Composing Local-As-View Mappings: Closure and Applications

Patricia C. Arocena
University of Toronto
prg@cs.toronto.edu

Ariel Fuxman
Microsoft Research
arielf@microsoft.com

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

ABSTRACT

Schema mapping composition is a fundamental operation in schema management and data exchange. The mapping composition problem has been extensively studied for a number of mapping languages, most notably source-to-target tuple-generating dependencies (s-t tgds). An important class of s-t tgds are local-as-view (LAV) tgds. This class of mappings is prevalent in practical data integration and exchange systems, and recent work by ten Cate and Kolaitis shows that such mappings possess desirable structural properties.

It is known that s-t tgds are not closed under composition. That is, given two mappings expressed with s-t tgds, their composition may not be definable by any set of s-t tgds (and, in general, may not be expressible in first-order logic). Despite their importance and extensive use in data integration and exchange systems, the closure properties of LAV composition remained open to date. The most important contribution of this paper is to show that LAV tgds are closed under composition, and provide an algorithm to directly compute the composition.

An important application of our composition result is that it helps to understand if given a LAV mapping $\mathcal{M}_{st}$ from schema S to schema T, and a LAV mapping $\mathcal{M}_{ts}$ from schema T back to S, the composition of $\mathcal{M}_{st}$ and $\mathcal{M}_{ts}$ is able to *recover* the information in any instance of S. Arenas et al. formalized this notion and showed that general s-t tgds mappings always have a recovery. Hence, a LAV mapping always has a recovery. However, the problem of testing whether a given $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$ is known to be undecidable for general s-t tgds. In contrast, in this paper we show the tractability of the problem for LAV mappings, and give a polynomial-time algorithm to solve it.

Categories and Subject Descriptors.
H.2.5 [**Heterogeneous Databases**]: Data Translation

1. Introduction

A *schema mapping* is a specification that describes how data in one schema (a source schema) may be transformed into data in a second schema (a target schema). In many data integration and data exchange applications, such as schema evolution and peer data sharing systems, we may want to be able to compose two schema mappings. Given a mapping $\mathcal{M}_{12} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_{12})$ from schema $\mathbf{S_1}$ to schema $\mathbf{S_2}$, and a second mapping $\mathcal{M}_{23} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_{23})$ from schema $\mathbf{S_2}$ to schema $\mathbf{S_3}$, we may want to produce a composed mapping $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$ directly from $\mathbf{S_1}$ to $\mathbf{S_3}$ (where $\mathbf{S_3}$ does not use any symbols from $\mathbf{S_2}$). In our work, we adopt the semantics for composition given by Fagin, Kolaitis, Popa and Tan [9]. Specifically, if a mapping is viewed as a binary relation over the instances of the source and target schemas, then for $\mathcal{M}_{13}$ to be the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ it must be the case that the following holds: for every pair of instances I and K, $\langle I, K\rangle \models \Sigma_{13}$ iff there exists some J such that $\langle I, J\rangle \models \Sigma_{12}$ and $\langle J, K\rangle \models \Sigma_{23}$.

A fundamental problem in mapping composition is understanding when a mapping specification language is closed under composition. One of the most important languages for schema mappings is that of source-to-target (s-t) tuple generating dependencies (tgds). An important class of s-t tgds are local-as-view (LAV) tgds. This class of mappings is prevalent in practical data integration and exchange systems, and recent work by ten Cate and Kolaitis [20] shows that it possesses desirable structural properties.

It is known that s-t tgds are not closed under composition [9]. That is, given two mappings expressed with s-t tgds, their composition may not be definable by any set of s-t tgds (and, in general, may not be expressible in first-order logic). Two languages that are known to be closed under composition are *full s-t tgds*, a restrictive subclass of s-t tgds containing no existentials, and *second-order tgds*, a generalization of s-t tgds which permits function symbols and equalities [9]. Little is known about the closure properties of classes of schema mappings in between these two extremes. Despite their importance and extensive use in data integration and exchange systems, the closure properties of LAV mapping composition remained open to date. The most important contribution of this paper is to show that LAV tgds are closed under composition, and provide an algorithm to compute the composition.

While our results imply that LAV mapping composition can always be expressed using LAV tgds, previous composition algorithms [9, 18] may produce mappings with second-order tgds when given LAV mappings as input. This is shown in the following example.

Example 1.1. *Consider the following LAV schema mappings $\mathcal{M}_{12} = (\mathbf{S}, \mathbf{T}, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{T}, \mathbf{R}, \Sigma_{23})$, where Σ_{12} and Σ_{23} are as follows:*

$$\begin{array}{ll} \Sigma_{12}: & \{\ \forall x_1 \forall x_2\, S(x_1, x_2) \rightarrow \exists z\, T(x_1, z)\ \} \\ \Sigma_{23}: & \{\ \forall y_1 \forall y_2\, T(y_1, y_2) \rightarrow \exists w\, R(y_2, w)\ \} \end{array}$$

If we run Fagin et al.'s composition algorithm [9] with the above input s-t tgds, we obtain the following second-order tgd as a result:

$$\begin{array}{ll} \Sigma_{13}: & \{\ \exists f \exists g\, (\forall x_1 \forall x_2\, S(x_1, x_2) \\ & \quad \rightarrow R(f(x_1, x_2), g(x_1, f(x_1, x_2))))\ \} \end{array}$$

In general, it is not obvious if the second-order tgds produced by this composition algorithm are equivalent to any s-t tgd (a point illustrated very well by Nash, Bernstein and Melnik [18] whose work we will consider in more detail in the next section). We consider this problem for LAV tgds and show, in this paper, that the composition of LAV mappings can always be expressed as a set of LAV tgds. In this particular example, the LAV formula produced by our composition algorithm is the following:

$$\Sigma'_{13}: \quad \{\ \forall x_1 \forall x_2\, S(x_1, x_2) \rightarrow \exists z_1 \exists z_2\, R(z_1, z_2)\ \}$$

As it has been pointed out by recent work of ten Cate and Kolaitis [20], LAV is a more desirable mapping language, in many ways, than second-order tgds. In particular, LAV mappings have properties that second-order mappings lack, namely closure under target homomorphisms and closure under union.

Definition 1.2. **[Closure under Target Homomorphisms]** *[20] Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a schema mapping from the source $\mathbf{S}$ to the target $\mathbf{T}$. We say $\mathcal{M}$ is closed under target homomorphisms if for all $\langle I, J \rangle \in \mathcal{M}$ and for all homomorphisms $h : J \rightarrow J'$, we have $\langle I, J' \rangle \in \mathcal{M}$.*

Definition 1.3. **[Closure under Union]** *[20] Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ be a schema mapping from the source $\mathbf{S}$ to the target $\mathbf{T}$. We say $\mathcal{M}$ is closed under union if $\langle \emptyset, \emptyset \rangle \in \mathcal{M}$ and for all $\langle I, J \rangle \in \mathcal{M}$ and $\langle I', J' \rangle \in \mathcal{M}$ (not necessarily disjoint), we have $\langle I \cup I', J \cup J' \rangle \in \mathcal{M}$.*

The former is a structural property shared by all mappings given in terms of s-t tgds, but not by all second-order tgds. The latter is a property that only LAV mappings possess; not even full mappings are closed under union. From a more practical point of view, the problem of *answering queries using views* which has important applications in data integration, has been extensively studied in the literature for LAV mappings [1] and efficient rewritings are known for LAV [19]. Our results imply that these results can also be applied to the *composition* of any number of LAV mappings.

