pages 453–470.
18. Richard Hull. Managing semantic heterogeneity in databases: a theoretical perspective. In *PODS-97*, pages 51–61.
19. Richard Hull. Relative information capacity of simple relational database schemata. *SIAM J. Comput.*, 15(3):856–886, August 1986.
20. Won Kim, editor. *Modern Database Systems*. ACM Press, New York, New York, 1995.
21. Yannis Kotidis and Nick Roussopoulos. Dynamat: a dynamic view management system for data warehouses. In *SIGMOD-99*.
22. Alon Y. Levy and P. Pandurang Nayak. A semantic theory of abstractions. In *IJCAI-95*, pages 196–203.
23. A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS-95*, pages 95–104.
24. John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.

25. R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In *VLDB-93*, pages 120–133.
26. R.J. Miller, Y.E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: bridging theory and practice. *Information Systems*, 19(1):3–31, 1994.
27. Jack Minker. Logic and databases: a 20 year retrospective. In D. Pedreschi and C. Zaniolo, editors, *Logic in Databases*, pages 3–57. Springer, 1996. (Proceedings of the LID'96 international workshop).
28. Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of deductive database systems. *J. Logic Progr.*, 23(2):125–149, May 1995.
29. J. Rissanen. On equivalences of database schemes. In *PODS-82*, pages 23–26.
30. K.A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: trading space for time. In *SIGMOD-96*, pages 447–458.
31. Devika Subramanian. *A theory of justified reformulations*. PhD thesis, Stanford University, 1989.
32. Jeffrey D. Ullman. Information integration using logical views. In *ICDT-97*, pages 19–40.
33. Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, New York, 1988.
34. Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, New York, 1989.

## A Some Examples and Proofs

### A.1 Example: Recursive Reformulation

Here is an example which illustrates the power of database reformulation for a deductive database with recursive rules.

*Example 3.* Consider the "founding fathers" problem invented by Devika Subramanian [31]. There is just one stored relation in this case: the *father* relation, where *father*( $X, Y$ ) means that  $X$  is the father of  $Y$ . In addition, there are two views defined as shown below. The (male) *ancestor* relation holds of two people  $X$  and  $Y$  if  $X$  is the father of  $Y$  or if there is a third person  $V$  who is a child of  $X$  and an ancestor of  $Y$ . The *samefamily* relation holds of two people if they have a common (male) ancestor.

$$\text{ancestor}(X, Y) : - \text{father}(X, Y); \quad (11)$$

$$\text{ancestor}(X, Y) : - \text{father}(X, V), \text{ancestor}(V, Y); \quad (12)$$

$$\text{samefamily}(Y, Z) : - \text{ancestor}(X, Y), \text{ancestor}(X, Z). \quad (13)$$

With the definitions above, determining whether two people are in the same family involves computing all of their ancestors and intersecting the two sets, a fairly expensive operation.

To improve the query processing time, we could materialize the *samefamily* relation. However, the table for that relation, in a database with schema { *father* }, would take up storage space that is, in the worst case, quadratic in the number of people in the database. For large databases, such materialization is impractical.

On the other hand, it is possible to get the same performance gain without any space growth whatsoever. The definitions below show how. We say that a person  $X$  is the founding father (*founder*) of the family of person  $Y$  if  $X$  is an ancestor of  $Y$  and there is no person  $W$  such that  $W$  is the father of  $X$  ( $\neg$  stands for negation). The *samefamily* relation can then be defined in terms of this new *founder* relation: two people are in the same family if and only if they have the same founding father.

$$\text{hasancestors}(X) : - \text{father}(W, X); \quad (14)$$

$$\text{founder}(X, Y) : - \text{ancestor}(X, Y), \neg \text{hasancestors}(X); \quad (15)$$

$$\text{samefamily}(Y, Z) : - \text{founder}(X, Y), \text{founder}(X, Z). \quad (16)$$

Using the definitions above, we can materialize the *founder* relation; furthermore, if we are interested only in the *samefamily* query we can also dematerialize the *father* relation. With such a reformulation we get the same performance improvement for our target query as if we did materialize the *samefamily* relation itself. However, the amount of space required for the reformulated database is the same as that required for the input database; in other words, this reformulation satisfies the no-growth bound. This reformulation is so good because we have succeeded in defining an equivalence relation (*samefamily*) by choosing a single representative (*founder*) from each equivalence class.  $\square$

## A.2 Example: Conjunctive Reformulation

Below is a very simple example of database reformulation for a conjunctive deductive database.

*Example 4.* Suppose in some database there are two stored relations: *parent* and *gender* (either male or female). Suppose the only elementary query of interest is *grandparent* — imagine that we only plan to ask queries about grandfathers or grandmothers. The *grandparent* relation can be defined in terms of the *parent* relation in an obvious way as follows:

$$\text{grandparent}(X, Y) :- \text{parent}(X, Z), \text{parent}(Z, Y). \quad (17)$$

Once we materialize the *grandparent* relation, we can dematerialize the *parent* relation since we are not interested in querying on that relation anyway. As a result, the new database contains only relevant information (relations *gender* and *grandparent*), the processing costs for the queries of interest are minimized, and the reformulation is linearly bounded with  $t = 2$ .  $\square$

## A.3 Proof of Theorem 3 in Section 5

To prove the theorem, we need the notion of a self-map.

