# Composing Mappings Among Data Sources

Jayant Madhavan
University of Washington
jayant@cs.washington.edu

Alon Y. Halevy
University of Washington
alon@cs.washington.edu

**Abstract**

Semantic mappings between data sources play a key role in several data sharing architectures. Mappings provide the relationships between data stored in different sources, and therefore enable answering queries that require data from other nodes in a data sharing network. Composing mappings is one of the core problems that lies at the heart of several optimization methods in data sharing networks, such as caching frequently traversed paths and redundancy analysis.

This paper investigates the theoretical underpinnings of mapping composition. We study the problem for a rich mapping language, GLAV, that combines the advantages of the known mapping formalisms global-as-view and local-as-view. We first show that even when composing two simple GLAV mappings, the full composition may be an *infinite* set of GLAV formulas. Second, we show that if we restrict the set of queries to be in $CQ_k$ (a common restriction in practice), then we can always encode the infinite set of GLAV formulas using a finite representation. Furthermore, we describe an algorithm that given a query and a finite encoding of an infinite set of GLAV formulas, finds all the *certain answers* to the query. Consequently, we show that for a commonly occuring class of queries it is possible to pre-compose mappings, thereby potentially offering significant savings in query processing.

## 1 Introduction

The problem of sharing data from multiple sources within or between enterprises has recently received significant attention in research and in the commercial world. Over the years, a succession of architectures for sharing data have been proposed, beginning with federated databases [27], followed by data integration systems [30, 14, 18], peer data management systems [12, 10, 2, 16, 17, 25], and data exchange systems [24, 26, 8]. A key element in all of these architectures is the specification of *semantic mappings* between data sources, or between sources and mediated schemas. The semantic mappings describe the relationships between the terms used in two or more schemas. In the past, research has focused on the development of languages for specifying semantic mappings [18, 21], and algorithms that use mappings to answer queries in data sharing systems (see [28, 18, 11] for surveys).

**Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003**

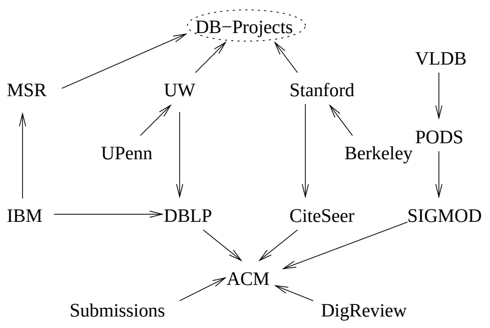

Figure 1: The topology of a data sharing network of sources related to database research.

This paper considers a new problem, namely the problem of composing semantic mappings. Specifically, given semantic mappings between data sources $A$ and $B$, and between $B$ and $C$, is it possible to generate a direct mapping between $A$ and $C$ that is equivalent to the original mappings? Equivalence means that for any query in a given class of queries $\mathcal{Q}$, and for any instance of the data sources, using the direct mapping yields exactly the same answer that would be obtained by the two original mappings.

### 1.1 Motivations

There are several independent motivations for the study of mapping composition. The main motivation for our work comes from query processing and optimization in peer-data management systems (PDMS) (and in particular, the Piazza System [12, 10]). Figure 1 shows the topology of an example PDMS in the domain of database research. In a PDMS each node can be a data source, a logical mediator, or both. Each node has its own schema, and the pairwise semantic mappings (denoted by the arrows in the figure) enable reformulating a query posed on one node to queries on its neighbors. As such, the nodes can share data without a central logical schema. Given a query on a particular node, query processing proceeds by iteratively reformulating the query using the semantic mappings until all the relevant data sources are reached [12]. In a sense, we *chain* together mappings at query time. Note that different paths between a pair of nodes may yield different sets of answers, and hence the maximal answer is obtained by following *all* possible acyclic paths.

Chaining mappings at run-time may be expensive because we may need to follow long and possibly redundant paths in the network. Furthermore, the resulting reformulations may contain significant redundancies or may not lend themselves to efficient query execution plans. (This is in the same spirit as getting better query execution plans by unnesting queries in SQL). Furthermore, if certain nodes leave the network, then we may lose valuable paths (even temporarily).

Addressing these issues raises several static analysis questions regarding the network of mappings, and mapping composition lies at the core of them all. First, we would like to develop a set of techniques that may judiciously pre-compose a select set of mapping chains in the network. By pre-computing the composition, we can also remove redundancies from it, leading to significant run-time savings. Second, we would like to find redundant paths in the network: two paths between a pair of nodes $A$ and $B$ are equivalent if given any query on $A$, reformulating the query along both paths will result in equivalent queries on $B$. Third, we note that data from source $A$ can be used in source $B$ only if the necessary concepts are modeled in each of the nodes on the path between $A$ and $B$. As a result, when paths in the network get longer, we may witness *information loss*. Hence, we would like to determine whether a path between $A$ and $B$ can possibly be useful for some query, and if not, find the weak links and try to improve the mappings there. To address any of these questions, we must first understand how to compute a mapping that represents a path, i.e., a composition of pairwise mappings.

A second motivation for mapping composition comes from the area of model management [3, 23]. The goal of model management is to provide an algebra for explicitly manipulating schemas and mappings between them. Based on such an algebra, we can build a system in which common meta-data tasks can be solved much more effectively. One of the basic operators in a model-management algebra is composition. In [3, 23], models and mappings are treated mostly as syntactic objects. Here we show how to compose mappings in a particular mapping language, and show that considering the semantic aspects of mappings raises several subtleties. As a final motivation, given the pervasive role that semantic mappings play in many systems, the question of composing them arises naturally.

## 1.2 Contributions

We consider the composition problem for a rich mediation language. Specifically, there are three main formalisms proposed for specifying semantic mappings (see [11, 18] for surveys). In the first, called *global-as-view* (GAV), the target schema is described as a set of views over the source schemas. In the second, *local-as-view* (LAV), the data sources are described as views over the target schema. This paper considers the mapping composition problem for the GLAV formalism [9, 8, 12], which combines the practical benefits of both GAV and LAV.

The contributions of this paper are the following. We begin by showing that even for relatively simple GLAV mappings, the composed mapping may be an *infinite* set of GLAV formulas. This means that in general, it may not be possible to obtain the aforementioned advantages of composition. We proceed, in several steps, to identify cases in which composition can be done. First, we describe an algorithm that encodes an infinite number of GLAV formulas in the composition using a finite structure. The algorithm works by building the formulas in the composition in increasing size, and associating *residues* with each formula. When two formulas have isomorphic residues they can be extended in the same ways. When there is a finite number of residues, our algorithm is guaranteed to terminate and to encode the *exact* composition. The algorithm also enables us to provide upper and lower complexity bounds on the problem of determining whether a given finite set of formulas is equivalent to a composition of the original mappings.

Second, we show that for the class of $CQ_k$ queries, there is a finite set of residues, and therefore it is possible to pre-compute the entire composition. Informally, $CQ_k$ is the class of conjunctive queries in which every nested expression has at most $k$ variables. $CQ_k$ queries cover many queries encountered in practice, and for that reason, they have also been studied in the past and shown to have other interesting properties [15, 29, 6]. Finally, to complete the picture, we show that given an infinite number of GLAV formulas encoded by our finite structure, it is possible to find all the answers to a given query. This query answering algorithm, which is of independent interest, generalizes a previous result [19] which showed how to answer queries using an infinite set of views, but only in the context of LAV formulas. In summary, the paper provides significant insights into the problem of mapping composition, and establishes several practical conditions under which compositions can be pre-computed and therefore optimized.

It is important to emphasize that the challenge in designing a mapping composition algorithm is that the composition needs to yield an equivalent answer for every data instance and *every query* in the language $\mathcal{Q}$ over $C$. This is very different from a query rewriting algorithm in which a particular query is *given* as input. In fact, we are aware of only one recent work [20] in the same spirit – there the goal was to show that two sets of views are equivalent for a LAV mapping, *i.e.* they would produce the same set of certain answers for any query. We are not aware of any work addressing the mapping composition problem.

We note that this paper is *not* about choosing which mappings to compose, but rather studying when mappings can be composed without losing information.

