Mechanism and Algorithm for Indirect Schema Mapping Composition

Bo Wang
College of Information System and Management, National University of Defense Technology, Changsha, China
wbsteven@163.com

Bo Guo
College of Information System and Management, National University of Defense Technology, Changsha, China

Abstract—There are a large number of indirect schema mappings between peers in the network. To improve the efficiency of data exchange and queries, indirect mappings are needed to be composed. Direct mappings can be derived directly by the constraints defined between schemas, but not for indirect mappings’ composition. Defined the combination operations of schema elements in indirect mappings, and gave the expression of indirect mappings. Analyzed the composition of indirect mappings, and proposed a strategy, named schema element back, to solve the problem of indirect mapping composition, and gave the indirect mapping composition generation algorithm based on such strategy. Experiments showed that indirect mapping composition can improve the efficiency of data exchange, and compared with other non-full mapping composition generation algorithms, and indirect mapping composition generated by our algorithm based on schema element back strategy can completely eliminate the infection of media schema with no reduction of the composition efficiency.

Index Terms—indirect mapping composition; combination operation; schema element back

I. INTRODUCTION

Many data management tasks, such as data translation, information integration, and database design require manipulation of database schemas and mappings between them[1]. Schema mappings define the relationship between instances of two given schemas which can be divided into direct and indirect ones by the relationships of elements. Indirect mappings are widely existed, especially when schemas evolve, most mappings that used to be direct become indirect. In indirect mappings, elements between schemas are not directly associated, but related by some algebra operations.

Mapping composition refers to combining two mappings into a single one, which is useful for a variety of data management problems. Figure 1 shows a topology of a data sharing network, each node can be a data sources, an integrated data center or a logical mediator. When data are exchanged from node E to the integrated schema G, instances of E should be translated to G by mapping sequences: E —— C, C —— A and A —— G, and chaining mappings at run-time may be expensive because we may need to follow long and possibly redundant paths in the network. Note that different paths between a pair of nodes may yield different sets of query answers. To ensure the reliability of data exchange, we usually need to get all of the possible paths, and execute each possible data transplantation process according to the paths. Cost of performing such work will be quite large when there are a large amount of data sources. If certain nodes leave the network, then we may lose the mapping paths. Addressing these issues raises several static analysis questions regarding the network of mappings, and mapping composition lies at the core of them all. By pre-composing a select set of mapping chains in the network, we can directly execute the data exchange between the source schema and the target schema, which leads to significant run-time savings. Moreover, mapping composition arises in many practical settings like data integration, schema evolution and database designing:

In data integration, a query needs to be composed with a view definition. If the view definition is expressed using global-as-view (GAV), then this is an example of composing two functional mappings: a view definition that maps a database to a view, and a query that maps a view to a query result.

In schema evolution, a schema evolves to a new one, and the relationships between the two schemas may be described by a mapping. The original mappings on the old schema can be updated by mapping composition.

In a database design process, schema may evolve frequently via a sequence of incremental modifications. This produces a sequence of mappings between successive versions of the schema until the desired schema is reached, so a mapping from the original schema to the final one is needed, which can be obtained by composing the mappings between the successive versions of the schema. With this mapping, the designer can migrate data from the old schema to the new schema.
Another motivation for mapping composition comes from the framework of model management\cite{2}. One of the basic operators in model-management algebra is composition, and mappings are treated mostly as syntactic objects. Mapping management can be achieved by defining the composition and inverse operations.

Query composition is widely supported by most of the commercial data management tools, while the mapping composition is still rest on researches and experiments. If the mappings are functional, we can do the composition similarly as the query composition. However, there are a large number of non-functional mappings, most of which are indirect mappings, and we can not directly give the composition. Different from the query composition, indirect mappings can not be composed directly.

In this paper, we mainly discuss the composition problems of indirect mappings, and the rest of the paper is organized as follows. In section 2, we show the related works on mapping composition. In section 3 we introduce some concepts of mapping composition. In section 4 and 5, we firstly define the indirect mappings composition, and then propose an indirect mapping composition algorithm. Section 4 presents the experimental results. And the last part is the conclusion.

II. RELATED WORKS

Mapping composition is a challenging problem\cite{1} whose difficulty is quite sensitive to the expressiveness of the allowed mappings, and there are several researches on such problem\cite{4,5,6,7,8}, including methods based on tuple generation constraints and schema transformation\cite{9,10,6}, etc. Those mapping composition methods based on schema transformation define complicated schema transformation operations, which only support the computing of direct mappings, and not for those indirect ones with complex algebra operations between schema elements. Model management is a generic approach to solving problems of data programmability where precisely engineered mappings are required\cite{2}. A model management system supports the creation, compilation, reuse, evolution, and execution of mappings between schemas represented in a wide range of meta-models, where the composition operation is one of the ways to realize mapping reusing.

