DOI:10.1145/1629175.1629201

Structural Characterizations of Schema-Mapping Languages

By Balder ten Cate and Phokion G. Kolaitis

Abstract

Information integration is a key challenge faced by all major organizations, business and governmental ones alike. Two research facets of this challenge that have received considerable attention in recent years are data exchange and data integration. The study of data exchange and data integration has been facilitated by the systematic use of schema mappings, which are high-level specifications that describe the relationship between two database schemas. Schema mappings are typically expressed in declarative languages based on logical formalisms and are chosen with two criteria in mind: (a) expressive power sufficient to specify interesting data interoperability tasks and (b) desirable structural properties, such as query rewritability and existence of universal solutions, that, in turn, imply good algorithmic behavior.

Here, we examine these and other fundamental structural properties of schema mappings from a new perspective by asking: How widely applicable are these properties? Which schema mappings possess these properties and which do not? We settle these questions by establishing structural characterizations to the effect that a schema mapping possesses certain structural properties if and only if it can be specified in a particular schema-mapping language. More concretely, we obtain structural characterizations of schema-mapping languages such as global-as-view (GAV) dependencies and local-as-view (LAV) dependencies. These results delineate the tools available in the study of schema mappings and pinpoint the properties of schema mappings that one stands to gain or lose by switching from one schema-mapping language to another.

1. INTRODUCTION

The aim of information integration is to synthesize information distributed over multiple heterogeneous sources into a single unified format. Information integration has been recognized as a key (and costly) challenge faced by large organizations today (see Bernstein and Haas[3, 12]). It is also well understood[12] that information integration is not a single problem but, rather, a collection of interrelated problems that include extracting and cleaning data from the sources, deriving a unified format for the integrated data, transforming data from the sources into data conforming with the unified format, and answering queries over the unified format. In this article, we focus on *relational* information integration, this is to say, we assume that the sources are databases over (different) relational schemas, called *source* or *local* schemas, and also that the unified format is some other relational schema, called the *target* or the *global* schema. A relational schema or simply a schema consists of names of relations and names of the columns of each relation. A database instance or simply an instance for a given schema is a collection containing, for each relation name in the schema, a finite relation (i.e., a table of records). An example of a source schema and a target schema is given in Figure 1. The source schema consists of three relation names that contain information about direct orders from a manufacturer together with information about retail sales; the target schema consists of a single relation name intended to summarize the sales records. Figure 1 also depicts a source instance and three target instances that will be used later on to illustrate the main concepts.

Two important facets of information integration are data exchange and data integration. Both these facets deal with the attainment of information integration, but they adopt distinctly different approaches. Data exchange is the problem of transforming data residing in different sources into data structured under a target schema; in particular, data exchange entails the materialization of data, after the data have been extracted from the sources and restructured into the unified format. In contrast, data integration can be described as symbolic or virtual integration: users are provided with the capability to pose queries and obtain answers via the unified format interface, while the data remain in the sources and no materialization of the restructured data takes place. Figure 2 depicts the data integration and data-exchange tasks.

In both data exchange and data integration, the relationship between the local schemas and the global schema must be spelled out. One way to accomplish this is via programs or SQL scripts written by human experts; this, however, can be an expensive and error-prone undertaking due to the complexity of the transformations involved. Instead, the research community has introduced *schema mappings*, a higher level

A previous version of this article appeared in the *Proceedings of the 12th International Conference on Database Theory*, 2009.

Figure 1. An example of a schema mapping.

Source database schema S:

DirectCustomer(cust-id, name, address)
DirectOrder(cust-id, date, prod, quant)
Retail(store-id, date, prod, quant)

Target database schema T:

Sales(date, cust, prod, quant)

Source database instance I:

DirectCustomer

cust-id	name	address
c1	UCSC	1156 High St, Santa Cruz, CA 95060

DirectOrder

cust-id	date	prod	quant
c1	05-01-2009	Quadcore-9950-PC	100
c1	05-01-2009	TFT-933SN-Wide	100

Retail

store-id	date	prod	quant
s1	05-03-2009	Quadcore-9950-PC	1

A target database instance J_1

Sales

date	cust	prod	quant
05-01-2009	UCSC	Quadcore-9950-PC	100
05-01-2009	UCSC	TFT-933SN-Wide	100
05-03-2009	N_1	Quadcore-9950-PC	1

A second target database instance J_2

Sales

date	cust	prod	quant
05-01-2009	UCSC	Quadcore-9950-PC	100
05-01-2009	UCSC	TFT-933SN-Wide	100
05-03-2009	UCLA	Quadcore-9950-PC	1