The paper is organized as follows. Section 2 sets up the problem, and Section 3 describes our mapping composition algorithm. Section 4 discusses composition for $CQ_k$ queries, and Section 5 describes query answering with the composed mapping. Section 6 concludes.

The complete proofs of our theorems are omitted due to space limitations, but they are available at [22].

## 2 Problem definition

In this section we define the problem of mapping composition, and explain the challenges involved in developing a composition algorithm. We begin by defining the terminology used throughout the paper.

### 2.1 Schemas and queries

Our discussion assumes data is represented in the relational model. Given a data source $A$, $R_A$ refers to its schema. We denote the relations in the schema $R_A$ by lowercase letters, e.g., $a, a_1$. Queries are assumed to be conjunctive (i.e., select, project, join), and we assume that they do not contain comparison predicates (e.g., $\neq$, $<$), hence only equi-joins are allowed. Views are named queries. When queries refer to views, the *unfolding* of the query refers to the query resulting from replacing the view atoms by the subgoals in their definitions (and using fresh variables for the existential variables in the view definition). We use the following standard notation for conjunctive queries:

$$Q(\bar{X}) :- p_1(\bar{X}_1),\ \ldots,\ p_n(\bar{X}_n)$$

$\bar{X}, \bar{X}_1, \ldots, \bar{X}_n$ are tuples of variables, and $\bar{X} \subseteq \bar{X}_1 \cup \ldots \cup \bar{X}_n$. The atoms $p_1(\bar{X}_1),\ \ldots,\ p_n(\bar{X}_n)$ are called the subgoals in the body of the query, and $Q(\bar{X})$ is the head of the query. The variables $\bar{X}$ are the *distinguished* variables, and the others are *existential.* Given a query $Q$ and a database instance $D$, $Q(D)$ denotes the result of evaluating $Q$ over $D$. Note that in the notation of conjunctive queries, joins are expressed by multiple occurrences of the same variable.

In our discussion, we also make use of two restricted classes of conjunctive queries. The class $CQ_k$ is the class of conjunctive queries that can be written by a set of nested expressions, each of which includes at most $k$ variables. For our purposes, the following definition of $CQ_k$ is most enlightening: $CQ_k$ queries can be written by non-recursive datalog program, where (1) each datalog rule contains at most $k$ variables, and (2) each predicate is defined by at most one rule.

**Example 1:** To illustrate $CQ_k$ queries, consider a simple query looking for a *chain* of length $n$ ($n > k$) in a database:

$$q(X,Y) :- e_1(X,X_1),\ldots,e_i(X_{i-1},X_i),\ldots,e_n(X_{n-1},Y)$$

This query can be written as a set of rules, each with 3 variables, where each rule corresponds to a nested expression. Hence, the query is in $CQ_3$:

$q_{n-1}(X_1,Y) :- e_{n-1}(X_1,Z), e_n(Z,Y)$
$q_{n-2}(X_1,Y) :- e_{n-2}(X_1,Z), q_{n-1}(Z,Y)$
...
$q(X,Y) :- e_1(X,Z), q_2(Z,Y)$ □

The class $linCQ_k$ is the subset of $CQ_k$ with a linear nesting of subexpressions. Formally, it is the class of $CQ_k$ queries where each datalog rule has at most one subgoal of an IDB predicate. The chain query above is in $linCQ_3$. Similarly, it is possible to show that cycle and star queries (as long as they select at most $k$ attributes), as well as many other queries encountered in practice, are also in $CQ_k$.

### 2.2 Semantic mediation

Our work is concerned with systems that provide access to multiple data sources spread throughout a network. No matter what the specific topology of such a network, the key element is how we describe the semantic relationships between different data sources. In this paper we employ the GLAV [8, 12] formalism. A semantic mapping between two data sources $A$ and $B$ in GLAV is specified by a set of *mapping formulas*, each of the form $Q_A(\bar{X}) \subseteq Q_B(\bar{X})$, where $Q_A$ and $Q_B$ are conjunctive queries over $R_A$ and $R_B$ respectively. We denote a mapping between $A$ and $B$ by $M_{A\to B}$.

For brevity, we sometimes slightly abuse notation by specifying only the bodies of $Q_A$ and $Q_B$. The variables that appear in both bodies are assumed to be the head variables in both.

The mapping below states that the tuples of relation $a$ in $A$ are a subset of paths of $b$ of length 2 in $B$.

$$M_{A\to B} \quad = \quad \{a(x,y) \subseteq b(x,x_1), b(x_1,y)\}$$

We say that database instance $D_B$ of $R_B$ is consistent with a database $D_A$ of $R_A$, with respect to mapping formula $Q_A(\bar{X}) \subseteq Q_B(\bar{X})$, if the containment holds true when the queries $Q_A$ and $Q_B$ are evaluated over $D_A$ and $D_B$, respectively. Hence, a given database instance $D_A$ defines a *set* of instances $\mathcal{D}_B$ that are consistent with $D_A$ with respect to every mapping formula in $M_{A\to B}$.

The semantics of answering queries in the presence of mappings is given by *certain answers* [1]. Given a query $Q$ over $R_B$, a tuple $\bar{t}$ is a certain answer to $Q$ w.r.t. $M_{A\to B}$ if $\bar{t} \in Q(D)$ for *every* $D \in \mathcal{D}_B$. In a similar fashion, it is possible to extend the definition of certain answers to the case where we are also given an instance of $R_B$, and the case where we have semantic mappings between multiple data sources [12].

We note that the GAV and LAV formalisms are special cases of GLAV. GAV is obtained when $Q_B$ is a single atom and has no projections, and LAV is obtained when $Q_A$ has that form. However, their advantages complement each other: while LAV facilitates relating many data sources to one target, GAV enables expressing joins on attributes that may not appear in outside a local source. Hence, in practice, GLAV is the most useful.

Given a semantic mapping and an instance of $R_A$, we answer $Q$ by computing the *maximally-contained rewriting* of $Q$ over $R_A$. The maximally-contained rewriting is a query $Q'$ over $R_A$ such that $Q'(D_A)$ is guaranteed to be the set of all certain answers to $Q$ for any instance $D_A$. Algorithms for producing maximally-contained rewritings are surveyed in [11]. We note that for certain query languages a maximally-contained rewriting may not yield all certain answers [1, 4], but in our context they do.

### 2.3 Mapping composition

The problem we address in this paper is the following. Suppose we have three data sources, $A$, $B$ and $C$, and two mappings $M_{A\to B}$ and $M_{B\to C}$. We are interested in computing a direct mapping $M_{A\to C}$ that is guaranteed to be equivalent to the two original mappings. Formally, the problem is as follows.

**Definition 1.** *The mapping $M_{A\to C}$ is a* **composition** *of the mappings $M_{A\to B}$ and $M_{B\to C}$ w.r.t. a query language $\mathcal{Q}$ if for all databases $D_A$ for $R_A$, and for all queries $Q$ over $R_C$ such that $Q$ is in the language $\mathcal{Q}$, the certain answers for $Q$ w.r.t. $M_{A\to C}$ are the same as the certain answers w.r.t. $M_{A\to B}$ and $M_{B\to C}$.* □

Note that Definition 1 does not define a unique composition. Instead, it defines what it means for a set of formulas to be one of several equivalent compositions. Unless otherwise mentioned, we are interested in composition w.r.t. the set of conjunctive queries. We illustrate mapping composition with two examples.

**Example 2:** Let the schemas $R_A$, $R_B$ and $R_C$ each have a single binary relation $a$, $b$ and $c$ respectively. Consider the two mappings,

$$\begin{aligned} M_{A\to B} &= \{a(x,y) \subseteq b(x,x_1), b(x_1,y)\} \\ M_{B\to C} &= \{b(x,x_1), b(x_1,x_2), b(x_2,y) \subseteq c(x,y)\} \end{aligned}$$

If $b$ encodes all the edges of a graph $G$, then $M_{A\to B}$ states that $a$ is a subset of the node pairs with paths of length 2 in $G$. Similarly, $M_{B\to C}$ states that all node pairs with paths of length 3 are a subset of $c$. Note that, for brevity, we later use the notation $v_a(x,y) \subseteq v_b^a(x,y)$ for the formula in $M_{A\to B}$, and $v_b^c(x,y) \subseteq v_c(x,y)$ for the formula in $M_{B\to C}$.