**Definition 4.** A containment mapping on a conjunctive query is called a self-map if it maps the query into itself.

Consider an arbitrary conjunctive query  $q$ ; consider another query  $\tilde{q}$  such that there exist two containment mappings  $M_1 : q \rightarrow \tilde{q}$  and  $M_2 : \tilde{q} \rightarrow q$ , where both  $M_1$  and  $M_2$  preserve head variables of the queries. Consider a composition  $M$  of the two mappings;  $M : q \rightarrow q$ . Notice that  $M$  is a self-map by construction.

Then it is easy to show that each such  $q$  has the following property:

**Lemma 1.**  $q$  is equivalent to its image  $q'$  under  $M$ :

$$q \equiv q'. \quad (18)$$

We proceed to prove the main result which is formulated as Theorem 3 in Section 5:

*Proof (Theorem 3).* Let  $\mathcal{R}_\mathcal{V}$  be the set of view definitions for the view relations in  $\mathcal{V}$  in terms of  $\mathcal{S}$ , and let  $q_{\mathcal{R}_\mathcal{V}}$  be the (unique and equivalent) expansion of  $q_\mathcal{V}$  in terms of  $\mathcal{R}_\mathcal{V}$ . By transitivity of equivalence,  $q_{\mathcal{R}_\mathcal{V}}$  is equivalent to  $q_\mathcal{S}$ .

From the containment mapping theorem, the equivalence of  $q_{\mathcal{R}_\mathcal{V}}$  and of  $q_\mathcal{S}$  means that there is a containment mapping  $M_{\mathcal{R}_\mathcal{S}} : q_{\mathcal{R}_\mathcal{V}} \rightarrow q_\mathcal{S}$  and there is a containment mapping  $M_{\mathcal{S}\mathcal{R}} : q_\mathcal{S} \rightarrow q_{\mathcal{R}_\mathcal{V}}$ , where both mappings preserve head variables of the queries.

Consider a composition  $M$  of  $M_{\mathcal{R}_\mathcal{S}}$  with  $M_{\mathcal{S}\mathcal{R}}$ :

$$M : q_{\mathcal{R}_\mathcal{V}} \rightarrow q_{\mathcal{R}_\mathcal{V}}. \quad (19)$$

By construction,  $M$  is a self-map that satisfies the conditions of Lemma 1; therefore, by that lemma, the image  $M(q_{\mathcal{R}_\mathcal{V}})$  of  $q_{\mathcal{R}_\mathcal{V}}$  under  $M$  is equivalent to  $q_{\mathcal{R}_\mathcal{V}}$ :

$$q_{\mathcal{R}_\mathcal{V}} \equiv M(q_{\mathcal{R}_\mathcal{V}}). \quad (20)$$

Consider an arbitrary view literal  $v_i$  in  $q_\mathcal{V}$  and the body  $r_i$  of the definition of  $v_i$  in  $q_{\mathcal{R}_\mathcal{V}}$ . Consider the image  $r'_i$  of  $r_i$  under  $M$ ;  $r'_i$  is a subset of the body of  $M(q_{\mathcal{R}_\mathcal{V}})$ . Notice that by construction of  $M$ ,  $r'_i$  defines some (at least one — depending on the choice of head variables) unifier of  $q_\mathcal{S}$  (see Definition 3 in Section 5).

Let us take  $r'_i$  as the body of a definition of some view relation  $v'_i$ , such that the head variables of  $v'_i$  are images, under  $M$ , of the head variables of  $v_i$  (in  $r_i$ ).

After we have built a new relation  $v'_i$  based on each  $v_i$  used in the definition  $q_{\mathcal{V}}$ ,  $M(q_{\mathcal{R}_{\mathcal{V}}})$  can be viewed as an expansion of some conjunctive query  $q_{\mathcal{V}'}$ , where  $\mathcal{V}' = \bigcup \{v'_i\}$ . It is easy to show that  $q_{\mathcal{V}'}$  exists and is equivalent to  $M(q_{\mathcal{R}_{\mathcal{V}}})$ ; therefore, from transitivity of equivalence,  $q_{\mathcal{V}'}$  is equivalent to both  $q_{\mathcal{V}}$  and to  $q_{\mathcal{S}}$ .

Consider an arbitrary view literal  $v_i$  in the body of  $q_{\mathcal{V}}$  and its counterpart  $v'_i$  in the body of  $q_{\mathcal{V}'}$ . Notice that  $v'_i$  is, in fact, constructed as a unifier of  $v_i$ , not of  $q_{\mathcal{S}}$ , although  $v'_i$  is also a unifier of  $q_{\mathcal{S}}$  by properties of the mapping  $M$ . Also, by the containment mapping theorem,  $v'_i$  is contained in  $v_i$ ; if  $M$  does not preserve head variables of some  $v_i$ , we assume that  $v'_i$  has the same number of head variables as  $v_i$ , only some head variables of  $v'_i$  may be unified with each other. Thus we have that in any database instance with schema  $\mathcal{S}$ , if both  $v_i$  and  $v'_i$  are materialized, then the table for  $v'_i$  takes up at most as much storage space as the table for its corresponding  $v_i$ .  $\square$