A number of notions of mapping inverse have been proposed in the literature [4, 7, 10, 11], including *schema recovery* and *maximum recovery* [4]. Arenas, Pérez, and Riveros showed that general s-t tgds mappings always have a recovery. Hence, this also holds for LAV mappings. To complete the picture on this property, it is necessary to address the following question: given mappings $\mathcal{M}_{st}$ and $\mathcal{M}_{ts}$, is $\mathcal{M}_{ts}$ a recovery of $\mathcal{M}_{st}$? The problem is known to be undecidable for general s-t tgds. In contrast, in this paper we show that the problem is tractable for LAV mappings, and give a polynomial-time algorithm to solve it.

Contributions. The main contributions of this work are the following:

- We show that the composition of LAV mappings is not only first-order, but can always be expressed as a set of LAV tgds. This adds to a number of properties recently studied for LAV mappings [20].
- We present a novel composition algorithm for composing LAV schema mappings. In contrast to algorithms for composing s-t tgds or more general mappings [9, 18, 5], our proposed algorithm avoids skolemization and the subsequent problem of de-skolemization (the elimination of Skolem functions). The algorithm directly produces a finite set of LAV tgds as output.
- We address the following question: given two LAV mappings $\mathcal{M}_{st}$ and $\mathcal{M}_{ts}$, is $\mathcal{M}_{ts}$ a recovery of $\mathcal{M}_{st}$? We show that the problem is tractable and give a polynomial-time algorithm to solve it. This is in contrast to the case of general s-t tgds, where the problem is undecidable.

The rest of the paper is organized as follows. In the next section, we review related work. In Section 3, we introduce the notation and terminology used in the paper. In Section 4, we present the algorithm to compose LAV schema mappings. In Section 5, we present our results on recovery checking for LAV mappings. Finally, in Section 6, we give some concluding remarks and directions for future work.

2. Related Work

Mapping composition is one of the fundamental operations in *model management* [6]. In this work, we use the semantics for mapping composition that was introduced by Fagin et al. [9], which is independent of the class of queries used. Other semantics have also been proposed. Madhavan and Halevy [16] defined the composition operator relative to a class of queries. Yu and Popa [21] introduced a semantics especially tailored for mapping adaptation, where the notion of equivalence in the composition problem formulation is defined in terms of universal solutions [8].

Despite the importance of LAV mappings in data integration systems [15], there were to date no algorithms guaranteed to always produce an s-t tgd from the composition of LAV tgds. Our results are the first to show that LAV tgds are actually closed under mapping composition. It is known that s-t tgds are not closed under composition [9]. Furthermore, Nash et al. showed that checking the closure properties for given first-order mappings is in general undecidable [18].

Fagin et al.'s composition algorithm [9] takes as input a set of second-order tgds, and produces another second-order tgd that expresses the composition. In general, transforming an existential second-order formula into an equivalent first-order formula may not be possible. The second order tgds may contain Skolem function symbols, and the process of removing these symbols has been studied in the logic literature under the names of *reverse skolemization*, *unskolemization* and *de-skolemization* [13]. Even for two LAV mappings, the Fagin et al. [9] composition algorithm may produce a mapping expression that uses second-order constructs (and is not expressed as a LAV or even an s-t tgd).

Recently Nash et al. [18] introduced the first de-skolemization algorithm specifically tailored to work on mapping composition settings. The algorithm is sound but not complete, and the authors show how to check in polynomial time whether it will succeed for a given input. Notably, the algorithm is not guaranteed to succeed even for second-order tgds produced by composing LAV mappings. In particular, the algorithm fails to de-skolemize the second-order tgd Σ_{13} in Example 1.1 due to the presence of nested Skolem terms.

Bernstein, Green, Melnik and Nash [5] presented an efficient algorithm for composing schema mappings that is based on view unfolding. The algorithm relies on the notion of *normalization* of the dependencies used in the mappings. A *normalized dependency* is a dependency that has exactly one relation symbol on either the left or the right-hand side of the implication. The output of Bernstein et al.'s algorithm is not guaranteed to replace all the relation symbols of the middle schema of the composition setting. That is, in some cases, the algorithm returns a formula that contains some relation symbols from that middle schema and it is thus not strictly a composition formula in the sense of Fagin et al. [9]. Furthermore, the algorithm relies on Nash et al.'s de-skolemization algorithm to remove Skolem terms introduced in the normalization step. In contrast, the algorithm that we present in this paper always removes all symbols from the middle schema and it avoids the problem of de-skolemization.

Likewise for mappings given in terms of s-t tgds, verifying whether a given pair of source and target instances satisfies a LAV tgd can be done in polynomial time for all fixed mappings and instances [8]. This is in contrast to second-order tgds where this problem is NP-complete [9]. In practice, this means that in the case of second-order tgds, if a source and a target act in an autonomous way (making updates independently), then an expensive check may have to be done to verify whether their updated instances satisfy a mapping.

Another fundamental operation used in model management is the inverse operator [6]. The precise semantics of the inverse of a schema mapping is based on the idea that a mapping composed with its inverse yields the identity mapping [7]. This definition of inverse proved to be quite strict and as a result, several relaxations have been studied [10, 4, 11, 3]. Of these, we will consider the notion of mapping recovery, for which it is known that all LAV mappings have a recovery [4]. Arenas et al. showed that the problem of deciding whether a mapping is a recovery of another one is undecidable for s-t tgds [4]. In the case of mappings given by full s-t tgds, the complexity of recovery checking is coNP-complete. In contrast, in this paper we show that this problem is tractable for LAV mappings, and give a polynomial-time algorithm to solve it.

Fagin et al. also studied extensions of invertibility to cope with the presence of nulls [11]. These extensions apply to mappings that are closed under target homomorphisms, which provide another motivation for understanding when the composition of s-t tgds can be expressed as a set of s-t tgds or LAV tgds.

3. Preliminaries

A *schema* is a non-empty finite set $\mathbf{R} = (R_1, \ldots, R_k)$ of relation symbols where each R_i has a fixed arity. We define the notion of *instance* I over a schema $\mathbf{R}$ in the normal way for relational schemas, that is, as the union of relation instances over R_i, where $R_i \in \mathbf{R}$. Unless otherwise noted, we assume that all values in an instance come from a set of *constant* values. Let $\mathbf{S} = (S_1, \ldots, S_n)$ and $\mathbf{T} = (T_1, \ldots, T_m)$ be two disjoint schemas. Unless otherwise stated, we follow the convention of referring to $\mathbf{S}$ as the *source* schema and to $\mathbf{T}$ as the *target* schema. The notation $(\mathbf{S}, \mathbf{T})$ denotes the schema $(S_1, \ldots, S_n, T_1, \ldots, T_m)$. When necessary, we distinguish between instances over $\mathbf{S}$ and instances over $\mathbf{T}$ by adding the prefix *source* and *target*, respectively.

We use the standard notion of satisfaction of a formula in first-order logic. If K is an instance and φ is a formula, we write $K \models \varphi$ to denote that K satisfies φ. The same notion is applied over a set of formulas Σ.

A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$, where $\mathbf{S}$ and $\mathbf{T}$ are schemas with no relation symbols in common and Σ_{st} is a set of logical formulas over $(\mathbf{S}, \mathbf{T})$ [9]. An *instance of* $\mathcal{M}$ is an instance $\langle I, J\rangle$ over $(\mathbf{S}, \mathbf{T})$ that satisfies every formula in Σ_{st}. We use the notation $Inst(\mathcal{M})$ to denote the set of all instances $\langle I, J\rangle$ of $\mathcal{M}$. If $\langle I, J\rangle \in Inst(\mathcal{M})$, then we call J a *solution of* I *under* $\mathcal{M}$.