Researches on indirect mappings mostly consider the definitions and basic operations of non-direct element-to-element mappings between the source and target schemas\cite{11} without any further discussions about composition. Madhavan and Haleby\cite{3} showed that the composition of two given mappings expressed as GLAV formulas may not be expressible in a finite set of first-order constraints. Fagin et al.\cite{4} proved that the composition of certain kinds of first-order mappings may not be expressible in any first-order mappings, even by an infinite set of constraints, because the mapping language is not closed under composition. Nash et al.\cite{8} showed that for certain classes of first-order languages, it is undecidable to determine whether there is a finite set of constraints in the same language that represents the composition of two given mappings. In \cite{4}, Fagin et al. demonstrated the second-order mapping language, namely second order source- to-target tuple-generation dependencies (denoted by SOtgd), is closed under composition, and they also present a composition algorithm for this language. The second-order languages uses existentially quantified function symbols, which essentially can be thought of as Skolem function. A tuple-generating dependency specifies an inclusion of two conjunctive queries, $Q_1 \subseteq Q_2$. It is called source-to-target when $Q_1$ refers only to symbols from the source schema and $Q_2$ refers only to symbols from the target schema. However, for implementation, such kind of languages is not supported by standard SQL-based database tools.

Yu and Popa extended the algorithm of Fagin’s to handle nesting and apply it to some schema evolution scenarios, and they also discussed the optimizations of the composition result. Nash et al.\cite{8} studied the composition of first-order constraints that are not necessarily source-to-target. They considered dependencies that can express key constraints and inclusions of conjunctive queries $Q_1 \subseteq Q_2$, where $Q_1$ and $Q_2$ may reference symbols from both the source and target schema, but the composition of constraints in this language is not closed and whether a composition result exists is undecidable. They also gave an algorithm that produces a composition.

Berstein et al.\cite{3}, explored the mapping composition problem for constraints that are not restricted to being source-to-target, which extended the work of Nash and Fagin. They applied a “left-compose” step which allows the algorithm to handle mappings on which the algorithm in \cite{8} fails. They used algebra-based instead of logic-based language to express mappings, which can be directly supported by database tools. We call their method “best effort composition approach” (denoted by BECA). BECA tries to eliminate as many relation symbols from the middle schema as possible. Given schemas $\sigma_1$, $\sigma_2$, and $\sigma_3$, mappings computed by BECA have the form of $\sigma_1 \rightarrow \sigma_2' \rightarrow \sigma_3$, where $\sigma_2' \subseteq \sigma_2$. That means, in some cases it may be better to eliminate some symbols from $\sigma_2$ successfully, rather than insist on either
eliminating all them or failing. Since there may be some elements in the middle schema remain at the end of BECA, the result is not a perfect composition.

A main reason that $\sigma_2'$ is not empty after BECA, is that there may be indirect mappings. In this paper, we use a strategy namely schema elements back to solve the problem. Although this method expands $\sigma_1 : \sigma_1' \rightarrow \sigma_1'$, it can completely eliminate $\sigma_2$, and a composed mapping only depends on the source and target schema will be generated.

III. MAPPING COMPOSITION

Definition 1 Let $R_i$ and $R_j$ be two binary relations, the composition $R_i \circ R_j$ of $R_i$ and $R_j$ is the binary relation: $R_i \circ R_j = \{(x, z) : \exists y \in R_i(z, y) \in R_j \}$.

Let $M = (S, T, \Sigma)$ be a schema mapping model, where $S$ and $T$ are the schemas with no relation symbols in common and $\Sigma$ is a set of formulas of some logical formulas over $<S, T>$. Then let $Inst(M)$ be a binary relation between instances over $S$ and $T$. We define composition of two schema mappings $M_i$ and $M_j$ using the composition of the binary relations $Inst(M_i)$ and $Inst(M_j)$.

Definition 2[4] Let $M_{12} = (S_i, S_2, \Sigma_{12})$ and $M_{32} = (S_2, S_3, \Sigma_{32})$ be two schema mappings such that the schemas $S_i, S_2, S_3$ have no relation symbol in common pairwise. A schema mapping $M = (S, S, \Sigma_{12})$ is a composition of $M_{12}$ and $M_{32}$ if $Inst(M) = Inst(M_{12}) \circ Inst(M_{32})$ which means that $Inst(M) = \{(i_1, i_2) : (i_i, i_2) \in Inst(M_{12}) \land (i_1, i_2) \in Inst(M_{32})\}$, where $i_1$ is the instance of $S_i$ with $1 \leq i \leq 3$.