A third target database instance J_3

Sales

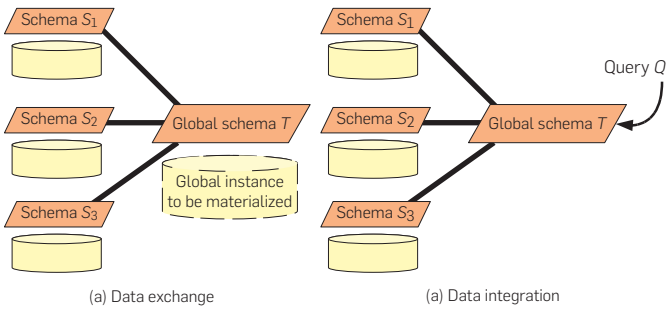
date	cust	prod	quant
05-01-2009	UCSC	Quadcore-9950-PC	100
05-03-2009	N_1	Quadcore-9950-PC	1

Schema mapping

$\forall x,y,z,u,v,w$ (DirectCustomer$(x,y,z) \wedge$ DirectOrder(x,u,v,w)
$\rightarrow$ Sales(u, y, v, w))

$\forall x,y,z,v,w$ (Retail$(x, y, v, w) \rightarrow \exists N$ Sales(y, N, v, w))

Figure 2. Data exchange and data integration.

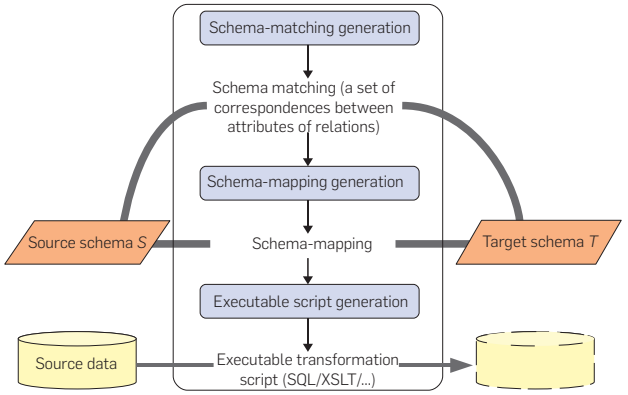


of abstraction that makes it possible to separate the specification of the relationship between the schemas from the actual implementation of the transformations. Schema mappings are declarative specifications that describe the relationship between two database schemas. In recent years, schema mappings have been used extensively in specifying data interoperability tasks and are regarded as the essential building blocks in data exchange and data integration (see, e.g., the surveys[14, 15]). The use of schema mappings helps the user understand and reason about the relationship between the source schemas and the target schema; furthermore, schema mappings can be automatically compiled into executable scripts in various languages.

A concrete example of a schema mapping is given in Figure 1. The schema mapping is specified by two sentences of first-order logic. The first sentence asserts that whenever the source relation DirectCustomer contains a triple (v_1, v_2, v_3) of values and the source relation DirectOrder contains a quadruple (v_1, v_4, v_5, v_6) of values so that the values for cust-id in these two tuples coincide, then the target relation Sales must contain the quadruple (v_1, v_2, v_4, v_5). The second sentence asserts that whenever the source relation Retail contains a quadruple (w_1, w_2, w_3, w_4) of values, then there must exist some value V so that the target relation Sales contains the quadruple (w_2, V, w_3, w_4). Clearly, the pair (I, J_1) consisting of the source relation I and the target relation J_1 satisfies both these formulas; the same holds true for the pair (I, J_2). In contrast, the pair (I, J_3) fails to satisfy the first formula because the Sales relation does not contain the required quadruple (05-01-2009, UCSC, TFT-933SN-Wide, 100).

This example also unveils some of the main conceptual and algorithmic issues arising in data exchange and data integration. On the data exchange side, suppose that we wish to transform the above source instance I to a target instance J according to the schema mapping in Figure 1. There are at least two distinct target instances that, together the source instance I, satisfy the specification of the schema mapping; in fact, it is easy to see that there are *infinitely* many such target instances. So, which one should we choose to materialize? What makes one target instance a "better" candidate to materialize than another, and how can one be computed? As we shall see, *universal solutions* turn out to be the preferred target instances to materialize. On the data integration side, suppose that a user poses a query over the target schema. Different answers may be obtained by evaluating the query on different target instances that (together with the given source instance) satisfy the schema-mapping specification. So, what is the "right" semantics of target queries in data integration? Is it possible to rewrite target queries into queries over the source schema so that they can be evaluated directly against the given source instance? This will lead us to the notions of the *certain answers* and of *allowing for query rewriting*. In the next section, we will introduce some of the most commonly used schema-mapping languages, including global-as-view (GAV) dependencies and local-as-view (LAV) dependencies, and we will find out that schema mappings specified in these formalisms indeed admit universal solutions and allow for rewriting of the most frequently asked

Figure 3. Architecture of the Clio data-exchange system.



queries in relational databases (see Figure 2).