The composition of $M_{A\to B}$ and $M_{B\to C}$ consists of the three formulas:

$$a(x,x_1), a(x_1,x_2) \subseteq c(x,y_1) \quad (1a)$$
$$a(x_1,x_2), a(x_2,x) \subseteq c(y_1,x) \quad (1b)$$
$$a(x,x_1), a(x_1,x_2), a(x_2,y) \subseteq c(x,y_1), c(y_1,y) \quad (1c)$$

Formula 1a captures the fact that if there is a path of length 4 in $b$ emanating from $x$ (guaranteed by the left hand-side and $M_{A\to B}$), then there is a path of length 3 emanating from $x$, which according to $M_{B\to C}$, means that $x$ is in the projection of $c$ on its first column. Formula 1b is similar, but with paths ending at $x$. Formula 1c shows that even though the composition cannot obtain facts for $c(x,y)$, we *can* obtain the end-points of paths of length two in $c$, by following paths of length 6 in $b$, which, in turn, are obtained by paths of length 3 in $a$. Intuitively these formulas capture all the relationships between $R_A$ and $R_C$, and any other relationships follow from these. □

Example 2 illustrates the key difficulty in constructing a composition. It does not suffice to consider only composition formulas whose right-hand side are the $c$-views that appear on the right-hand side of formulas in $M_{B\to C}$. The composition may require formulas with more complex $c$-views, as in Formula 1c. The key challenge is to *bound* the set of $c$-views that are considered. In fact, the following example shows that the situation is even more subtle; the set of $c$-views may not even be finite.

**Example 3:** Consider the following mappings. Here, the graph encoded by $R_B$ has red edges (relation $b_r$) and green edges (relation $b_g$).

$$\begin{aligned} M_{A\to B} &= \{a_{rg}(x,y) \subseteq b_r(x,x_1), b_g(x_1,y) \\ &\qquad a_{gg}(x,y) \subseteq b_r(x,x_1), b_g(x_1,y)\} \\ M_{B\to C} &= \{b_r(x,x_1), b_g(x_1,x_2), b_g(x_2,y) \subseteq c_{rgg}(x,y) \\ &\qquad b_g(x,x_1), b_g(x_1,y) \subseteq c_{gg}(x,y)\} \end{aligned}$$

As in the earlier example, $a_{rg}$ is a subset of the node pairs with red-green paths. The other relations $a_{gg}, c_{rgg}$ and $c_{gg}$ can be described similarly. Observe that the following sequence of formulas are all in the composition of $M_{A\to B}$ and $M_{B\to C}$:

$$\begin{aligned}
a_{gg}(x,y) &\subseteq c_{gg}(x,y) && \text{(2a)}\\
a_{rg}(x,x_1), a_{gg}(x_1,x_2) &\subseteq c_{rgg}(x,y_1) && \text{(2b)}\\
a_{rg}(x,x_1), a_{gg}(x_1,x_2), &\subseteq c_{rgg}(x,y_1), c_{gg}(y_1,y_2) && \text{(2c)}\\
a_{gg}(x_2,x_3) & && \\
a_{rg}(x,x_1), a_{gg}(x_1,x_2), &\subseteq c_{rgg}(x,y_1), c_{gg}(y_1,y_2), && \text{(2d)}\\
\ldots, a_{gg}(x_n,x_{n+1}) & \quad\ \ \ldots, c_{gg}(y_{n-1},y_n) &&
\end{aligned}$$

The sequence is infinite. Each equation in the infinite sequence captures the formula that a path comprising of one red ($b_r$) edge followed by $2n+1$ green ($b_g$) edges (the query over $R_A$) is contained within a path comprising of a red edge followed by $2n+2$ green edges (the query over $R_C$). None of them can be expressed in terms of the others (*e.g.* the rule 2c is not implied by rules 2a and 2b). Fortunately, as we will see later, in some cases (including this one) there is a finite encoding of the infinite set of GLAV formulas. □

Before proceeding, we remark that the definitions and results we present in the paper apply to multiple levels of composition, but our discussion focuses on the composition of two mappings.

**Remark 1.** In a sense, the union of the formulas being composed is already an *implicit* representation of the composition. However, as pointed out earlier, the direct representation of the composition has several computational advantages in a PDMS, such as yielding more efficient reformulation, better query execution plans (e.g., minimizing the number of joins), and pruning redundant paths in a PDMS. One of our goals is to identify cases in which it is possible to produce the entire composition ahead of time, and therefore optimize it in advance. One of the key results of this paper is to show that if we restrict the set of queries to those in $CQ_k$, then we can produce the entire composition. Fortunately, $CQ_k$ queries are representative of many queries encountered in practice.

As a particular case in point, we can consider representing the composition as a set of *inverse rules* [7]. As shown in [12], given a query $Q$, we can reformulate it into a datalog program that accesses the base data. The datalog program includes the query $Q$ as one rule, and a set $\mathcal{R}$ of inverse rules that essentially invert the LAV-style mediation formulas. Hence, one could try to optimize the rules in $\mathcal{R}$ ahead of time. However, because of the special structure of inverse rules, they can only be optimized *after* the query $Q$ is given, and hence are of limited use for composition. In fact, our techniques can be viewed as a method for optimizing a set of inverse rules in advance of the query. □

## 3 Mapping composition algorithm

In this section we describe our mapping composition algorithm. In order to explain some of the basic terms used in the algorithm, we begin by explaining how

$$\begin{array}{ll}
M_{A\to B} & v_a:\ a(x,y)\ \subseteq\ v_b^a:\ b(x,x_1), b(x_1,y)\\
M_{B\to C} & v_b^c: b(x,x_1), b(x_1,x_2), b(x_2,y)\ \subseteq\ v_c: c(x,y)\\
 & \\
Q(x,y) & q_c(x,y)\ :-\ c(x,y_1), c(y_1,y)\\
 & \Downarrow\\
Rew_C(Q) & q'_c(x,y)\ :-\ v_c(x,y_1), v_c(y_1,y)\\
 & \downarrow\\
Query_B(Q) & q_b(x,y)\ :-\ v_b^c(x,y_1), v_b^c(y_1,y)\\
 & q_b(x,y)\ :-\ b(x,z_1), b(z_1,z_2), b(z_2,y_1),\\
 & \qquad b(y_1,z_3), b(z_3,z_4), b(z_4,y)\\
 & \Downarrow\\
Rew_B(Q) & q'_b(x,y)\ :-\ v_b^a(x,z_2), v_b^a(z_2,z_3), v_b^a(z_3,y)\\
 & \downarrow\\
Query_A(Q) & q_a(x,y)\ :-\ v_a(x,z_2), v_a(z_2,z_3), v_a(z_3,y)\\
 & q_a(x,y)\ :-\ a(x,z_2), a(z_2,z_3), a(z_3,y)
\end{array}$$

Figure 2: Composing $M_{A\to B}$ and $M_{B\to C}$ in Example 2 w.r.t. a single query as a sequence of reformulations.

mappings can be composed for a *single* query (Section 3.1). In Section 3.2 we show that our algorithm need only consider composition formulas that are *minimal*, and that such minimal formulas can be constructed in increasing size. Given those observations, we introduce *mapping residues* (Section 3.3) that determine when two minimal formulas can be extended in similar ways. In Section 3.4 we put this all together and describe *query rewrite graphs*, that represent all the minimal formulas, and whose construction terminates based on comparing residues of its nodes.

### 3.1 Composing for a single query

As a basis for the discussion of our composition algorithm, we first describe how to compose $M_{A\to B}$ and $M_{B\to C}$ for a *single* query, $Q$, over $R_C$. Informally, we proceed in two steps. In the first we reformulate the query using $M_{B\to C}$, and in the second, we reformulate the result using $M_{A\to B}$. Each of the steps has two parts; first we reformulate the query using the right-hand side of the formulas, and then we replace the views on the right-hand side with those appearing on the left. We illustrate this process in Figure 2 for the query $Q(x,y) :- c(x,y_1), c(y_1,y)$ and the mappings in Example 2. We proceed by computing the following queries:

$\mathbf{Rew_C(Q)}$: the maximally-contained rewriting of $Q$ in terms of the views on the right-hand sides of the formulas in $M_{B\to C}$.