Figure 2 shows an example of direct mapping composition, where $S_1$ is the middle schema. Data exchange between $S_1$ and $S_2$ is achieved by composing $M_{12}$ and $M_{32}$. Fagin et al. have proved the existence inevitability of mapping composition, and Alan Nash[1] gives the prerequisite for the existence of mapping composition, that is two mappings have a common schema, which is consistent with Fagin’s result. Both of them proved the existence of mapping composition by the relationship between query and mappings, and the instance isomorphism.

Example 1: Consider the following schemas $S_1, S_2$, and $S_3$. $S_1$ consists of a single binary relation symbol $Maint$, which associates the name of maintenance person with the equipment he maintains. Schema $S_2$ consists of a similar binary relation symbol $Maint'$, which is a copy of $Maint$, and of an additional binary relation symbol $MP$, that associates each person with an id. Schema $S_3$ consists of one binary relation symbol $Reg$, that associates person ids with the equipments the persons take. Consider the following schema mappings $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{32} = (S_2, S_3, \Sigma_{32})$, where $\Sigma_{12} = \forall \forall \forall (Maint(n, c) \rightarrow Maint(n, c)), \forall (\forall \forall (MP(n, c) \rightarrow \exists \exists Reg(s, c)))$.

Given the instances of $S_1, S_2, S_3$ as follows:

$I_1 : Maint = \{(A, Eq1), (A, Eq2)\}$
$I_2 : Maint = \{(A, 0001), (B, 0002)\}$
$I_3 : Reg = \{(0001, Eq1), (0001, Eq2)\}$

According to $\Sigma_{12}$ and $\Sigma_{32}$, we have $\langle I_1, I_2 \rangle \in Inst(M_{12})$ and $\langle I_1, I_3 \rangle \in Inst(M_{32})$. By definition 2, if there is a mapping $M_{12}$, that satisfies $\langle I_1, I_2 \rangle \in Inst(M_{12})$, then $M_{12}$ is the composition of $M_{12}$ and $M_{32}$, so $M_{12}$ can be expressed as $\forall \forall \forall (Maint(n, c) \rightarrow \exists \exists Reg(s, c))$.

Theory 1 The mapping composing operation $Inst(M_{12}) \circ Inst(M_{32}) = Inst(M_{12})$ is closed under the homomorphism of instances.

Proof: Given $\langle I_1, I_2 \rangle \in Inst(M_{12})$, for the instances $I_1'$ and $I_2'$, satisfying $I_1' \cong I_1$ and $I_2' \cong I_2$ ("\cong" denotes homomorphism), we just need to show that $\langle I_1', I_2' \rangle \in Inst(M_{12})$. Since $Inst(M_{12})$ and $Inst(M_{32})$ are closed under homomorphism, and $Inst(M_{12}) \circ Inst(M_{32}) = Inst(M_{12})$, there exists an instance $I_2'$ that $\langle I_1', I_2' \rangle \in Inst(M_{12}) \land \langle I_1, I_2 \rangle \in Inst(M_{32})$. So there is an instance $I_2' \cong I_2$ such that $\langle I_1', I_2' \rangle \in Inst(M_{12})$, and also $\langle I_1, I_2 \rangle \in Inst(M_{32})$, then $\langle I_1', I_2' \rangle \in Inst(M_{12})$. This was to be shown.

IV. INDIRECT MAPPING

A direct mappings can be given by a Source-to-Target tuple generating dependency (denoted by S2Tgd): $\forall x(\phi_S(x) \rightarrow \exists y \psi_T(x, y))$, where $\phi_S$ and $\psi_T$ are respectively the formulas on $S$ and $T$, and $x, y$ are the set of variables. A full S2Tgd mapping can be expressed by $\forall x(\phi_S(x) \rightarrow \psi_T(x))$, and a mapping expressed by $\forall x(\phi_S(x) \rightarrow \phi_R(x))$ equals to the set of some full S2Tgds $\forall x(\phi_S(x) \rightarrow \phi_R(x))$, where $i = 1, \ldots, k$, and $\phi_R(x)$ is the atomic formula.
A. Describe indirect mappings

In indirect mappings, the algebra operation results of the elements in both the source and target schema are related, as shown in Figure 3.

![Indirect mappings](image)

Figure 3. Indirect mappings

In element correspondence "[1]" (see Fig. 3), nodes in the pane of the source schema are mapped to an element in the target schema after some algebra operations. The algebra operations of schema elements includes some basic ones, like Union, Cartesian, Projection and Selection, and some compositions, like Intersection, Join, Nature Join and division etc.

Definition 3 Indirect elements correspondence (denoted by ICOE) is the set of element correspondences that elements are mapped after some algebra operations.