A *source-to-target tuple-generating dependency* (s-t tgd) is a formula of the form $\forall \mathbf{z}, \mathbf{x}(\phi(\mathbf{z}, \mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where $\mathbf{z}$, $\mathbf{x}$, and $\mathbf{y}$ are disjoint vectors of variables; $\phi(\mathbf{z}, \mathbf{x})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$; and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$. We require that all variables of $\mathbf{z} \cup \mathbf{x}$ are used in ϕ and all variables of $\mathbf{x} \cup \mathbf{y}$ are used in ψ. A *full source-to-target tuple-generating dependency* (full s-t tgd) is an s-t tgd of the form $\forall \mathbf{z}, \mathbf{x}(\phi(\mathbf{z}, \mathbf{x}) \rightarrow \psi(\mathbf{x}))$, where $\phi(\mathbf{z}, \mathbf{x})$ is a conjunction of atomic formulas over the source schema $\mathbf{S}$, and $\psi(\mathbf{x})$ is a conjunction of atomic formulas over the target schema $\mathbf{T}$.

A *second-order tuple-generating dependency* (second-order tgd) is an existential second-order formula of the form

$\exists \mathbf{f}(\,(\forall \mathbf{x_1}(\phi_1 \rightarrow \psi_1)) \wedge \cdots \wedge (\forall \mathbf{x_n}(\phi_n \rightarrow \psi_n))\,)$ where (1) each member of $\mathbf{f}$ is a function symbol; (2) each ϕ_i is a conjunction of (a) atomic formulas over the source schema $\mathbf{S}$ and (b) equalities of the form $t = t'$ where t and t' are terms based on $\mathbf{x_i}$ and $\mathbf{f}$; (3) each ψ_i is a conjunction of atomic formulas over the target schema $\mathbf{T}$; and (4) each variable in $\mathbf{x_i}$ appears in some atom of ϕ_i.

Given a schema mapping $\mathcal{M}$ and a source instance I over $\mathbf{S}$, the problem of finding a solution J over the target schema $\mathbf{T}$ is known as the *data exchange problem.* For any mapping $\mathcal{M}$, there may be many solutions for a given source instance I. Let $\mathbf{R}$ be a schema, and J and J' two instances over $\mathbf{R}$. A function h is a *homomorphism* from J to J' if (1) $h(c) = c$ for every constant c, and (2) for every relation symbol of R in $\mathbf{R}$, and every tuple $R(a_1, \ldots, a_k) \in J$, we have that $R(h(a_1), \ldots, h(a_k)) \in J'$.[1] Given $\mathcal{M}$ and a source instance I, a *universal solution* of I under $\mathcal{M}$ is a solution U of I under $\mathcal{M}$ such that for every solution J of I under $\mathcal{M}$, there exists a homomorphism $h : U \rightarrow J$ with the property that $h(v) = v$ for every source value occurring in I. If Σ consists of s-t tgds, then *chasing* I with Σ produces a universal solution U of I under $\mathcal{M}$. Target values introduced at any time during a chase step that do not appear in the source instance (i.e., that are not constants from the source instance) are called *instance labeled nulls* (or instance nulls, for short) [8]. Thus, values in a target instance come from the union of a set of *constant* values and a set of (disjoint) *instance labeled nulls.*

Next, we recall the concept of mapping composition [9]. Given two schema mappings $\mathcal{M}_{12} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_{23})$, the *composition* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is a schema mapping $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$ such that for every instance I over $\mathbf{S_1}$ and every instance K over $\mathbf{S_3}$, we have $\langle I, K\rangle \models \Sigma_{13}$ if and only if there is an instance J over $\mathbf{S_2}$ such that $\langle I, J\rangle \models \Sigma_{12}$ and $\langle J, K\rangle \models \Sigma_{23}$. When mappings are understood from the context, we shall often use $\Sigma_{12} \circ \Sigma_{23}$ to denote the *composition formula* of $\mathcal{M}_{12} \circ \mathcal{M}_{23}$.

A *tableau* over a schema $\mathbf{R}$ is an instance of $\mathbf{R}$ where a tuple of the tableau may contain values drawn from the set of constants or from a (disjoint) set of variables [2]. An *embedding* of a tableau $\mathbf{D}$ into instance I is a valuation v for the variables occurring in $\mathbf{D}$ such that $v(\mathbf{D}) \subseteq I$. In our work, we will use the notion of tableau as an embodiment of the information captured by a formula over $\mathbf{R}$ [17]. Others have described how to create a tableau for a given query [2]. Here we show how to do this for a formula of the form $\forall \mathbf{x} \exists \mathbf{y} \phi(\mathbf{x}, \mathbf{y})$. A *formula tableau* is constructed by creating a tableau tuple for each single atomic relational formula appearing in $\phi(\mathbf{x}, \mathbf{y})$. For each universally quantified variable, we create a *tableau universal variable* in the tableau, and for each existential variable in the formula, we create a *tableau existential variable.* We shall often refer to tableau existential variables as *tableau existential nulls*, or simply, *tableau nulls.* Thus an embedding of a formula tableau tuple into an instance tuple maps (a) tableau universal variables to constants values in the instance tuple, and (b) tableau existential variables to either constant values or instance labeled nulls in the instance. We use lowercase letters (e.g. x and y) and upper case letters (e.g. Z, U, V and W), possibly with subscripts, to indicate tableau universal variables and tableau existential variables, respectively.

[1]For notational convenience we say that the homomorphism is from J to J', but as a function the homomorphism is from the variables and constants of J to the variables and constants of J'.

4. Composing LAV Mappings

In this section, we present the main result of the paper: an algorithm that given two LAV mappings, produces a LAV mapping that expresses their composition. We first define the the notion of LAV tgds and mappings. Second, we present the intuition behind our algorithm. Finally, we present the algorithm and its correctness proof.

4.1 Local-As-View Mappings

LAV mappings were initially introduced by Levy, Rajaraman, and Ordille [15] to overcome some limitations in the use of traditional views for data integration. The idea of a LAV mapping is that each relation symbol of the source (the *local* schema in the terminology of the day [14]) is defined with respect to the target schema (the *global* schema).

The following is the definition of LAV tgds that we use in this paper.

Definition 4.1. [**LAV tgd**] *A* LAV source-to-target tuple generating dependency (LAV tgd) *is an s-t tgd of the form*

$$\forall \mathbf{z} \forall \mathbf{x}\, S(\mathbf{z}, \mathbf{x}) \rightarrow \exists \mathbf{y}\, \psi_T(\mathbf{x}, \mathbf{y})$$

where $S(\mathbf{z}, \mathbf{x})$ is an atomic relational formula over a source $\mathbf{S}$; $\mathbf{z}$, $\mathbf{x}$ and $\mathbf{y}$ are mutually disjoint vectors of variables; and $\psi_T(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over a target schema $\mathbf{T}$. Every variable in $\mathbf{z}$ and $\mathbf{x}$ appears exactly once in $S(\mathbf{z}, \mathbf{x})$.

The definition simply states that a LAV tgd is a source-to-target tgd such that (1) it has exactly one literal on the left-hand side; and (2) every variable of $S(\mathbf{z}, \mathbf{x})$ must be distinct. This latter condition is a common underlying assumption in LAV systems such as the Information Manifold [15], where the goal is to explain a relation symbol of the source using a formula on the target (global) schema. Thus, the source relation symbol is denoted with one literal that has a (distinct) variable for each attribute of the source relation (and hence, in SQL terminology, there are no selection conditions on the source relation). Interestingly, some theoretical studies have allowed repeated variables in the left-hand side of a LAV tgd [8, 20].