A system that makes systematic use of schema mappings is Clio, a data-exchange system that started as a research prototype at the IBM Almaden Research Center and is now part of IBM's suite of information integration tools.[13] The architecture of Clio is depicted in Figure 3. The system has a *schema-matching* component, a *schema-mapping generation* component, and an *executable-script generation* component. The schema-matching component produces a set of correspondences between attributes of relations in a source schema and a target schema; these correspondences are derived automatically or semi-automatically through an interaction with the user and via a graphical user interface that allows the user to intervene and make changes. The schema-mapping generation component takes as input these attribute correspondences and returns a schema mapping. In general, there are more than one schema mapping that are consistent with a set of attribute correspondences. Clio produces just one of these possible schema mappings but the user can again intervene and edit the schema mapping returned by the system. Finally, the executable-script generation component automatically transforms this schema mapping into a set of scripts in some language, such as SQL or XSLT.

2. SCHEMA MAPPINGS AND LANGUAGES

In this section, we define the basic notions about schema mappings and present some of the main schema-mapping languages studied by the research community.

2.1. Basic notions

A *(relational) schema* is a tuple $\mathbf{R} = (R_1, \ldots, R_n)$ of relation symbols each of which has a fixed arity (number of attributes). An **R**-*instance* is a tuple $I = (R_1^I, \ldots, R_n^I)$ of finite relations, whose arities match those of the relation symbols of **R**. A *fact* of I is an expression $R_i\mathbf{a}$, where $i \le n$ and **a** is a tuple of values belonging to the relation R_i^I. The *active domain* of I, denoted by *adom* (I), is the set of all values occurring in the relation R_i^I, for $1 \le i \le n$. We will be usually working with two disjoint schemas, called the *source schema* $\mathbf{S} = (S_1, \ldots, S_n)$ and the *target schema* $\mathbf{T} = (T_1, \ldots, T_m)$. An **S**-instance is called a *source instance* and a **T**-instance is called a *target instance*. Whenever we consider a pair of instances (I, J), it is understood that I is a source instance and J is a target instance.

As stated earlier, a schema mapping is a declarative specification that describes the relationship between two schemas. More precisely, a *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where **S** is a source schema, **T** is a target schema disjoint from **S**, and Σ is a set of sentences (i.e., formulas with no free variables) in some logical formalism. This is the *syntactic* view of schema mappings. There is also a complementary *semantic* view of schema mappings that we present next. Let I be a source instance and J a target instance. We say that J is a *solution for I w.r.t. a schema mapping* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ if $(I, J) \models \Sigma$, which means that (I, J) satisfies every sentence in Σ. Consequently, from a semantic standpoint, a schema mapping $\mathcal{M}$ can be thought of as a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \mathcal{W})$, where $\mathcal{W}$ is the set of all pairs (I, J) such that J is a solution for I w.r.t. $\mathcal{M}$. So, as a semantic object, a schema mapping $\mathcal{M}$ is triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \mathcal{W})$, where **S** is a source schema, **T** is a target schema disjoint from **S**, and $\mathcal{W}$ is a collection of pairs (I, J) with I a source instance and J a target instance.

Let L be a logical language, let Σ be a set of L-sentences, and let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \mathcal{W})$ be a schema mapping given as a semantic object. We say that $\mathcal{M}$ is *L-definable by* Σ if for every source instance I and every target instance J, we have that $(I, J) \in \mathcal{W}$ if and only if $(I, J) \models \Sigma$. When we work with schema mappings in the sequel, it will be clear from the context if the schema mapping at hand is viewed as a syntactic object or as a semantic one.

EXAMPLE 2.1. To illustrate these notions, consider the schema mapping in Figure 1. In this example, the source schema **S** consists of the relations DirectCustomer, DirectOrder, and Retail, while the target schema **T** consists of the single relation Sales. The relationship between source and target is then described by the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ consists of the two first-order sentences listed in Figure 1. As a semantic object, $\mathcal{M}$ is the triple $(\mathbf{S}, \mathbf{T}, \mathcal{W})$, where $\mathcal{W}$ consists of all pairs (I, J) satisfies the two sentences in Σ. In particular, the pairs (I, J_1) and (I, J_2) belong to $\mathcal{W}$, but the pair (I, J_3) does not. In other words, J_1 and J_2 are solutions for I w.r.t. to $\mathcal{M}$, but J_3 is not.