Let $\phi_3$ be the formulas of $S$, and $S'$ be the scheme of $S'$, the algebra operations have six forms, as shown below, where 1–4 are the basic ones and 5–6 can be derived from 1–4:

- **Union** Let $\phi_3(x) = \bigwedge_{i, j} (S_i(x) \wedge \cdots \wedge S_j(x))$, $S_i(x) \in S$, and it is the same for $\psi_7$.

  ICOE: $\forall e(S_i(e) \lor S_j(e) \lor \cdots \lor S_k(e) \rightarrow T(e))$ (or $\forall e(S_i(e) \lor S_j(e) \lor \cdots \lor S_k(e) \rightarrow \exists w(T(e), w))$ if there is an existence quantifier).

- **Cartesian** $\phi_3(x) = \bigwedge (S_i(x) \times \cdots \times S_j(x))$, $S_i(x) \in S$

  ICOE: $\forall e(S_i(e) \times S_j(e) \times \cdots \times S_k(e) \rightarrow T(e))$ (or $\forall e(S_i(e) \times S_j(e) \rightarrow \exists w(T(e), w))$).

- **Projection** $\phi_3(x) = \bigwedge (\pi_{k, l, m}(S_i(x)))$, $S_i(x) \in S$

  ICOE: $\forall x(\pi_{a, b}(S_i(a, b, c, d)) \wedge \pi_{c, d}(S_i(b, c, d, e)) 

  \rightarrow T(a, b, c, d, e))$ (or $\forall x(\pi_{a, b}(S_i(a, b, c, d)) \wedge \pi_{c, d}(S_i(b, c, d, e)) 

  \rightarrow \exists y(T(a, b, c, d, e)))$, where $x$ and $y$ are the set of variables.

1. **Selection** This operation considers the conditional mappings, which represents the user’s integrality. It has the form of

   $\phi_3(x) = \bigwedge (\pi_{k, l, m}(S_i(x))) = \bigwedge (S_i(x) \land F(t)), S_i(x) \in S,t \in Inst(S)$.

   ICOE: $\forall e((S(e) \land (Inst(e) \Rightarrow \theta)) \rightarrow T(e))$.

2. **Intersection** refers to several schemas, which is a kind of selection operation essentially.

3. **Join** is the composition of Cartesian and Selection

   Let $\Theta = \{\land, \lor, \sigma, \theta\}$ be the basic algebra operation symbol set (see 1–4 in definition 3), and indirect mappings can be expressed as $\phi_3'(x) = \bigotimes_{i} (\pi_{k, l, m}(S_i(x)))$, $S_i(x) \in S$. Take the indirect mapping shown in Figure 4 as an example, $M_{12}$ can be expressed as:

   $\forall x((\pi_{a, b}(R_{1}^{S}(x))) \land \pi_{c, d}(R_{2}^{S}(x))) \land \pi_{a, b}(R_{3}^{S}(x)))$ $\rightarrow \exists y(R_{1}^{S}(y) \lor R_{2}^{S}(y) \lor R_{3}^{S}(y))$, and $M_{23}$:

   $\forall y(R_{1}^{S}(y) \land \pi_{c, d}(R_{2}^{S}(y)) \rightarrow \exists z(R_{1}^{S}(z) \land R_{2}^{S}(z)))$.

   If $\psi_i(x, y)$ has the same form with $\phi'_3(x)$, then the non-full indirect mapping can be expressed as:

   $\forall x(\phi'_3(x) \rightarrow \exists y\psi'_i(x, y))$.

![Indirect mappings between schemas](image)

Figure 4. Indirect mappings between schemas.

B. Exchange data under indirect mappings

Now we give the concept of homomorphic instances under indirect mappings (concept of homomorphic direct mappings is given in [4], and we do not discuss it here). Consider the following example:

**Example 2:** Given schemas $S_1$, $S_2$, and $S_3$. $S_1$ consists of a single binary relation symbol $Consistof$, which associates the name of equipments with the parts they have. $S_2$ consists of a similar binary relation symbol $Consistof$, which is a copy of $Consistof$, and of an additional binary relation symbol $MCode$, that associates each equipment name with an id. $S_3$ consists of a single binary relation symbol $MRegister$, which associates the code of equipments with the related parts. Consider the following mappings: $M_{12} = (S_1, S_2, \Sigma_{12})$ and $M_{23} = (S_2, S_3, \Sigma_{23})$, where

   $\Sigma_{12} = \{\forall e \forall p(Consistof(e, p) \rightarrow Consistof(e, p)), \forall e \forall p(Consistof(e, p) \rightarrow \exists c MCode(e, c))\}$,

   $\Sigma_{23} = \{\forall e \forall p(MCode(e, c) \land Consistof(e, p) \rightarrow MRegister(e, p))\}$.