$\mathbf{Query_B(Q)}$: a query over $R_B$ obtained by replacing the views in $Rew_C(Q)$ by the corresponding views on the left-hand sides of the formulas in $M_{B\to C}$, and unfolding these view definitions.

$\mathbf{Rew_B(Q)}$: the maximally-contained rewriting of $Query_B(Q)$ using the views on the right-hand sides of the formulas in $M_{A\to B}$.

$\mathbf{Query_A(Q)}$: a query over $R_A$ obtained by replacing the views in $Rew_B(Q)$ by the corresponding views on the left-hand sides of the formulas in $M_{A\to B}$, and unfolding these view definitions.

The following proposition is a simple corollary of Theorem 4.2 in [1]. It says that the reformulations above will provide all the certain answers to a given query.

**Proposition 1.** *Let $Q$ be a conjunctive query over $R_C$, and let $Q'$ be the union of conjunctive queries $Query_A(Q)$. Then for any instance $D_A$ of $R_A$, $Q'(D_A)$ is the set of all certain answers to $Q$ given $D_A$.* $\Box$

Proposition 1 also provides the first characterization of the composition of $M_{A\to B}$ and $M_{B\to C}$ as a set of GLAV formulas:

**Proposition 2.** *Let $\mathcal{C}$ be the set of all GLAV formulas of the form $Q_A(\bar{x}) \subseteq Q_C(\bar{x})$, where:*

- *$Q_C(\bar{x})$ is a conjunctive query over $R_C$, and*
- *$Q_A(\bar{x})$ is one of the conjunctive queries in $Query_A(Q_C(\bar{x}))$.*

*Then, $\mathcal{C}$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. the set of conjunctive queries over $R_C$.* $\Box$

Although $\mathcal{C}$ in the above proposition may be infinite, we can now identify several special cases where it is finite.

**Proposition 3.** *Let $\mathcal{Q}$ be a query language such that for any fixed schema, $\mathcal{Q}$ can express only a finite number of non-equivalent queries. Then, there is a procedure to compute the composition of $M_{A\to B}$ and $M_{B\to C}$, which is a finite set of GLAV formulas.* $\Box$

The above composition can be obtained by rewriting each of the non-equivalent queries using the procedure defined earlier in this section. An example of where Proposition 3 may apply, is when the size of the queries in $\mathcal{Q}$ is bounded or includes a bounded number of variables. (Note that $CQ_k$ and $linCQ_k$ are substantially more powerful than queries with a bounded number of variables.) Finally, we note that if the semantic mappings are all in LAV, or all in GAV, then the composition will also be finite.

### 3.2 Minimal mapping formulas

We now identify a set of formulas, which we call *minimal formulas*, and show that they are sufficient for producing a composition. Intuitively, these composition formulas are minimal in the sense that we cannot get the same results by using a combination of smaller formulas. That is, there exists a database instance such that these formulas will produce certain answers that cannot be produced by piecing together smaller formulas in the composition.

We illustrate the intuition using the same query $Q(x,y) :- c(x,y_1), c(y_1,y)$ from Figure 2. We claim that the composition *must* include a rule with $Q$ on the right-hand side (see formula 1c).

To see why, we note that to show that $Rew_B(Q)$ is a rewriting of $Query_B(Q)$ in terms of the view $v_b^a$, there must be a *containment mapping* [5] from the unfolding of $Query_B(Q)$ to the unfolding of $Rew_B(Q)$. A variable mapping from a query $Q_1$ to a query $Q_2$ is said to be a containment mapping if it maps each subgoal in $Q_1$ to a subgoal in $Q_2$, and it maps the head of $Q_1$ to the head of $Q_2$. The existence of a containment mapping is a necessary and sufficient condition for showing that $Q_1$ contains $Q_2$. The queries $Query_B(Q)$, $Rew_B(Q)$ and their respective unfoldings are shown in Figure 3. The containment mapping in this example is the obvious one implied by our left-to-right ordering of the subgoals.

Note that the $z_i$ variables are existential in $v_b^c$, and that the $u_i$ variables are existential in $v_b^a$. We refer to the $u_i$ variables as *internally existential* – visible only in the unfolding of $Rew_B(Q)$.

The important thing to note about the containment mapping in the example is that it maps variable $y_1$, which appears in *both* subgoals of $Q$, to variable $u_2$, which is internally existential. The join between the two subgoals in $Q$ could only be enforced in the rewriting because they are part of a single composition formula. This join condition cannot be imposed using the formulas 1a and 1b. Hence, we would not be able to find paths of length 2 in $c$ without this composition formula.

In general, if $Q$ is formed by piecing together two composition formulas, say $Q_1$ and $Q_2$, then the internally existential variables in $Rew_B(Q)$ can only be the image of variables in *one* of them, and are useless for enforcing join conditions between $Q_1$ and $Q_2$. On the contrary, *any formula $Q_a \subseteq Q_c$ such that all join variables in $Q_c$ map to internally existential variables, must be a part of the composition.*

Formally, we can define minimal composition formulas as follows.

**Definition 2.** *Let $R : Q_A(\bar{x}) \subseteq Q_C(\bar{x})$ be a formula in the composition of $M_{A\to B}$ and $M_{B\to C}$. We say that $R$ is a minimal composition formula if no proper subset of the subgoals of $Q_C$ satisfies the following condition.*

*Let $S$ be a subset of the subgoals of $Q_C$, and suppose that the containment mapping from the unfolding of $Query_B(Q_C)$ to the unfolding of $Rew_B(Q_C)$ maps a variable $x$ that appears in $S$ to an internally existential variable in $Rew_B(Q_C)$. Then all atoms in $Q_C$ that mention $x$ are in $S$.*

The following theorem provides the first step in designing our composition algorithm; it shows that we can restrict our attention to minimal mapping formulas.

**Theorem 1.** *Let $\mathcal{C}$ be the set of all minimal GLAV composition formulas of the form $Q_A(\bar{x}) \subseteq Q_C(\bar{x})$, where $Q_C(\bar{x})$ is a conjunctive query over $R_C$, and $Q_A(\bar{x})$ is one of the conjunctive queries in*

$$
\begin{array}{rcll}
\mathit{Query}_B(Q) & = & q_b(x,y) & :-\ v^c_b(x,y_1),\ v^c_b(y_1,y) \\
 & & q_b(x,y) & :-\ \overbrace{b(x,z_1),b(z_1,z_2),b(z_1,y_1)}^{v^c_b(x,y_1)},\ \overbrace{b(y_1,z_3),b(z_3,z_4),b(z_4,y)}^{v^c_b(y_1,y)} \\
\mathit{Rew}_B(Q) & = & q'_b(x,y) & :-\ v^a_b(x,x_1),\ v^a_b(x_1,x_2),\ v^a_b(x_2,y) \\
 & & q'_b(x,y) & :-\ \overbrace{b(x,u_1),b(u_1,x_1)}^{v^a_b(x,x_1)},\ \overbrace{b(x_1,u_2),b(u_2,x_2)}^{v^a_b(x_1,x_2)},\ \overbrace{b(x_2,u_3),b(u_3,y)}^{v^a_b(x_2,y)}
\end{array}
$$

Figure 3: Query unfolding for $\mathit{Query}_B(Q)$ and $\mathit{Rew}_B(Q)$ from Figure 2

*$\mathit{Query}_A(Q_C(\bar{x}))$. Then, $\mathcal{C}$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. the set of conjunctive queries over $R_C$.* $\square$

The theorem is proved by showing that every answer that would be obtained from a non-minimal formula could be obtained by piecing together multiple minimal formulas. Note that there may still be an infinite number of minimal composition formulas.