Our definition of LAV is slightly more general than the one given by Lenzerini [14]. Lenzerini's definition of sound LAV views restricts $\mathbf{z}$ to be empty; in other words, all source variables must appear in the target formula [14]. We relax this last condition as is common in practical cases of data exchange.

We now define a LAV mapping as a source, a target, and a set of LAV tgds.

Definition 4.2. [**LAV Schema Mapping**] *A* LAV schema mapping *is a mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$*, where* $\mathbf{S}$ *and* $\mathbf{T}$ *are schemas with no relation symbols in common and* Σ *is a finite set of LAV tgds over* $(\mathbf{S}, \mathbf{T})$.

We will show shortly that LAV mappings are closed under composition. The two conditions defining LAV mappings are tight, in the sense that minimal relaxations of their conditions lead to mappings that are not closed under composition. We can show the tightness of the class by resorting to existing examples from the literature whose composition has been shown to be inexpressible using (first-order) s-t tgds.

Example 4.3. *For the relaxation of condition (1), consider the following two mappings in which there are no repeated variables on the left-hand side of the dependencies, but one of the dependencies has two literals on the left-hand side.*

$$\begin{aligned}\Sigma_{12}: \quad & \{\ \forall x\,(A(x) \rightarrow \exists y\, F(x,y)),\\ & \ \ \forall u\,(B(u) \rightarrow \exists v\, G(u,v))\ \}\\ \Sigma_{23}: \quad & \{\ \forall x \forall u \forall y \forall v\,(F(x,y) \wedge G(u,v)\\ & \quad \rightarrow T(x,y,u,v))\ \}\end{aligned}$$

Composing Σ_{12} *with* Σ_{23} *gives the following second-order result:*

$$\begin{aligned}\phi: \quad & \{\exists f \exists g\ (\forall x \forall u\\ & \ A(x) \wedge B(u) \rightarrow T(x, f(x), u, g(u))\ )\}\end{aligned}$$

A first guess for re-writing ϕ *as an equivalent s-t tgd might involve replacing Skolem functions* f *and* g *with existentially quantified variables* y *and* v*, respectively, as shown below:*

$$\phi': \quad \forall x \forall u\ (A(x) \wedge B(u) \rightarrow \exists y \exists v\ T(x,y,u,v)\ )$$

However, ϕ' *does not correctly capture the composition result since it introduces unwanted relationships: both existential variables* y *and* v *depend on both universally quantified variables* x *and* u*, instead of having* y *depend only on* x *and* v *depend on only* u*, as originally captured by the second-order result* ϕ*. In fact, it has been shown (see [18] Section 6, page 34) that the composition of the above mappings cannot be expressed using s-t tgds. However, they can be expressed using the following equivalent set of first-order sentences* θ*:*

$$\begin{aligned}& \forall x\, A(x) \rightarrow \exists y \forall u\ (B(u) \rightarrow T(x,y,u,v))\\ & \forall u\, B(u) \rightarrow \exists v \forall x\ (A(x) \rightarrow T(x,y,u,v))\end{aligned}$$

Example 4.4. *For the relaxation of condition (2), consider the following mappings where all dependencies have exactly one literal on the left-hand side, but one of the dependencies has repeated variables.*

$$\Sigma_{12}: \quad \{\forall e(Emp(e) \rightarrow \exists m Mgr_1(e,m))\}$$

$$\begin{aligned}\Sigma_{23}: \quad & \{\forall e \forall m(Mgr_1(e,m) \rightarrow Mgr(e,m)),\\ & \ \forall e(Mgr_1(e,e) \rightarrow SelfMgr(e))\}\end{aligned}$$

Tableau D

S	
x_1	x_2

Tableau E^*

T_1	
x_1	Y

T_2	
Y	x_2

Tableau F^*

R	
x_1	W
x_2	V
x_1	U
Y	U

Figure 1: Illustration of tableaux

Fagin et al. showed that the composition of these mappings is a second-order tgd with equalities (see [9], Section 5, page 1013 and pages 1016/17), that cannot be equivalently defined by any set of s-t tgds.

$$\begin{aligned}\Sigma_{13}: \quad & \{\exists f(\forall e(Emp(e) \rightarrow Mgr(e, f(e)))\\ & \ \wedge \forall e(Emp(e) \wedge e = f(e) \rightarrow SelfMgr(e)))\}\end{aligned}$$

4.2 Intuition of the Algorithm

We now present a simple example that illustrates the intuition behind our algorithm for composing LAV mappings. Consider the schemas $\mathbf{S}$, $\mathbf{T}$ and $\mathbf{R}$, and the schema mappings $\mathcal{M}_{12} = (\mathbf{S}, \mathbf{T}, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{T}, \mathbf{R}, \Sigma_{23})$ where Σ_{12} contains a single LAV dependency α, and Σ_{23} contains three LAV dependencies β_1, β_2, and β_3.

$$\alpha: \quad \forall x_1 \forall x_2\ (S(x_1,x_2) \rightarrow \exists y\, T_1(x_1,y) \wedge T_2(y,x_2))$$

$$\begin{aligned}\beta_1: \quad & \forall t_1 \forall t_2\ (T_1(t_1,t_2) \rightarrow \exists w\, R(t_1,w))\\ \beta_2: \quad & \forall t_3 \forall t_4\ (T_2(t_3,t_4) \rightarrow \exists v\, R(t_4,v))\\ \beta_3: \quad & \forall t_5 \forall t_6\ (T_1(t_5,t_6) \rightarrow \exists u\, R(t_5,u) \wedge R(t_6,u))\end{aligned}$$

Our algorithm constructs tableaux from the formulas, and relies on chasing these tableaux with the dependencies of Σ_{12} and Σ_{23}. The goal is to create a composition result that effectively removes terms from the middle schema (that is, in this example it means removing references to relational symbols T_1 and T_2 in $\mathbf{T}$).

We consider each dependency of Σ_{12} in turn. In our example, there is only one such dependency α. We take the left-hand-side of α and create a formula tableau $\mathbf{D}$ containing the single tuple $S(x_1, x_2)$ as illustrated in Figure 1, where universally quantified variables x_1 and x_2 are defined as tableau universal variables in the tableau. We interpret tableau $\mathbf{D}$ as defining a representative source instance over schema $\mathbf{S}$.

We now chase tableau $\mathbf{D}$ by applying α, and create a tableau $\mathbf{E}^*$. In this step, the chase creates two new tableau tuples $T_1(x_1, Y)$ and $T_2(Y, x_2)$, where the existential variable y in α is assigned a fresh tableau existential variable Y (as shown in Figure 1).

We proceed by chasing tableau $\mathbf{E}^*$ with all the dependencies in Σ_{23} until the chase terminates. In this example, we create a tableau $\mathbf{F}^*$ (Figure 1). At each chase step, a rule

β_i of Σ_{23} is fired if there is a row tuple in $\mathbf{E}^*$ that satisfies β_i's left-hand-side. Take dependency β_1: in this chase step, the application of β_1 generates the tuple $R(x_1, W)$ in $\mathbf{F}^*$, where the tableau universal variable x_1 comes from the tuple $T_1(x_1, Y)$ in $\mathbf{E}^*$ and, where the existential variable w in β_1 is assigned a fresh tableau existential variable W. Observe that any tableau universal variable appearing now in tableau $\mathbf{F}^*$ comes originally from tableau $\mathbf{D}$.