   The second formula in $\Sigma_{12}$ is a direct mapping that associate equipment name $e$ in the source schema to the equipment name $e$ in the target schema. According to Theory 1, let $I, J$ be the instance of a mapping $M$, that means $\langle I, J \rangle \in Inst(M)$, and if there is an instance pair $\langle I', J' \rangle$ that $\langle I', J' \rangle \cong \langle I, J \rangle$, then $\langle I', J' \rangle \in Inst(M)$.

   For the converse, if $\langle I', J' \rangle$ and $\langle I, J \rangle$ are respectively the instance of $M$, then we have $\langle I', J' \rangle \cong \langle I, J \rangle$. The above conclusion is true for that $M$ is the direct mapping. For the indirect mappings, the homomorphism between instances is different. For example, let $S_1'$ be a schema,
which is a copy of $S_i$ with a little differences that certain element in $S_i$ splits into several parts in $S_i'$. For example, the equipment name in $S_i'$ contains two parts (supername+name), and the corresponding formula set is $\Sigma_i'$. Therefore, under the condition of homomorphism, there may be some non-one-to-one correspondences.

Let $S_i$, $T$ be the source and target schema respectively, and $R_i$ is a scheme of $S_i$. If there is an indirect mapping $M$ from $S_i$ to $T$, and let $R_i'$ be the scheme constructed as follows: substituting each element combination that participates in the indirect mappings with a single element. We call $R_i'$ a generating scheme. If we define mappings from $R_i'$ to $T$ with the form of $\forall x(R_i'(x) \rightarrow \psi_f(x))$, and mappings from $R_i$ to $T$ with the form of $\forall x(R_i(x) \rightarrow \psi_f(x))$, then the mappings from $R_i(x)$ to $\psi_f(x)$ are direct, and those from $R_i'(x)$ to $\psi_f(x)$ are indirect, as shown in figure 5:

![Instance homomorphism under indirect mappings](image)

Figure 5. Instance homomorphism under indirect mappings.

We define the homomorphism function $h$ under indirect mappings as:

$$h(a_i) = \begin{cases} 
&e, & e \in \text{Inst}(\pi_k(R_i)) \\
&\text{compound}(e_1, \ldots, e_n), & e_i \in \text{Inst}(\pi_i(R_i))
\end{cases}$$

Then $h$ is called a general homomorphism function, where compound is the algebra operation of elements.

**Definition 4** Let $R$ and $R'$ be two schemas, $h$ is the general homomorphism function shown above. Let $J_i, J_i'$ be the instances of $R$ and $R'$ respectively. If for each scheme symbol $R_i$ in $R$, and each tuple $(a_1, \ldots, a_n) \in R'$, there is a tuple $(h(a_1), \ldots, h(a_n)) \in R'$, then $R$ and $R'$ are homomorphic instances under $h$, then the data exchange process under indirect mappings can be implemented by the homomorphism given in Definition 4 comparing with the process under direct mappings (see[12]).

### V. MECHANIC OF INDIRECT MAPPING COMPOSITION

#### A. Schema-elements-back

As shown in figure 4, although there is a common schema $S_i$, we can not directly construct the mappings from $S_i$ to $S_{i'}$, according to $M_{i_1}$ and $M_{i_2}$, even can not give the mapping composition directly. Using direct mapping composition, elements in $S_i$ can only map to part of the certain elements of $S_{i'}$. During the data exchange procedure, a complete instance of $S_i$ may consist part of the instance from $S_{i'}$, so data exchange from $S_i$ to $S_{i'}$ by direct mapping composition $M_{i_2} \circ M_{i_3}$ may lose some information, that is because $M_{i_1}$ and $M_{i_3}$ contains some elements that is not in common, there is no such an instance $I_i$ of $S_i$ satisfying conditions in Definition 2.

Since the expressions of indirect mapping composition can not be derived by Definition 2 directly, we deal with this problem based on general instance homomorphism, the main idea of which is to construct a schema that can be directly used for mapping composition, and make sure that the instances of the constructed schema is homomorphic with the instances of the original schema under the general instance homomorphism function. We propose a strategy named schema-elements-back by constructing some virtual elements in the source schema to deal with the composition of indirect mappings. Compared with BECA, schema-elements-back can eliminate the infection of the middle schema.