For simplicity of further exposition, we make two assumptions: (1) all the formulas in $M_{B\to C}$ have a single atom on the right-hand side (i.e., $c$-views are trivial), and (2) every relation name appears on the right-hand side of a single formula in $M_{B\to C}$. See Remark 3 for a brief explanation on removing these assumptions.

The following lemma provides the second observation underlying our algorithm. It shows that minimal composition formulas can be constructed in increasing size.

**Lemma 1.** *Let $Q_A(\bar{x}) \subseteq Q_C(\bar{x})$ be one of the minimal composition formulas in $\mathcal{C}$, where $Q_C$ has $n$ subgoals, $n > 1$. Then there exists another minimal composition formula $Q'_A(\bar{x}) \subseteq Q'_C(\bar{x})$ in $\mathcal{C}$ also satisfying the description of Theorem 1, where $Q'_C$ has $n-1$ subgoals, a subset of the subgoals in $Q_C$, and hence possibly a subset of the head variables of $Q_C$.* $\square$

**Remark 2.** Definition 2 and Theorem 1 can be extended to $n$ levels of composition, for an arbitrary $n$. As a consequence, we can generalize our composition algorithm to arbitrary fixed number of levels of composition. The details are omitted because of space limitations. $\square$

Given Lemma 1, a mapping composition algorithm can begin with composition formulas whose right-hand sides have only a single atom. At every step, the algorithm considers the minimal formulas computed thus far, and tries to *extend* them by adding another atom to their right-hand side. If this process terminates, *i.e.* when no new minimal rules can be obtained by extension, the set of all computed minimal rules (a finite set) will be a composition of the given mappings.

### 3.3 Residues in minimal formulas

The next issue we need to consider is how to deal with a possibly infinite number of composition formulas. We try to encode the infinite formulas in a finite structure. To do so, we identify a condition on pairs of mapping formulas that essentially tells us that the two formulas *can be extended in similar ways.* With that condition, we can partition formulas into equivalence classes and treat all the formulas in an equivalence class identically. We formalize the condition with the notion of *residues*, which we describe below.

To illustrate the notion of a residue, consider how formula 1a in Example 2 could be extended to obtain formula 1c. The intermediate steps in deriving formula 1a are shown in Figure 4.

$$
\begin{array}{ll}
q(x,y_1) :-\ c(x,y_1) & \mathit{Rew}_B(q):\ v_c(x,y_1) \\
\mathit{Query}_B(q): & \overbrace{b(x,z_1),b(z_1,z_2),b(z_1,y_1)}^{v^c_b(x,y_1)}, \\
\mathit{Rew}_B(q): & \overbrace{b(x,u_1),b(u_1,x_1)}^{v^a_b(x,x_1)},\ \overbrace{b(x_1,u_2),\mathbf{b(u_2,x_2)}}^{v^a_b(x_1,x_2)}
\end{array}
$$

Figure 4: Deriving formula 1a in Example 2

The containment mapping from $\mathit{Query}_B(q)$ to $\mathit{Rew}_B(q)$ maps the variable $y_1$ in $\mathit{Query}_B(q)$ to the internally existential variable $u_2$ in $\mathit{Rew}_B(q)$. The last atom in $\mathit{Rew}_B(q)$, $\mathbf{b(u_2,x_2)}$, is not the target of any atom in $\mathit{Query}_B(q)$. Observe that we can extend $\mathit{Rew}_C(q)$ (and the containment mapping) to introduce an atom in $\mathit{Query}_B(q)$ that includes $y_1$, such that $b(u_2,x_2)$ is in the target of the extended containment mapping.

The extended query would include a join condition (using variable $y_1$) that cannot be captured using formulas 1a and 1b (since the join variable $y_1$ maps to the internally existential variable $u_2$). Atom $b(u_2,x_2)$ and the position of variable $u_2$ in that atom characterize the possible extensions of the formula, and constitute the *residue* of the formula. A complete formula is obtained by extending the maximally contained rewriting $\mathit{Rew}_B(q)$ to cover the new atoms introduced in $\mathit{Query}_B(q)$.

Informally, the residue is a quadrapule $\langle \mathit{Atoms}, \bar{E}, \bar{D}, \psi_d\rangle$, where $\mathit{Atoms}$ are the atoms in the unfolding of $\mathit{Rew}_B$ that can be mapped to in an extension of the formula, $\bar{E}$ is the set of internally existential variables that can be used to enforce further join conditions, $\bar{D}$ is the set of variables that can be distinguished in extended formulas (*i.e.* appear in both sides of the mapping formula), and $\psi_d$ is the containment mapping restricted to the variables whose image is in $\bar{E}$ or $\bar{D}$.[1] In our example, the residue for the formula in Figure 4 is $\langle\{b(u_2,x_2)\},\{u_2\},\{x_2\},\{y_1\to u_2\}\rangle$.

Residues in in minimal composition formulas are constructed as follows. Let $r : Q_A(\bar{x}) \subseteq Q_C(\bar{x})$ be

[1]$\psi_d$ is used to link a minimal formula with its extensions.

such a formula; let $\bar{D}_1$, the variables that appear both in $Rew_B(Q_C)$ and its unfolding; and $\psi$, a containment mapping from the unfolding of $Query_B(Q_C)$ to the unfolding of $Rew_B(Q_C)$. Construct a hypergraph $G$ for $Rew_B(Q_C)$ such that there is a node for every variable in its unfolding and an edge $(x_1, \ldots, x_n)$ with label $b_i$ for every atom $b_i(x_1, \ldots, x_n)$ in its unfolding. The residue of $r$ can be constructed as follows: an atom $b_i \in Atoms$ if it lies on a path in $G$ between two variables (nodes), $x$ and $y \in \bar{D}_1$, where $x$ is in the image of $\psi$ and $y$ is not, and the path includes an internally existential variable in the image of $\psi$. $\bar{E}$ is the set of internally existential variables in *Atoms*. $\bar{D}$ is the subset of $\bar{D}_1$ restricted to the variables in *Atoms*. Finally, $\psi_d$ is the restriction of $\psi$ to variables whose image is in $\bar{E} \cup \bar{D}$.

Observe that minimal composition formulas can have *null* residues, *e.g.* the formula in Figure 3. Such minimal rules cannot be extended as they have no available internally existential variables.

Going back to Figure 4, we can clearly see that this formula can be extended by adding the atom $c(y_1, y)$ to $q$ to obtain formula 1c. The atom $b(y_1, z_3)$ in the unfolding of $v_b^c(y_1, y)$ maps to $b(u_2, x_2)$ in the extended containment mapping. In general, it is important to note that any atom $c'$ that extends the $c$-query in a minimal formula must satisfy the following conditions

- $c'$ must include a variable, say $y'$, that is mapped by $\psi$ to a variable in $\bar{E}$, and
- all atoms in the unfolding of $c'$ with $y'$ must be mapped by the extension of $\psi$ to atoms in the residue.

A residue concisely captures *all* information that is required to extend a formula. As a consequence, if two minimal formulas have isomorphic residues, they will also have isomorphic extensions. We exploit this key fact in the next section to encode infinite compositions.

## 3.4 Query Rewrite Graphs

We now describe the construction of a *Query Rewrite Graph* (QRG) that encodes the composition of two sets of mapping formulas. Briefly, a QRG consists two kinds of nodes: *Query* nodes and *Rewrite* nodes. Paths in a QRG contain alternate query nodes ($Q_i$s) and rewrite ($R_i$s) nodes. Every path $p : Q_1R_1 \ldots Q_nR_n$, from a root node $Q_1$, encodes a minimal composition formula $r(p) : Q_A(\bar{x}) \subseteq Q_C(\bar{x})$. Each $Q_i$ contains a single atom from $R_c$ such that the $Q_i$s along $p$ can be chained to obtain the query $Q_C$. Similarly, the $R_i$s can be chained to obtain the query $Q_A$.

The construction of the QRG, as we shall shortly see, mirrors the extension of minimal mapping formulas. For example, the rewrite node $R_n$ (at the end of path $p$) has as children the query nodes that contain possible single atom extensions to the query $Q_C$ of the minimal rule $p(r)$. Further, the QRG is able to encode infinite composition formulas using cyclic paths.