Consider dependency β_2. The application of β_2 to the tableau tuple $T_2(Y, x_2)$ generates the row $R(x_2, V)$ in $\mathbf{F}^*$, where existential variable v in β_2 is assigned a fresh tableau existential variable V.

Last, consider dependency β_3. The application of β_3 generates two rows $R(x_1, U)$ and $R(Y, U)$ in $\mathbf{F}^*$, where both tableau universal variable x_1 and tableau existential variable Y come from row $T_1(x_1, Y)$ in $\mathbf{E}^*$ and, where existential variable u in β_3 is assigned a fresh tableau existential variable U. Note that this chase step creates two tuples with the same tableau existential null (this effectively captures the role of existentially-quantified variable u representing a join in the right-hand-side of β_3). Again here, observe that tableau universal variable x_1 in tableau $\mathbf{F}^*$ comes originally from tableau $\mathbf{D}$; furthermore, observe that tableau existential variable Y was created while chasing tableau $\mathbf{D}$ with α.

Now consider taking the two tableaux $\mathbf{D}$ and $\mathbf{F}^*$ of Figure 1. We will use these tableaux to create a LAV tgd δ representing the composition. We take the only tuple of tableau $\mathbf{D}$ (remember this is exactly the case since α is a LAV tgd), and we use it to create the left-hand side of δ. The part of δ constructed so far reads as follows: "$\forall x_1 \forall x_2\ S(x_1, x_2)$". We do so by universally quantifying all variables of $\mathbf{D}$.

Then, we construct the right-hand side of δ. First, we take tableau $\mathbf{F}^*$, we identify the tableau existential variables, and we create a quantifier prefix that consists of one existentially-quantified variable for each tableau existential variable. Note that in the figure, we have used upper case letters for tableau existential variables; when we create existential variables for them, we shall instead use lower case letters. Second, we create a conjunction of atomic relational formulas, where each atomic formula corresponds to a tuple in $\mathbf{F}^*$. As result, we obtain the following formula for the right-hand side of δ: $\exists w \exists v \exists u \exists y\ (R(x_1, w) \wedge R(x_2, v) \wedge R(x_1, u) \wedge R(y, u))$.

Putting it all together, the LAV tgd δ that expresses the composition of Σ_{12} and Σ_{23} is the following:

$$\delta : \forall x_1 \forall x_2\ (S(x_1, x_2) \rightarrow \\ \exists w \exists v \exists u \exists y\ (R(x_1, w) \wedge R(x_2, v) \wedge R(x_1, u) \wedge R(y, u)))$$

4.3 Algorithm and Correctness Proof

We now present the algorithm to compose LAV schema mappings, which we call `ComposeLAV` $(\mathcal{M}_{12}, \mathcal{M}_{23})$. Algorithm 1 takes as input two LAV schema mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. It constructs the output LAV mapping $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$ by considering each dependency α of Σ_{12} one at a time (the loop of Lines 2-13). For each rule α it constructs a tableau $\mathbf{D}$ from α's left-hand side (Line 3), and chases $\mathbf{D}$ with Σ_{12} to obtain $\mathbf{E}^*$ (Line 4). Then, another tableau $\mathbf{F}^*$ is obtained by chasing $\mathbf{E}^*$ with Σ_{23} (Line 5). In Lines 7 to 11, the tableaux $\mathbf{D}$ and $\mathbf{F}^*$ are used to create a LAV tgd that is added to the output of the algorithm. The algorithm runs in polynomial time.

Algorithm 1 `ComposeLAV` $(\mathcal{M}_{12}, \mathcal{M}_{23})$

```
Input: LAV schema mappings $\mathcal{M}_{12} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_{12})$
    and $\mathcal{M}_{23} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_{23})$
Output: LAV schema mapping $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$,
    which is the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$
 1: Initialize $\Sigma_{13}$ to be the empty set
 2: for each rule $\alpha$ in $\Sigma_{12}$ do
 3:   Let $\mathbf{D}$ be the tableau of the left hand side of $\alpha$
 4:   Let $\mathbf{E}^*$ be the result of chasing $\mathbf{D}$ with $\Sigma_{12}$
 5:   Let $\mathbf{F}^*$ be the result of chasing $\mathbf{E}^*$ with $\Sigma_{23}$
 6:   if $\mathbf{F}^*$ is not empty then
 7:     Create a literal $R(\mathbf{x})$ from the (only) tuple in $\mathbf{D}$
 8:     Create a conjunction of literals $\psi(\mathbf{y})$ from the tuples
        in tableau $\mathbf{F}^*$
 9:     Let $w_1, \ldots, w_m$ be the variables of $\mathbf{y}$ that are
        tableau existential variables of $\mathbf{F}^*$
10:     Let $\delta$ be the LAV tgd $R(\mathbf{x}) \rightarrow \exists w_1 \ldots \exists w_m \psi(\mathbf{y})$
11:     Add $\delta$ to $\Sigma_{13}$
12:   end if
13: end for
14: Let $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$
15: return $\mathcal{M}_{13}$
```

Next, we show the correctness of the composition algorithm. That is, we show that given two schema mappings specified by LAV tgds, the algorithm returns a set of LAV tgds that is indeed their composition. The correctness proof uses properties of the chase procedure with s-t tgds [8] and homomorphism techniques. Note that if our definition of LAV were to allow repeated variables, our algorithm would have to generate equalities as shown in Example 4.4. Because we assume LAV without repeated variables, we do not need to do this. This is key to the correctness proof of our algorithm.

Theorem 4.5. *Let $\mathcal{M}_{12} = (\mathbf{S_1}, \mathbf{S_2}, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S_2}, \mathbf{S_3}, \Sigma_{23})$, where both Σ_{12} and Σ_{23} are sets of LAV tgds. Then, the algorithm* `ComposeLAV`$(\mathcal{M}_{12}, \mathcal{M}_{23})$ *returns in polynomial time a schema mapping $\mathcal{M}_{13} = (\mathbf{S_1}, \mathbf{S_3}, \Sigma_{13})$ such that Σ_{13} is also a set of LAV tgds and $\mathcal{M}_{13} = \mathcal{M}_{12} \circ \mathcal{M}_{23}$.*

PROOF. (*Sketch*) To show that the schema mapping $\mathcal{M}_{13}$ generated by the algorithm is the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$, we need to show that for every instance I over $\mathbf{S_1}$ and for every instance K over $\mathbf{S_3}$, we have that $\langle I, K \rangle \models \Sigma_{13}$ if and only if there is an instance J over $\mathbf{S_2}$ such that $\langle I, J \rangle \models \Sigma_{12}$ and $\langle J, K \rangle \models \Sigma_{23}$.

($\Rightarrow$) We will illustrate this direction of the proof with the diagram of Figure 2. Assume that there are instances I and K of $\mathbf{S_1}$ and $\mathbf{S_3}$, respectively, such that $\langle I, K \rangle \models \Sigma_{13}$. We need to show that there exists an instance J over $\mathbf{S_2}$ such that $\langle I, J \rangle \models \Sigma_{12}$ and $\langle J, K \rangle \models \Sigma_{23}$.