**Definition 5** Let $M_{S_T} = (S, T, \Sigma_{S_T})$ be a model of indirect schema mapping, where $\Sigma_{S_T}$ is the formula set with the form of $\forall x(\phi(x) \rightarrow \exists \psi_f(x,y))$, and mappings from $S$ to $T$ are sound[13]. Let Dom$_{\psi_f}(S)$ be the domain of $S$, and Dom(T) be the domain of $T$, and then we have Dom$_{\psi_f}(S) \subseteq$ Dom(T). Let $y_{\psi_f}$ and $x_{\psi_f}$ be the set of variables that participate in the indirect mappings in $T$ and $S$ respectively, and we construct the virtual elements in $S$ by set $y_{\psi_f} - x_{\psi_f}$. As shown in figure 6, the newly constructed source schema is denoted by $S'$. We can construct direct mappings from $T$ to $S'$, and then the process of virtual elements construction is called a schema-elements-back process.

![Inverse of indirect mappings based on virtual elements](image)

Figure 6. Inverse of indirect mappings based on virtual elements.
B. Indirect mapping composition algorithm

When data are exchanged under the strategy of \textit{schema-elements-back}, instance of some elements in the middle schema may be firstly transformed to the source, and there is a reverse data flow during the data exchanging process from the source to the target. Such a data flow can be implemented by the reverse mapping from the element in the middle schema to the newly added element in the source schema. Since the object of composing two schema mappings is to directly construct the mapping from the source to the target without considering the middle schema, \textit{schema-elements-back} can ensure that under the condition of indirect mapping, directly data exchange from the source to the target can be realized, which can improve the efficiency of data transformation. Now we show the process of indirect mapping composition.

\textit{Schema-elements-back} is the process of adding new elements to the source schema according to the middle schema, in order to achieve a directly mapping from the source schema to the target schema under the condition that new elements are introduced. As shown in figure 7, the dashed lines represent the correspondences between the newly added schema elements and elements in the middle schema, from which we can see that when \( S_1 \) is translated to \( S'_1 \), by \textbf{Theorem 2}, mapping composition \( M_{13} \) can be directly constructed from \( S'_1 \) to \( S_3 \).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{schema_elements_back.png}
\caption{Process of Schema-elements-back}
\end{figure}

Now we give an indirect mapping composition algorithm (denoted by IMCA), without loss of generality, we consider the case shown in figure 7, and for the other cases we just need some small modifications.

\textbf{Algorithm IMCA}

\textbf{Input:} \( M_{12} = (S_1, S_2, \Sigma_{12}) \) and \( M_{23} = (S_3, S_4, \Sigma_{23}) \), where \( \Sigma_{12} \) and \( \Sigma_{23} \) are extended S2Ttgds that describe indirect schema element correspondences.

\textbf{Output:} A schema mapping \( M_{13} = (S'_1, S_2, \Sigma_{13}) \), which is the composition of \( M_{12} \) and \( M_{23} \).

\textbf{Step1. Normalize} \( \Sigma_{12} \) and \( \Sigma_{23} \):

Rename the function and element symbols so that the symbols that appear in \( \Sigma_{12} \) are all distinct from those in \( \Sigma_{23} \).

\textbf{Step2. Construct virtual elements in} \( S_1 \):

If \( M_{12} : \pi_{\phi_1}(R_{\phi_1}^3(x_i)) \rightarrow \pi_{\phi_2}(R_{\phi_2}^3(x_i)) \) and \( M_{23} : \pi_{\phi_1}(R_{\phi_1}^4(x_i)) \rightarrow \pi_{\phi_2}(R_{\phi_2}^4(x_i)) \) satisfying that \( \theta'_2 \supset \theta_2 \).

\textbf{Then} for each \( i \) do projection operation \( \pi_{\theta_i}(R_{\theta_i}^5(x_i)) \), and get an attribute set \( E = \{E_1, \ldots, E_i\} \), where \( E_i \) is the set of attributes generated by \( R_{\theta_i}^5(x_i) \).

Add each set \( E_i \) into \( S_1 \) and combine it with attributes computed by \( \pi_{\phi_1}(R_{\phi_1}^3(x_i)) \), which is called a schema elements back process. As shown in the left dashed box of figure 7, we get the new constructed schema \( S'_1 \) by combining elements in \( E \) to \( S_1 \) (virtual elements are from \( S'_1 \)). Then we get the mapping \( M_{13} \) from \( S'_1 \) to \( S_3 \) with the formula set \( \Sigma_{13} \). Since the mapping from \( S_1 \) to \( S'_1 \) is one-to-one, we get \( M_{13} = M_{23} \), where \( S_2 \) gives the data back rules from \( S_1 \) to \( S'_1 \) during the data exchange process from \( S'_1 \) to \( S_3 \).

\textbf{Step3. Construction of} \( S_{12} \) and \( S_{23} \):

(3.1) Initialize \( S_{12} \) and \( S_{23} \) to each be the empty set.