In Figure 5 we show the QRG for the composition of the mappings in Example 3. This finite sized QRG encodes the infinite mapping formulas in that composition.

### Roots of a QRG

The roots of a QRG are query nodes. There is a root node for each single atom query $Q_C(\bar{x}) :- c(\bar{x}, \bar{y})$ such that there exists a non-null minimal composition formula $Q_A(\bar{x}) \subseteq Q_C(\bar{y})$. If $g$ is a query node, then $Atom(g)$ is the single atom of $R_C$, and $Query_B(g)$ is the query $Query_B(Atom(g))$.

The root node $g$ has one child rewrite node for every possible minimal composition formula whose right-hand-side is $Q_C(\bar{x})$. If $r$ is one such rewrite node, then $Rew_B(r)$ and $Query_A(r)$ are the queries $Rew_B(Q_C)$ and $Query_A(Q_C)$ respectively. We denote by $\psi_r$ the containment mapping from the unfolding of $Query_B(g)$ to the unfolding of $Rew_B(r)$.

Thus the root nodes and their child rewrite nodes encode all minimal formulas where $Q_C$ has one atom. In Example 3 and Figure 5, $Q_c^1(x, y) :- c_{gg}(x, y)$ and $Q_c^2(x) :- c_{rgg}(x, y)$ are the only two such queries over $R_C$ for which there exists a non-null minimal formula, and hence are represented in query nodes $Q_1$ and $Q_2$. The two corresponding queries over $R_A$ are in the rewrite nodes $R_1$ and $R_2$. Observe that the node pairs $Q_1R_1$ and $Q_2R_2$ encode the formulas 2a and 2b respectively in the composition.

### Internal nodes

Paths starting from a root of a QRG encode minimal composition formulas. We explain this encoding by induction. Assume, as we have seen for already from paths of length 2, that a path from a root to a rewrite node $r$ encodes a minimal composition formula $form(r) : Q_A(\bar{x}) \subseteq Q_C(\bar{x})$. The rewrite node $r$ has a child query node $g'$ for each possible way of extending $form(r)$ to another minimal composition formula by adding a single atom to $Q_C$. Then, $Atom(g')$ is a $R_C$ atom such that there exists a minimal composition formula of the form $Q'_A(\bar{x}) \subseteq Q'_C(\bar{x})$, where $Q'_C$ is the result of adding $Atom(g')$ to the body of $Q_C$, and $Q'_A$ is an extension of $Q_A$. $\psi'_r(g')$ is the variable mapping from the unfolding of $Query_B(g')$ to the residue in $form(r)$. Note that such extensions may, in addition, also apply a homomorphism to the variables in the residue in $form(r)$.

The node $g'$ has a child rewrite node $r'$ for every possible extended formula $Q'_A(\bar{x}) \subseteq Q'_C(\bar{x})$. For the rewrite node $r'$, $Rew_B(r')$ and $Query_A(r')$ are the sets of atoms that are *added* to $Rew_B(Q_c)$ and $Query_A(Q_c)$ to obtain $Rew_B(Q'_c)$ and $Query_A(Q'_c)$, respectively. We denote by $\psi_r(r')$ the variable mappings that need

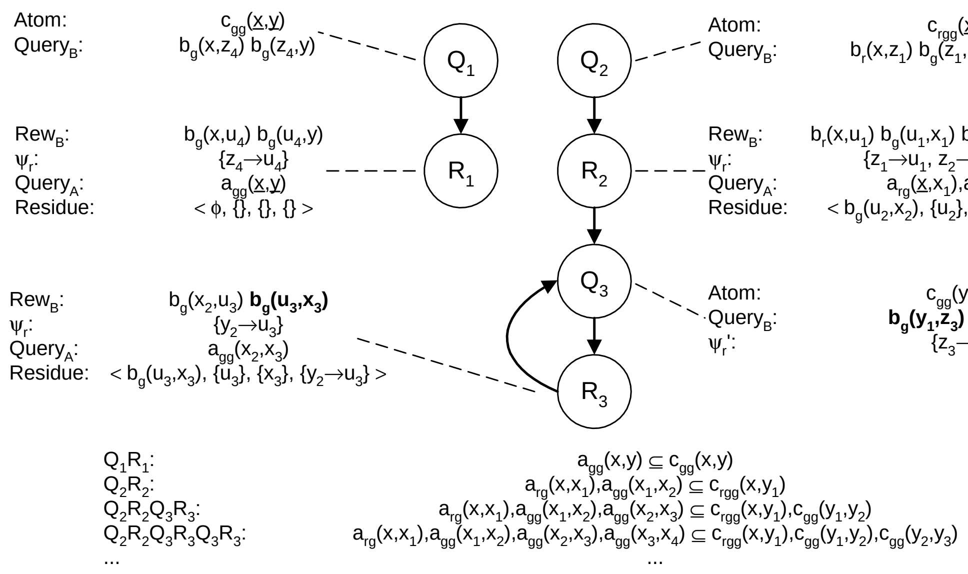
Atom:
Query$_B$:
$c_{gg}(\underline{x},\underline{y})$
$b_g(x,z_4)\ b_g(z_4,y)$
$Q_1$
$Q_2$
Atom:
Query$_B$:
$c_{rgg}(\underline{x},y_1)$
$b_r(x,z_1)\ b_g(z_1,z_2)\ b_g(z_2,y_1)$
Rew$_B$:
$\psi_r$:
Query$_A$:
Residue:
$b_g(x,u_4)\ b_g(u_4,y)$
$\{z_4 \to u_4\}$
$a_{gg}(\underline{x},\underline{y})$
$< \phi, \{\}, \{\}, \{\} >$
$R_1$
$R_2$
Rew$_B$:
$\psi_r$:
Query$_A$:
Residue:
$b_r(x,u_1)\ b_g(u_1,x_1)\ b_g(x_1,u_2)\ \mathbf{b_g(u_2,x_2)}$
$\{z_1 \to u_1,\ z_2 \to x_1,\ y_1 \to u_2\}$
$a_{rg}(\underline{x},x_1),a_{gg}(x_1,x_2)$
$< b_g(u_2,x_2), \{u_2\}, \{x_2\}, \{y_1 \to u_2\} >$
$Q_3$
Atom:
Query$_B$:
$\psi_r'$:
$c_{gg}(y_1,y_2)$
$\mathbf{b_g(y_1,z_3)}\ b_g(z_3,y_2)$
$\{z_3 \to x_2\}$
Rew$_B$:
$\psi_r$:
Query$_A$:
Residue:
$b_g(x_2,u_3)\ \mathbf{b_g(u_3,x_3)}$
$\{y_2 \to u_3\}$
$a_{gg}(x_2,x_3)$
$< b_g(u_3,x_3), \{u_3\}, \{x_3\}, \{y_2 \to u_3\} >$
$R_3$
$Q_1R_1$:
$a_{gg}(x,y) \subseteq c_{gg}(x,y)$
$Q_2R_2$:
$a_{rg}(x,x_1),a_{gg}(x_1,x_2) \subseteq c_{rgg}(x,y_1)$
$Q_2R_2Q_3R_3$:
$a_{rg}(x,x_1),a_{gg}(x_1,x_2),a_{gg}(x_2,x_3) \subseteq c_{rgg}(x,y_1),c_{gg}(y_1,y_2)$
$Q_2R_2Q_3R_3Q_3R_3$:
$a_{rg}(x,x_1),a_{gg}(x_1,x_2),a_{gg}(x_2,x_3),a_{gg}(x_3,x_4) \subseteq c_{rgg}(x,y_1),c_{gg}(y_1,y_2),c_{gg}(y_2,y_3)$
...
...

Figure 5: Query-Rewrite Graph for Example 3. The query-rewrite graph consists of query nodes and rewrite nodes. The right-hand sides of formulas in the composition are encoded by paths of query nodes from the root, and the left-hand sides are encoded by the corresponding rewrite nodes. Below the tree we show how the different composition formulas are encoded by paths in the tree.

to be added to $\psi_r(r)$ to obtain the containment mapping from the unfolding of $Query_B(Q'_c)$ to the unfolding of $Rew_B(Q'_c)$.