2.2. Schema-mapping languages

What is a "good" language for specifying schema mappings? To address this question, let us reflect on the relational database model, introduced by E.F. Codd 40 years ago.[4] One of the reasons for the success of this model is that it supports powerful, high-level database query languages, such as *relational algebra* and *relational calculus*, that have formed the foundation for SQL. Relational algebra and relational calculus have the same expressive power as first-order logic[5]; in fact, relational calculus is a syntactic variant of first-order logic tailored for databases. Thus, at first sight, it is natural to ask: can first-order logic also be used as a schema-mapping language? It is not hard to show, however, that basic algorithmic problems in data integration and data exchange, such as existence-of-solutions and

query answering, become *undecidable* if unrestricted use of first-order logic is allowed in specifying the relationship between database schemas (intuitively, this is so because such tasks involve testing first-order sentences for satisfiability, since they quantify over all solutions of a source instance). Consequently, we have to identify proper sublanguages of first-order logic that strike a good balance between expressive power for data interoperability purposes and algorithmic properties. Towards this goal, let us consider the following basic tasks that every schema-mapping language ought to support.

- **Copy (Nicknaming):** Copy a source relation into a target relation and rename it.
- **Projection (Column Deletion):** Form a target relation by deleting one or more columns of a source relation.
- **Augmentation (Column Addition):** Form a target relation by adding one or more columns to a source relation.
- **Decomposition:** Decompose a source relation into two or more target relations.
- **Join:** Form a target relation by joining two or more source relations.
- **Combinations of the above:** For example, combine join with column augmentation.

These tasks can be easily specified in first-order logic, as shown next. For concreteness, we use relations of arities two or three.

Copy	$\forall x, y, z(P(x, y, z) \rightarrow T(x, y, z))$
Projection	$\forall x, y, z(P(x, y, z) \rightarrow U(x, y))$
Augmentation	$\forall x, y(R(x, y) \rightarrow \exists z T(x, y, z))$
Decomposition	$\forall x, y, z(P(x, y, z) \rightarrow U(x, y) \wedge V(y, z))$
Joint	$\forall x, y, z(R(x, z) \wedge S(z, y) \rightarrow T(x, y, z))$
Combinations of the above	$\forall x, y, z(R(x, z) \wedge S(z, y) \rightarrow \exists t(U(x, t) \wedge W(x, y, z, t)))$

Observe that the above formulas have a striking syntactic similarity. As a matter of fact, they all belong to a class of first-order formulas called *source-to-target tuple generating dependencies* that we define in what follows.

- If $\mathbf{R} = (R_1, \ldots, R_n)$ is a relational schema, then an *atomic formula over* $\mathbf{R}$ is an expression of the form $R_i(\mathbf{x})$, where $i \leq n$ and $\mathbf{x}$ is a tuple of variables of length equal to the arity of R_i.
- A *tuple-generating dependency* (*tgd*) is a first-order formula of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})),$$

  where $\mathbf{x}$ and $\mathbf{y}$ are tuples of variables, $\phi(\mathbf{x})$ is a conjunction of atomic formulas with variables in $\mathbf{x}$, each variable in $\mathbf{x}$ occurs in at least one conjunct of $\phi(\mathbf{x})$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas with variables among those in $\mathbf{x}$ and $\mathbf{y}$.
  A *full tgd* is a tgd with no existential quantifiers in the right-hand side, i.e., it is of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$.
- A *source-to-target tuple-generating dependency* (*s-t tgd*) is a tgd such that $\phi(\mathbf{x})$ is a conjunction of atomic formulas over a source schema $\mathbf{S}$ and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over a target schema $\mathbf{T}$.

Informally, s-t tgds assert that if a pattern of facts appears in the source, then another pattern of facts must appear in the target. They are also known as *global-and-local-as-view* (*GLAV*) dependencies. In recent years, s-t tgds have been studied extensively in the context of data exchange and data integration[14, 15] because, in spite of their syntactic simplicity, they can express many data interoperability tasks arising in applications. Furthermore, s-t tgds have desirable structural properties that we will discuss in the next section. The following two types of dependencies are important special cases of s-t tgds.