Assume the formulas in \( \Sigma_{12} \) is:

\( (\forall x_i(\phi_1 \rightarrow \psi_i)) \wedge \cdots \wedge (\forall x_i(\phi_i \rightarrow \psi_i)) \)

(3.2) Put each of the \( n \) implications \( \phi_i \rightarrow \psi_i \), for \( 1 \leq i \leq n \), into \( S_{12} \). We do likewise for \( \Sigma_{23} \) and \( \Sigma_{23} \).

Each implication \( \chi \) in \( \Sigma_{12} \) has the form \( \phi(x) \rightarrow \bigotimes_{j} (\pi_{\phi_j}(R_{\phi_j}^3(x_j))) \otimes \cdots \otimes \pi_\phi(R_{\phi}^3(x_i))) \) where every member of \( x \) is a universally quantified variable, and each \( x_j \), for \( 1 \leq j \leq k \), is a sequences of terms on \( x \).

(3.3) Replace each such implication \( \chi \) in \( S_{12} \) with \( k \) implications:

\( \phi(x) \rightarrow \pi_{\phi_1}(R_{\phi_1}^3(x_i)) \otimes \cdots \otimes \pi_{\phi_k}(R_{\phi_k}^3(x_i)), \ldots, \phi(x) \rightarrow \pi_{\phi_k}(R_{\phi_k}^3(x_i)) \otimes \cdots \otimes \pi_\phi(R_{\phi}^3(x_i))) \).

\textbf{Step4. Compose} \( S_{12} \) and \( S_{23} \):

Repeat the following until every schema symbol in the left-hand side of every formula in \( S_{23} \) is from \( S_1 \).

For each implication \( \chi \) in \( S_{23} \) of the form \( \psi \rightarrow \gamma \) where there is an atom \( R(y) \) (\( R \) is a scheme symbol in \( S_2 \)), perform the following steps to replace \( R(y) \) with atoms in \( S'_1 \).

(4.1) Let \( \phi_1 \rightarrow R(t_1), \ldots, \phi_k \rightarrow R(t_k) \) be all the formulas in \( S_{12} \) whose right-hand side has \( R \) in it. If no such implications exist in \( S_{12} \), we remove \( \chi \) from \( S_{23} \).

Otherwise, for each such formula \( \phi_i \rightarrow R(t_i) \), rename the variables in this formula so that they do not overlap with the variables in \( \chi \).

(4.2) Remove \( \chi \) from \( S_{23} \) and add \( p \) formulas to \( S_{23} \) as follows: replace \( R(y) \) in \( \chi \) with \( \phi_i \) and add the resulting formula to \( S_{23} \), for \( 1 \leq i \leq p \).

\textbf{Step5. Construct} \( M_{13} \):

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Copyright © 2009 MECS} & \\
\hline
\textbf{I.J. Image, Graphics and Signal Processing, 2009, 1, 50-58} & \\
\hline
\end{tabular}
\end{table}
Let \( S_{23} = (\chi_1, \ldots, \chi_r) \) where \( \chi_1, \ldots, \chi_r \) are all the formulas from step 4. Let \( \Sigma_{12} = \forall z_i \chi_i \wedge \forall z_i \chi_i \) where \( z_i \) is the set of variables found in \( \chi_i \), for \( 1 \leq i \leq r \).

**Return** \( M_{12} = (S'_1, S_1, \Sigma_{12}) \).

A new source schema should be maintained by the composition algorithm, and such a schema may not be useful in other mapping compositions. However, if the scale of data to be exchanged is large, such new extended schema can improve the efficiency of data exchange.

**VI. EXPERIMENTS**

**A. Complexity of ICMA**

Let the elements that schema \( S_1, S_2 \) and \( S_3 \) respectively contain be \( N_{11}, N_2 \) and \( N_3 \). In step 1 of ICMA, the rename operation need to match each elements among \( S_1, S_2 \) and \( S_3 \), whose complexity is \( O(N_1 \times N_2 \times N_3) \).

Let the correspondences in \( M_{12} \) be \( N_{12} \) and correspondences in \( M_{23} \) be \( N_{23} \), where \( N_{12} \) and \( N_{23} \) are the numbers of atom elements. The number of virtual elements constructed in schema-element-back process is \( N_{23} - N_{12} \), so the complexity for constructing virtual elements is \( O(N_{23} - N_{12}) \).