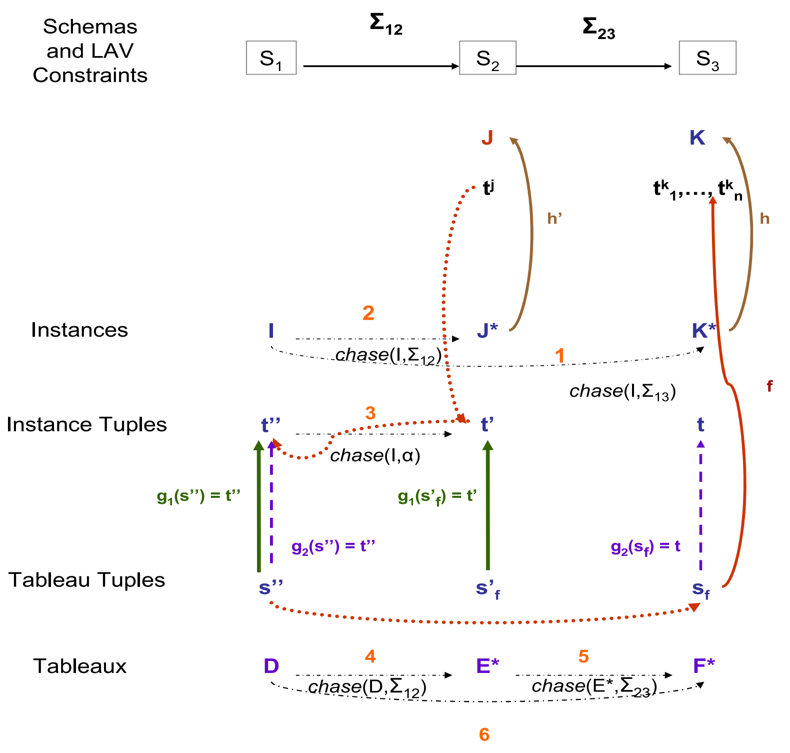

Figure 2: An illustration of Theorem 4.5

We construct J as follows. Let K^* be the result of chasing I with Σ_{13} (Arrow 1 in the figure). Since K^* is a universal solution for I in Σ_{13}, and instance K (from our hypothesis) is a solution for I in Σ_{13}, there must exist a homomorphism h from K^* to K. Next, let J^* be the result of chasing I with Σ_{12} (Arrow 2).

In the following steps, we shall construct an instance J over source $\mathbf{S_2}$, and show that $\langle I, J\rangle \models \Sigma_{12}$ and $\langle J, K\rangle \models \Sigma_{23}$. Two pieces of information are fundamental at this point: (1) homomorphism $h : K^* \rightarrow K$, and (2) the syntactic LAV restrictions on the input mappings. We shall use both of them to derive a homomorphism h' that, when applied to instance J^*, unveils the desired "middle" instance J.

The intuition behind why we can derive a homomorphism h' to construct J is as follows. To be a solution for Σ_{12}, J must be a homomorphic image of J^*; as such, it should contain constant values that are already present in J^*, or constant values which instantiate (some) nulls in J^*. In the first case, those constant values should be source values from instance I that were passed to J^* at some point during the chase procedure due to some rule α of Σ_{12}. Intuitively, a subset of those constant values should also be present in instance K^*. In the second case, constant values instantiating nulls should come directly from instance K (and, thus from homomorphism h) if we want to later show that $\langle J, K\rangle \models \Sigma_{23}$. Choosing exactly which labeled nulls in J^* map to constant values of K is the crux of the proof.

Because we do not know how to relate instances J^* and K^* at this point, the construction of homomorphism h' exclusively depends on the derivation chain of chase steps 4 and 5 (of Figure 2), and on the embeddings of the tableaux (generated by the algorithm) into instances I, J^* and K^*. As mentioned earlier, an important piece in this analysis is the fact that the dependencies are LAV. Consider a chase step with a rule α. Given a tuple t' in J^*, derived from an application of α, we know that since α is in LAV, there is exactly one instance tuple t'' in I such that tuple t' is the result of chasing tuple t'' with α (Arrow 3). For each rule α of Σ_{12}: let $\mathbf{D}$ be the tableau of the left-hand-side of α; following the algorithm, let $\mathbf{E}^*$ be the result of chasing $\mathbf{D}$ with Σ_{12} (arrow 4); let $\mathbf{F}^*$ be the result of chasing $\mathbf{E}^*$ with Σ_{23} (Arrow 5). By definition of the chase procedure, there are two homomorphisms $g_1 : \mathbf{D} \cup \mathbf{E}^* \rightarrow I \cup J^*$ and $g_2 : \mathbf{D} \cup \mathbf{F}^* \rightarrow I \cup K^*$, such that specific tableau tuples (i.e., s'', s'_f and s_f in the figure) can be transformed into instance tuples (i.e., t'', t' and t, respectively). Note that s'_f (respectively s_f) is one of the tuples in $\mathbf{E}^*$ (respectively $\mathbf{F}^*$).

Using homomorphisms h, g_1 and g_2, we can construct a homomorphism h' such that $h'(J^*) = J$. We define h' by analyzing the role played by each tableau variable in $\mathbf{E}^*$. Careful consideration should be taken when dealing with tableau existential variables as some of them embody exchanged existentials in Σ_{23}. In essence, the homomorphic instance-level images of those (exchanged) tableau existential variables (i.e., instance nulls in J^*) are the ones that are mapped to constant values in K.

After constructing an instance J over $\mathbf{S_2}$, we have $\langle I, J\rangle \models \Sigma_{12}$ because J^* is a solution and J is a homomorphic image of J. The last step involves showing that $\langle J, K\rangle \models \Sigma_{23}$. For this, we must show that for any rule β of Σ_{23}, a tuple

t_j in J satisfying β's left-hand-side produces one or more tuples $t_1^k, t_2^k, ..., t_n^k$ in instance K (this would mean that K satisfies β's right-hand-side). Again here, by reasoning on the derivation chain of chase steps 1 and 6, using the syntactic LAV restrictions on the setting and using homomorphisms h, h', g_1 and g_2, we show that there exists a homomorphism f from $\mathbf{D} \cup \mathbf{F}^*$ to $I \cup K$. Using homomorphism f, we know that tuple t_j in J produces a tuple t^k in K. That was to be shown.

($\Leftarrow$) As for the converse, it is not difficult to show that if $\langle I, K\rangle \in \mathit{Inst}(\mathcal{M}_{12}) \circ \mathit{Inst}(\mathcal{M}_{23})$, then $\langle I, K\rangle \models \Sigma_{13}$. $\square$

5. Recovery Checking

We now turn our attention to another important property of LAV mappings. This property is related to the *recovery* of a mapping, which was defined recently by Arenas, Pérez, and Riveros [4]. Informally, given a schema mapping $\mathcal{M}_{st}$, from the source to the target, a recovery operator computes a reverse schema mapping $\mathcal{M}_{ts}$ from the target to the source, such that for any instance I of S, it recovers all information in I. This was formalized in [4] as follows.

Definition 5.1. **[Schema Recovery]** [4] *Let $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$ be two schema mappings. Then, $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$ iff for every instance I, we have that $\langle I, I\rangle \in \mathcal{M}_{st} \circ \mathcal{M}_{ts}$.*

Arenas, Pérez, and Riveros [4] showed that every mapping defined as a set of s-t tgds has a recovery (in fact, they showed this for the stronger notion of *maximum recovery*). They also presented the following decision problem: given mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, check whether $\mathcal{M}_2$ is a recovery of $\mathcal{M}_1$. In the following, we will call this problem *recovery checking*. The main result of this section is to show a sharp contrast in the complexity of this problem for LAV mappings, as opposed to mappings expressed with general s-t tgds or even full tgds. In particular, we show that the problem can be solved in polynomial time for LAV mappings, whereas it is undecidable for general s-t tgds and coNP-complete for full tgds [4].