In Figure 5, atom $c_{gg}(y_1, y_2)$ in $Q_3$ extends query $c_{rgg}(x, y_1)$ such that $\psi'_r$ maps atom $b_g(y_1, z_3)$ to atom $b_g(u_2, x_2)$ in the residue of $R_2$. Atom $a_{gg}(x_2, x_3)$ completes the rewriting of the extended query. Thus $Q_2R_2Q_3R_3$ encodes formula 2c. Variable $y_1$ is mapped to the internally existential variable $u_2$. Thus this formula cannot be obtained by individual formulas for $c_{rgg}(x, y_1)$, and $c_{gg}(y_1, y_2)$ and is hence a minimal formula.

## Residue labels

As stated earlier, residues enable us to detect when minimal formulas can be extended in similar ways. Hence, with every rewrite node $r$, we attach $residue(r)$, as described in the last section. We say that rewrite nodes $r$ and $r'$ are isomorphic if there is a variable isomorphism $\phi$ such that $\phi(residue(r')) = residue(r)$. When we build the QRG, we do not expand (i.e., create children for) a rewrite node $r'$ if there is already another expanded isomorphic node $r$. Hence, if there is a finite number of possible residue labels, then the QRG will be finite.

Isomorphism between rewrite nodes essentially creates cycles in the QRG. Using these cycles, we can encode an infinite number of composition formulas. Specifically, we can extend $form(r')$ in exactly the same ways we can extend $form(r)$, modulo the isomorphism $\phi$. Note when a child $g'$ of $r$ uses a variable that does not appear in $r$, then the extension of $r'$ should use a fresh variable that appears nowhere else. Hence, we say that a composition formula $R$ is encoded by the QRG if there is some path through the nodes of the QRG (possibly with cycles) that encodes $R$.

In Figure 5, all the other minimal formulas are encoded by cyclic paths of the form $Q_2R_2(Q_3R_3)^*$, and each in turn has a residue isomorphic to $R_2$.

The following theorem shows that the QRG encodes *all* composition formulas. The crux of the proof shows that any formula in the composition can be encoded by a path in the QRG.

**Theorem 2.** *Let $\mathcal{C}$ be the set of composition formulas encoded by the QRG constructed for $M_{A\to B}$ and $M_{B\to C}$. If the QRG is finite, then $\mathcal{C}$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. conjunctive queries.* $\Box$

If the QRG terminates and is acyclic, it encodes a finite set of formulas each of which can be extracted by a top-down traversal of the QRG. If the QRG is finite but cyclic, it encodes an infinite set of formulas. We can use a similar procedure to extract a finite encoding of the composition formulas (we describe the encoding in Section 5). We omit details of this procedure for lack of space.

The QRG construction algorithm and Theorem 2 have the following practical implications:

- When the algorithm terminates and the graph is acyclic (i.e., encodes a finite number of composition formulas), we can extract the composition and apply to it a variety of optimizations. While examples of infinite compositions exist, we still expect that a large number of practical cases will result in finite compositions.

- Any subset $\mathcal{C}_1$ of $\mathcal{C}$ in Theorem 2 is a valid subset of the composition, and therefore can serve as an approximate composition. Approximate compositions will yield only correct answers but possibly only a subset of the certain answers to a query. In certain scenarios of integrating data over a large collection of sources, complete answers are anyway not obtainable. In such scenarios, the efficiency advantages offered by the composition may yield an attractive query processing alternative. Of course, additional knowledge is needed to determine how good an approximation a particular subset $\mathcal{C}_1$ offers. An obvious measure can be given by considering the properties of the particular data sources that are accessed.

In the next section we show that the QRG construction algorithm is guaranteed to terminate for an important class of queries. The algorithm is also of theoretical interest, as it enables us to establish the following complexity bounds on the problem of testing whether a set of formulas is a composition.

**Theorem 3.** *Let $M_{A\to B}$ and $M_{B\to C}$ be two mappings, each consisting of a finite set of GLAV formulas. Let $\mathcal{C}_1$ be a finite set of GLAV formulas relating directly between $R_A$ and $R_C$.*

- *Determining whether $\mathcal{C}_1$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. the set of conjunctive queries is in $\Pi_2^p$.*
- *Determining whether $\mathcal{C}_1$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. a language that has only a finite number of non-equivalent queries for a given schema is $\Pi_2^p$-hard.*

□

The upper bound is established by noting that if the largest view in $\mathcal{C}_1$ has $k$ atoms, then $\mathcal{C}_1$ can be a composition of $M_{A\to B}$ and $M_{B\to C}$ only if all formulas in the composition have $k$ or less atoms on their right-hand side (corollary of Lemma 1). Any single atom extension of a known minimal formula in $\mathcal{C}_1$ must also exist in $\mathcal{C}_1$. The lower bound is obtained by a reduction from the $\forall\exists 3SAT$ problem.

Finally, we address the simplifying assumptions mentioned in the beginning of the section.

**Remark 3.** The algorithm for mappings that don't satisfy our simplifying assumptions is conceptually similar, though more involved. If $M_{B\to C}$ has more than one formula per relation name, we multiply the number of query nodes we have: one per combination of atom and formula. If the formulas of $M_{B\to C}$ have non-trivial views on their right-hand sides, then instead of adding a single atom to a formula as we go through paths of the QRG, we add a block of atoms (corresponding to all the subgoals in the definition of a c-view) every time. □

## 4 Composition w.r.t. $linCQ_k$ and $CQ_k$

In this section we show that our composition algorithm (with a slight tweak) will terminate if we consider composition w.r.t. the query languages $linCQ_k$ and $CQ_k$. Recall that composition w.r.t. a query language $\mathcal{Q}$ means that the composed mapping generates all certain answers for any given query in $\mathcal{Q}$. (The composition is still correct for queries outside $\mathcal{Q}$, but may not be complete). Hence, for these two classes of frequently occurring queries, we can obtain the advantages offered by pre-composition, and from a theoretical perspective, termination shows that composition is decidable for these classes of queries.

We begin by explaining the algorithm for $linCQ_k$, and then sketch the extension for $CQ_k$. To create a composition w.r.t. $linCQ_k$, we construct our QRG with the following slight modification: we only consider residues that have at most $k^2$ internally existential variables (if a node has a bigger residue, then we instead create multiple nodes each with a different subset of $k^2$ internally existential variables). A simple counting exercise shows that this restriction guarantees that there are a finite number of residues.

The reason we can restrict residues in this way is the following. Because of the structure of $linCQ_k$ queries, there is always an ordering of the subgoals in $Q_C$ such that all variable interactions are local to small sets of $k$ variables. Further, we can always reorder the atoms in $Q_c$ such that the algorithm is guaranteed to construct the formula in that order, and that at most $k^2$ variables are required to capture all possible interactions between variables. Hence, as we go down the tree, we only need to keep track of $k^2$ variables. This claim is the key behind the following theorem.

**Theorem 4.** *Let $\mathcal{C}$ be the set of composition formulas encoded by the QRG if we only consider nodes whose residues have at most $k^2$ internally-existential variables. Then $\mathcal{C}$ is a composition of $M_{A\to B}$ and $M_{B\to C}$ w.r.t. the class of $linCQ_k$ queries.* □

Queries in $CQ_k$ also have a similar locality property, but not in a linear ordering of the subgoals. Instead, if we represent the atoms of a $CQ_k$ query as a tree, then we can show that along every path from the root to a leaf, their variable interactions are limited to $k$ variables. To compute the composition w.r.t. the class $CQ_k$, we need a slightly more involved algorithm. Instead of encoding composition formulas by paths through the graph, formulas are encoded by *prefix subtrees* of the QRG. As in the case of $linCQ_k$, we can reorder the atoms of $Q_C$ along paths of the tree so that we are guaranteed that the algorithm will create the prefix subtree corresponding to any minimal formula. The following theorem summarizes the result.

| $P_C$ | $P_A$ |
|---|---|
| $q(x) :- c_{rgg}(x, y), p(y)$ | $q(x) :- a_{rg}(x, y_1), a_{gg}(y_1, y_2), p(y_2)$ |
| $p(x) :- c_{gg}(x, y), p(y)$ | $p(x) :- a_{gg}(x, y), p(y)$ |
| $p(x) :- c_{gg}(x, y)$ | $p(x) :- a_{gg}(x, y)$ |

Figure 6: Encoding the infinite composition formulas in Example 3 using recursive datalog programs.