The complexity for composing is decided by the atom implications that \( \Sigma_{12} \) and \( \Sigma_{23} \) contain. Let \( N_{12} \) be the number of implications in \( \Sigma_{12} \) with the form of \( \phi_i \rightarrow \gamma_i \) and \( n_{12} \) be the atom formula in \( \gamma_i \), and then the number of formulas in \( \Sigma_{12} \) is \( N_{12} \times n_{12} \). If the number of implications with the form of \( \phi_i \rightarrow \gamma_i \) in \( \Sigma_{23} \) is \( N_{23} \), where \( \psi \) is the set of atom formulas. The number of pairs of atom formulas that should be matched during the replacing process is \( N_{12}^2 \times N_{23}^2 \), so the composing complexity is \( O(N_{12}^2 \times N_{23}^2 \times N_{23}^2) \). The whole complexity for indirect mapping composition is \( O(N_1 \times N_2 \times N_3) + O(N_{23} - N_{12}) + O(N_{12}^2 \times N_{23}^2) \). The complexity of direct mapping composition process is \( O(N_1 \times N_2 \times N_3) + O(N_{12}^2 \times N_{23}^2) \), both are polynomial complexities.

**B. Results and analysis**

(1) Query efficiency under indirect mapping composition

Given the numbers of elements in schema \( S_1, S_2 \) and \( S_3 \), and the numbers of implications in \( \Sigma_{12} \) and \( \Sigma_{23} \). Given different scales of indirect mappings, now we compute the numbers of virtual elements constructed during the process of schema-element-back. We use the example in DBLP, chose 20 literature base by the sequence of letters, and pick up the meta-information of each literature. We construct the mappings manually as follows (take three of them for example):

**Author \( \otimes \) Author \( \otimes \ldots \otimes \) Author \( \rightarrow \) AuthorSet
Country \( \otimes \) University \( \otimes \ldots \otimes \) City \( \rightarrow \) AuthorAddress

Since virtual elements are introduced during the process of indirect mapping composition, figure 8 shows a comparison for numbers related by the query for non-mapping-composition and mapping-composition respectively.

As we can see that, although some virtual elements participate the querying and data exchanging, compared with the non-composition way, indirect mapping composition can reduce the number of mappings related by the query and improve the query efficiency.

Consider the indirect mappings shown in figure 4, we simulate the data exchange process from \( S_1 \) to \( S_3 \), and compare the executing times between indirect mapping composition and non-composition under different scale of instances. We use Oracle 9i as the database, and the time for reading records in the oracle database is the increase function of record scale and attributes numbers.

In the case of non-composition, instances \( \{I_1\} \) of \( S_1 \) are firstly transformed to instances of \( S_2 \) according to \( M_{12} \), denoted as \( \{I_2\} \), then \( \{I_2\} \) are transformed to \( \{I_3\} \) of \( S_3 \), the read and write of data happens among three schemas.

In the case of composition, since there are indirect mappings from \( S_1 \) to \( S_3 \), using the schema element back given in IMCA, instance for part attributes \( S_2 \) of are firstly transformed to \( S_3 \), and we get the new instances \( \{I'_1\} \), then instances \( \{I'_1\} \) of \( S_1 \) are generated according to \( M_{12} \).

Now we give the executing time (milliseconds) for the data exchange process from \( S_1 \) to \( S_3 \) under different instance scale of \( S_1 \), Figure 9 show the data exchange executing time comparison results.
Mechanism and Algorithm for Indirect Schema Mapping Composition

Figure 9. Data exchange process executing time comparisons.

Figure 10. Comparison for affection of middle schema between BECA and IMCA

(2) Comparison of BECA and IMCA

In BECA, symbols in the middle schema are eliminated by some algebra operations step by step using view unfolding, left composing and right composing. The main idea of BECA is to replace the symbols in the middle schema by the symbols in the source and target schemas as much as possible. Let the elements that participate in the process of elimination obey the equality distribution, and the numbers of virtual elements are decided by step 2 of IMCA, figure 10 shows the comparison between number of element symbols eliminated by BECA and number of virtual elements constructed in IMCA.

VII. CONCLUSION

Mapping composition is an active aspect in the research of schema mapping management, the object of which is to reuse mappings as a model, and simplify the process of data exchange. Especially when the scale of peers in the network is large, composing mappings can improve the data exchange efficiency. In this paper, we give the definition of indirect mappings, and discuss the composability of indirect mappings. Then we propose an indirect mapping composition algorithm using the schema-element-back strategy. Experiment results show that although IMCA does not improve the query performance, but the affection of middle schema can be eliminated.

REFERENCES


---

**Wang Bo**, born in Shenyang, China, in September 18th, 1980, got his Ph.D in National University of Defense Technology, Changsha, China in 2009. His current research interests include heterogeneous information integration and schema mapping.

**Guo Bo**, born in 1963. Professor and PH.D. supervisor. His main research interests include system management and integration and information management system.