The problem of recovery checking has an important practical application: checking *update preservation* in, for example, systems where peers may autonomously develop their own mappings [12]. Consider a scenario where we have materialized versions of the source and the target. That is, we have mappings $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$, a source instance I, and a target instance J. Now, suppose that we update instance I to produce another instance I'. A natural question is whether it is possible to update J in order to produce another instance J' in such a way that $\langle I', J'\rangle$ satisfies the constraints of the mappings. More formally, we expect that $\langle I', J'\rangle \in \mathcal{M}_{st}$ and $\langle J', I'\rangle \in \mathcal{M}_{ts}$. This means that we must check whether for every I', $\langle I', I'\rangle \in \mathcal{M}_{st} \circ \mathcal{M}_{ts}$, which is precisely the recovery checking problem. In the following example, we illustrate the application of the recovery checking problem to update preservation.

Example 5.2. *Consider the scenario of an airline company that maintains a historical database with flight segments (point of origin and destination), and it has a Web application where it exposes non-stop and direct flights. Suppose that there is a mapping stating that every flight on the Web application has a corresponding flight segment on the database with the same airport of origin; and that all flight segments from the historical database have in the Web application a corresponding non-stop flight from the destination to the point of origin (thus allowing passengers to return to their point of origin using the same connection).*

To formalize this, let $\mathbf{S}$ be the schema of the Web application, and $\mathbf{T}$ be the historical database. Let $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$ be two schema mappings where Σ_{st} and Σ_{ts} are as follows:

$$\Sigma_{st}: \quad \forall x \forall y\, Flight(x, y) \rightarrow \exists c\, Segment(x, c)$$

$$\Sigma_{ts}: \quad \forall o \forall d\, Segment(o, d) \rightarrow Flight(d, o)$$

It follows from the results on recovery checking that we will present shortly that the recovery condition is not satisfied for these mappings. That is, there is some instance I such that $\langle I, I\rangle \not\models \mathcal{M}_{st} \circ \mathcal{M}_{ts}$. This implies that there are cases where an update to the Web application may be impossible to be translated to the historical database. In particular, consider a historical database that has information about two flight segments from $Toronto$ to $Munich$, and viceversa; and a Web application has non-stop flights from $Toronto$ to $Munich$ and from $Munich$ to $Toronto$. That is, $I = \{Flight(Toronto, Munich), Flight(Munich, Toronto)\}$, and $J = \{Segment(Toronto, Munich), Segment(Munich, Toronto)\}$. Now, suppose that we update the Web application and add a new non-stop flight from London to Munich. Let I' be the updated instance, that is: $I' = \{Flight(Toronto, Munich), Flight(Munich, Toronto), Flight(London, Munich)\}$. A natural question is: can we modify the historical database J in such a way that it remains consistent with respect to the updated Web application I' and the mappings? It is easy to see that $\langle I', I'\rangle \not\models \mathcal{M}_{st} \circ \mathcal{M}_{ts}$, which answers the question to the negative.

Consider now what happens if we have a different mapping which states that each flight segment in the historical database should have a corresponding flight in the Web application with the same point of origin. Formally, we have the following mappings $\mathcal{M}^2_{st} = (\mathbf{S}, \mathbf{T}, \Sigma^2_{st})$ and $\mathcal{M}^2_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma^2_{ts})$.

$$\Sigma^2_{st}: \quad \forall x \forall y\, Flight(x, y) \rightarrow \exists c\, Segment(x, c)$$

$$\Sigma^2_{ts}: \quad \forall o \forall d\, Segment(o, d) \rightarrow \exists z Flight(o, z)$$

In this case, we can use our results on recovery checking to show that any update to the Web application can be translated to the historical database. In particular, from our results it follows that $\mathcal{M}^2_{st}$ and $\mathcal{M}^2_{ts}$ satisfy the recovery condition. In terms of our example, this means the following. Let I be a Web application, and J be a historical database. Suppose that we update I and *obtain a new instance I'. Since the recovery checking condition is satisfied, we have that $\langle I', I'\rangle \models \mathcal{M}^2_{st} \circ \mathcal{M}^2_{ts}$. This means that we are guaranteed to have some sequence of updates to the historical database that produces an updated database instance J' that is consistent with the updated Web application I' and the mappings.*

5.1 Recovery Checking for LAV Mappings

Before presenting the algorithm, let us give some intuition on how our result on composition can be used to solve the recovery checking problem for LAV mappings. Consider the following dependency α of a composition formula Σ_{ss}:

$$\Sigma_{ss}: \quad \forall x \forall y\, S(x,y) \rightarrow \exists z\, S(x,z)$$

Now, let $\mathbf{A}$ and $\mathbf{B}$ be the tableaux of the left and right hand sides of α, respectively. That is, $\mathbf{A} = \{S(x,y)\}$ and $\mathbf{B} = \{S(x,Z)\}$. It is easy to see that $\mathbf{A}$ is contained in $\mathbf{B}$, that is, there is a homomorphism from $\mathbf{B}$ to $\mathbf{A}$. Since all the instances of $\mathbf{A}$ will be contained in the instances of $\mathbf{B}$, we can conclude that the mapping satisfies the recovery condition. Notice that it would have not been possible to make this argument if the composition was not expressible as tgds, since we are relying on conjunctive query containment.

The above intuition leads to the following algorithm. First, given $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$, compute the composition Σ_{ss} of Σ_{st} and Σ_{ts}. Now, in order to check whether $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$, it suffices to check that for each dependency α of Σ_{ss}, there is a homomorphism from the right-hand side of α to the left-hand side of α. We give Algorithm 2 and prove its correctness in Theorem 5.3.

Algorithm 2 `CheckRecovery`$(\mathcal{M}_{st}, \mathcal{M}_{ts})$

```
Input: LAV schema mappings $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$
    and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$
Output: True/False accordingly
1: Compute $\Sigma_{ss} = \Sigma_{st} \circ \Sigma_{ts}$
2: for each tgd $\alpha : \phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ of $\Sigma_{ss}$ do
3:   if there is no homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$
     then
4:     return false
5:   end if
6: end for
7: return true
```

Theorem 5.3. *Given LAV mappings $\mathcal{M}_{st}$ and $\mathcal{M}_{ts}$, the algorithm* `CheckRecovery`$(\mathcal{M}_{st}, \mathcal{M}_{ts})$ *returns* **true** *iff $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$.*

Proof. ($\Leftarrow$) Assume that the algorithm `CheckRecovery`-$(\mathcal{M}_{st}, \mathcal{M}_{ts})$ returns **true**. Let α be a dependency of Σ_{ss} of the form $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi(\mathbf{x}, \mathbf{y})$. Let I be a source instance. We must show $\langle I, I\rangle \models \Sigma_{ss}$. Assume that there is a homomorphism g from $\phi(\mathbf{x})$ to I such that $\phi(g(\mathbf{x}))$ is in I. Since `CheckRecovery`$(\mathcal{M}_{st}, \mathcal{M}_{ts})$ returns **true**, there is a homomorphism h from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$. Let $g' = g \circ h$. We must prove that $\psi(g'(\mathbf{x}, \mathbf{y}))$ is in I. To do so, let $R(\mathbf{w})$ be a literal of $\psi(\mathbf{x}, \mathbf{y})$. Since h is a homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$, we have that $R(h(\mathbf{w}))$ is a literal of $\phi(\mathbf{x})$. Since g is a homomorphism from $\phi(\mathbf{x})$ to I, we have that $R(g(h(\mathbf{w}))$ is in I. Thus, $R(g'(\mathbf{w}))$ is in I, which was to be shown.