**Theorem 5.** *Let $M_{A \to B}$ and $M_{B \to C}$ be GLAV mappings, each consisting of a finite number of formulas. There is a procedure to compute a composition of $M_{A \to B}$ and $M_{B \to C}$ w.r.t. the class of $CQ_k$ queries.* □

In summary, this section has shown a new way to use a restriction on the expected queries for the purpose of optimization. If we know that our queries (or a large number of them) will be in $CQ_k$, then we can precompute the compositions. Hence, in practice, benefits of composition can be achieved in many cases.

## 5 Using Infinite Compositions

Up to now, we have presented an algorithm for composition that will generate a finite composition for many common cases. To complete the picture, we examine the case where the algorithm terminates, but encodes an *infinite* set of composition formulas. The utility of our finite encoding is dependent on the ability to use the encoded formulas to obtain the certain answers to queries. However note that traditional query answering algorithms [11] cannot utilize such an encoding. This section shows that in this case too, we can use the composition provided by the algorithm to provide complete answers to a query. The algorithm for doing so is a result of independent interest.

**Encoding an infinite composition:** When our algorithm returns an infinite composition, it can be encoded as follows with a datalog program. Recall that a recursive datalog program $P$ defines a query predicate $p$ in terms of a set of extensional predicates (and possible additional intensional predicates). The program $P$ can be viewed as encoding an infinite number of conjunctive queries, which are the finite unfoldings of $p$ in terms of the extensional predicates. Hence, $p$ is defined to be the union of its unfoldings.

A composition mapping $M_{A \to C}$ is given by a datalog program whose finite unfoldings encode the right-hand sides of $M_{A \to C}$, and a function that specifies the left-hand side for any given right-hand side. Formally, a mapping $M_{A \to C}$ is encoded by a pair ($P_C$, $M_{CA}$), where $P_C$ is a datalog program over $R_C$, and $M_{CA}$ is a function that for every finite unfolding, $Q_C$, of $P_C$ returns all the conjunctive queries, $Q_A$, such $Q_A \subseteq Q_C \in M_{A \to C}$. The important aspect of $M_{CA}$ is that it is defined on the rules of $P_C$. Hence, we can define a datalog program $P_A$ such that the unfoldings of $P_A$ correspond to the right unfoldings of $P_C$. Figure 6 shows such a representation for the infinite formulas in the QRG in Figure 5.

**Computing certain answers**: Given the above encoding, the crux of using the reformulation boils down to the following problem. Given a query $Q$, we need to reformulate $Q$ using an infinite number of views that are encoded by a datalog program (the right-hand sides of the composition). A similar problem was solved in [19], where infinite sets of views represented data sources that may have query answering capabilities, and the views represented all the possible queries that a source can answer. However, [19] only considered the rewriting problem when *equivalent* rewritings are considered, rather than maximally-contained rewritings.

In the full version of the paper we prove the following generalization of [19] to maximally-contained rewritings:

**Theorem 6.** *Given an infinite set of views $\mathcal{V}$ encoded by the expansions of a datalog program $P$, and a query $Q$ over the EDB predicates of $P$, we can compute a maximally-contained rewriting, $Q'$, of $Q$ over $\mathcal{V}$, and $Q'$ yields all the certain answers to $Q$ for any instance of the EDB predicates of $P$.* □

Given the reformulation provided by Theorem 6 for a query $Q$, we can apply to it the mapping $M_{CA}$, thus obtaining a reformulation of $Q$ in terms of $R_A$. This reformulation is guaranteed to yield all the certain answers to $Q$. Furthermore, to obtain additional savings at run-time, the datalog program encoding the composition can be optimized in advance, using techniques such as [13] for pushing selections and removing redundant rules.

## 6 Conclusion

This paper presented the first treatment of composition of semantic mappings. The motivations for mapping composition are both fundamental, as mappings become objects of significant interest [3], and as they are used in large-scale data sharing systems. From a theoretical perspective, we showed that (1) the composition of GLAV mappings may be infinite, (2) for queries in $CQ_k$, the composition can be precisely computed, and (3) bounds can be established on the complexity of composition. From a practical perspective, we have shown that in many common cases, compositions of mappings can be computed in advance. These compositions can be pre-optimized to remove redundant rules and joins, push selections, and determine join orders. In contrast to previous work that needs to *chain* semantic mappings at run-time [12], the optimized compositions can reduce reformulation time, prevent the following of redundant paths in the network, and produce better query optimization plans.

This paper provides the basis on which to design optimization methods for query processing over networks of semantically related data. The key challenge we are addressing now is to decide *which* paths to pre-compose and ensure that the optimizer uses the composition appropriately. As pertaining to the composition itself, we would like to extend our algorithm to compose mappings that are themselves finite encodings of infinite GLAV formulas, and investigate efficient algorithms for performing composition.

## Acknowledgements

We would like to thank Surajit Chaudhuri, Zachary Ives, Todd Millstein, Tova Milo, Rachel Pottinger, and Ashish Sabharwal for discussions and comments relating to earlier drafts of this paper. This work was funded in part by NSF ITR grants IIS-0205635 and IIS-9985114 and a gift from Microsoft Research.

## References

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

[2] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proceedings of the WebDB Workshop*, 2002.

[3] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.

[4] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Vardi. View-based query processing for regular path queries with inverse. In *In Proceedings of PODS*, pages 58–66, 2000.

[5] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[6] C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. In *Proceedings of the International Conference on Database Theory (ICDT)*, 1997.

[7] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, pages 109–116, Tucson, Arizona., 1997.

[8] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of the International Conference on Database Theory (ICDT)*, 2003.

[9] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of AAAI*, 1999.

[10] A. Halevy, Z. Ives, I. Tatarinov, and P. Mork. Piazza: Data management infrastructure for semantic web applications. In *Proc. of the Int. WWW Conf.*, 2003.

[11] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[12] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.

[13] A. Y. Halevy, I. Mumick, Y. Sagiv, and O. Shmueli. Static analysis in datalog extensions. *Journal of the ACM*, 48(5):971–1012, September 2001.

[14] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of PODS*, pages 51–61, Tucson, Arizona, 1997.

[15] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, 1989.

[16] P. Kalnis, W. Ng, B. Ooi, D. Papadias, and K. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proc. of SIGMOD*, 2002.

[17] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proc. of SIGMOD*, 2003.

[18] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. of PODS*, 2002.

[19] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proc. of PODS*, pages 227–237, Montreal, Canada, 1996.

[20] C. Li, M. Bawa, and J. Ullman. Minimizing View Sets without Losing Query-Answering Power. In *Proceedings of the International Conference on Database Theory (ICDT)*, 2001.

[21] J. Madhavan, P. Bernstein, P. Domingos, and A. Y. Halevy. Representing and Reasoning about Mappings between Multiple Domain Models. In *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI)*, 2002.

[22] J. Madhavan and A. Halevy. Composing mappings among data sources. www.cs.washington.edu/homes/jayant/full-composition.pdf, 2003.

[23] S. Melnik, E. Rahm, and P. Bernstein. Rondo: A programming platform for generic model management. In *Proc. of SIGMOD*, 2003.

[24] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1998.

[25] W. S. Ng, B. C. Ooi, K.-L. Tan, and A. Zhou. Peerdb: A p2p-based system for distributed data sharing. In *ICDE*, Bangalore, India, 2003.

[26] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[27] A. P. Sheth and J. A. Larson. Federated database systems for managing, distributed, heterogenous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[28] J. D. Ullman. Information integration using logical views. In *Proc. of ICDT*, pages 19–40, Delphi, Greece, 1997.

[29] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proc. of PODS*, pages 266–276, 1995.

[30] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, 1992.