($\Rightarrow$) Assume that $\mathcal{M}_{ts}$ is a schema recovery of $\mathcal{M}_{st}$. Assume towards a contradiction that the algorithm `Check-Recovery`$(\mathcal{M}_{st}, \mathcal{M}_{ts})$ returns **false**. Thus, there is some dependency of Σ_{ss} of the form $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi(\mathbf{x}, \mathbf{y})$ such that there is no homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$. Let I be an instance that consists exclusively of the tableau of $\phi(\mathbf{x})$. Since there is no homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$, we conclude that $I \not\models \psi(\mathbf{x}, \mathbf{y})$. Thus, $\langle I, I\rangle \not\models \Sigma_{ss}$; contradiction. $\square$

Two important observations are in order regarding the algorithm. First, it relies on the fact that Σ_{ss} is a set of tgds in order to check conjunctive query containment. This is the case when the input mappings are in LAV, as a result of our Theorem 4.5 of Section 4. Second, although in general conjunctive query containment is NP-complete, it is in polynomial-time for LAV mappings in particular. This follows from the following property.

Property 5.4. *Let $\alpha : \phi(\mathbf{x}) \rightarrow \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})$ be a LAV tgd. Then, the containment of $\phi(\mathbf{x})$ in $\psi(\mathbf{x}, \mathbf{y})$ can be checked in polynomial time.*

Proof. Since α is a LAV formula, it has exactly one literal in its left-hand side. Furthermore, by the definition of LAV, there are no repeated variables in ϕ. Let $\mathbf{A}$ and $\mathbf{B}$ be the tableaux of the left-hand-side and right-hand-side of α, respectively. Let $R(\mathbf{x})$ be the only tuple in tableau $\mathbf{A}$. Let $T(\mathbf{w})$ be one of the tuples in tableau $\mathbf{B}$. If $T \neq R$, there is no homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$. Now, assume that all the literals in the right-hand side of α are on the relation symbol R. Let f be a function constructed as follows. For every literal $R(\mathbf{w})$ in $\psi(\mathbf{x}, \mathbf{y})$, let f map the i-th variable of $\mathbf{w}$ to the i-th variable of $\mathbf{x}$. Now, it can be checked in linear time whether f is a homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$. It is easy to see that if f is not a homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$, there is no such other homomorphism from $\psi(\mathbf{x}, \mathbf{y})$ to $\phi(\mathbf{x})$ $\square$

From the above property, Theorem 5.3, and Theorem 4.5, the class of LAV schema mappings arises as a practical class where the recovery condition is tractable. This is in sharp contrast with the general case of mappings given by both s-t tgds, and also full s-t tgds, as earlier mentioned. Observe that even though LAV mappings permit incompleteness and have been shown to be of practical value, their complexity for the recovery checking problem is even less than for full tgds.

Corollary 5.5. *Let $\mathcal{M}_{st} = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$ and $\mathcal{M}_{ts} = (\mathbf{T}, \mathbf{S}, \Sigma_{ts})$ be two LAV schema mappings. Then, it can be checked in polynomial time whether $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$.*

6. Conclusions

In this paper, we showed that the composition of LAV mappings is closed under composition, and presented an algorithm for efficiently computing the composition. An important application of our composition result is that it helps to understand if a given LAV mapping $\mathcal{M}_{st}$ from schema S to schema T, and a LAV mapping $\mathcal{M}_{ts}$ from schema T back to S, the composition of $\mathcal{M}_{st}$ and $\mathcal{M}_{ts}$ is able to *recover* the information in any instance of S. Arenas et al. formalized this notion and showed that general s-t tgds mappings always have a recovery. Hence, a LAV mapping always has a recovery. However, the problem of testing whether a given $\mathcal{M}_{ts}$ is a recovery of $\mathcal{M}_{st}$ is known to be undecidable for general s-t tgds. In contrast, in this paper we show the tractability of the problem for LAV mappings, and give a polynomial-time algorithm to solve it.

Recovery checking for LAV is less expensive than for full tgds (where recovery checking is coNP-complete). This is despite the fact that LAV mappings permit the modeling of incompleteness and have proven to have much greater application in practical data integration systems. The key to this difference may lie in the fact that LAV mappings are closed under union [20], while full mappings are not. Our work exploited this property to construct compositions in a modular way.

There are many directions of future work. Regarding the closure properties, it would be interesting to find new classes of mappings that are closed under composition. In terms of practical applications, notice that the algorithms presented in this paper are efficient, simple to implement, and rely only on the schemas and mappings rather than the instances. We are thus planning to implement our techniques in a data integration and exchange system.

7. Acknowledgments

We would like to thank Phil Bernstein, Phokion Kolaitis, Lucian Popa and Wang-Chiew Tan for discussions with us, and to the anonymous referees for their comments. Arocena was partially supported by an NSERC Postgraduate Scholarship and the NSERC Business Intelligence Network.

8. References

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *PODS*, pages 254–263, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. Inverting Schema Mappings: Bridging the Gap between Theory and Practice. *PVLDB 2(1)*, pages 1018–1029, 2009.

[4] M. Arenas, J. Pérez, and C. Riveros. The Recovery of a Schema Mapping: Bringing the Exchanged Data Back. *PODS*, pages 13–22, 2008.

[5] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing Mapping Composition. In *VLDB*, pages 55–66, 2006.

[6] P. A. Bernstein, A. Y. Halevy, and R. A. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, 29(4):55–63, Dec. 2000.

[7] R. Fagin. Inverting Schema Mappings. *ACM Trans. Database Syst.*, 32(4):24, 2007.

[8] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theor. Computer Science*, 336(1):89–124, 2005.

[9] R. Fagin, P. Kolaitis, L. Popa, and W.-C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[10] R. Fagin, P. Kolaitis, L. Popa, and W.-C. Tan. Quasi-Inverses of Schema Mappings. *ACM Trans. Database Syst.*, 33(2):1–52, 2008.

[11] R. Fagin, P. Kolaitis, L. Popa, and W.-C. Tan. Reverse Data Exchange: Coping with Nulls. *PODS*, pages 23–32, 2009.

[12] A. Fuxman, P. Kolaitis, R. Miller, and W.-C. Tan. Peer Data Exchange. *ACM Trans. Database Syst.*, 31(4):1454–1498, 2006.

[13] D. Gabbay, R. Schmidt, and A. Szalas. *Second Order Quantifier Elimination: Foundations, Computational Aspects and Applications*. College Publications, 2008.

[14] M. Lenzerini. Data Integration: a Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[15] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, pages 251–262, 1996.

[16] J. Madhavan and A. Halevy. Composing Mappings among Data Sources. In *VDLB*, pages 572–583, 2003.

[17] A. O. Mendelzon. Database States and Their Tableaux. *ACM Trans. Database Syst.*, 9(2):264–282, 1984.

[18] A. Nash, P. Bernstein, and S. Melnik. Composition of Mappings Given by Embedded Dependencies. *ACM Trans. Database Syst.*, 32(1):4, 2007.

[19] R. Pottinger and A. Halevy. MiniCon: A Scalable Algorithm for Answering Queries using Views. *VLDB J.*, 10(2-3):182–198, 2001.

[20] B. ten Cate and P. Kolaitis. Structural Characterizations of Schema-Mapping Languages. In *ICDT*, pages 63–72, 2009.

[21] C. Yu and L. Popa. Semantic Adaptation of Schema Mappings when Schemas Evolve. *VDLB*, pages 1006–1017, 2005.