- A *GAV* dependency is an s-t tgd in which the right-hand side of the implication consists of a single atomic formula. Thus, a GAV dependency is of the form

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow U(\mathbf{x}'))$$

  with $\phi(\mathbf{x})$ a conjunction of atomic formulas over a source schema and $U(\mathbf{x}')$ an atomic formula over a target schema such that the variables in $\mathbf{x}'$ are among those in $\mathbf{x}$.
- A *LAV* dependency is an s-t tgd in which the left-hand side of the implication is a single atomic formula. Thus, a LAV dependency is of the form

$$\forall \mathbf{x}(R(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$$

  with $R(\mathbf{x})$ an atomic formula over a source schema and $\psi(\mathbf{x}, \mathbf{y})$ a conjunction of atomic formulas over a target schema.

Consider again the schema mapping in Figure 1. The first sentence used to specify that schema mapping is a GAV dependency, while the second one is a LAV dependency. Note also that the expressions for the Copy and the Projection tasks are both GAV and LAV dependencies, the expressions for the Augmentation and the Decomposition tasks are LAV dependencies, and the expression for the Join task is a GAV dependency. The expression for the Decomposition task is a full tgd. It is easy to see that every full tgd is logically equivalent to finitely many GAV dependencies. For example, the full tgd $\forall x, y, z(P(x, y, z) \rightarrow U(x, y) \wedge V(y, z))$ for the Decomposition task is logically equivalent to the set consisting of the GAV dependencies $\forall x, y, z(P(x, y, z) \rightarrow U(x, y))$ and $\forall x, y, z(P(x, y, z) \rightarrow V(y, z))$.

Before being used in data exchange and data integration, tuple-generating dependencies had been investigated in depth in the context of *dependency theory* during the 1970s and the 1980s. Dependency theory is the study of integrity constraints in databases; in this context,

tuple-generating dependencies possess desirable algorithmic properties and contain as special cases many well-known classes of integrity constraints in databases, such as inclusion dependencies, join dependencies, and multi-valued dependencies.[10] It is also interesting to note that tuple-generating dependencies seemed to have first appeared (at least in implicit form) a very long time ago. Indeed, in a recent article[2] presenting a formal system for Euclid's Elements, the authors argue convincingly that the theorems in the Elements can be expressed using tuple-generating dependencies! Intuitively, this is so because the typical theorem of Euclidean Geometry states that if a certain pattern between geometric objects (points, lines, triangles, or circles) exists, then another pattern between geometric objects must also exist.

3. PROPERTIES OF SCHEMA MAPPINGS

In this section, we present several structural properties of schema mappings, which have been widely used in the literature on data exchange and data integration. They will turn out to play a key role in our characterizations of schema-mapping languages. Before presenting these properties, however, we need to introduce some important notions in data exchange and data integration.

Homomorphisms. A central notion in the study of schema mappings is that of a *homomorphism*. Let K and K' be two instances over the same schema $\mathbf{R} = (R_1, \ldots, R_n)$. A homomorphism for K to K' is a function from the active domain of K to the active domain of K' such that if $(a_1, \ldots, a_m) \in R_i^k$, then $(h(a_1, \ldots, a_m)) \in R_i^{k'}$, for $i = 1, \ldots, n$.

A homomorphism h is said to be *constant* on a set X if h restricted to $X \cap dom\ (h)$ is the *identity* function, i.e., $h(x) = x$, if $x \in X$ and $h(x)$ is defined. Here, we will consider homomorphisms between target instances and we will require that they are constant on the active domain of some source instance. The rationale behind this requirement is as follows. Typically, when data is exchanged from source to target, the elements in the active domain of a given source instance are "known" values. The schema mapping at hand, however, may under-specify the relationship between source and target. In turn, this may force using "unknown" values, called *labeled nulls*, to materialize solutions for a given source instance. Thus, target instances may have both "known" values, originating from the source instance, and "unknown" values, chosen freshly and acting as placeholders. An example of labeled null is the value N_1 in the target instance J_1 in Figure 1. Homomorphisms are required to leave "known" values untouched, but are free to replace an "unknown" value by another, "known" or "unknown", value.

Universal Solutions. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping. Recall that if I is a source instance, then a solution for I w.r.t. $\mathcal{M}$ is a target instance J such that $(I, J) \models \Sigma$. In general, a source instance I may have multiple solutions and, in fact, infinitely many; in particular, this holds true for schema mappings $\mathcal{M}$ specified by s-t tgds because if J is a solution for I w.r.t. $\mathcal{M}$, then every instance J' containing J (as a set of facts) is also a solution for I. Which among those solutions should one materialize if I is to be transformed into a target instance? To address this question, the following concept of a *universal* solution was introduced in Fagin et al.[7]

Let $\mathcal{M} = (\mathrm{S}, \mathrm{T}, \mathcal{W})$ be a schema mapping and let I be a source instance. A target instance J is a *universal solution for I* if

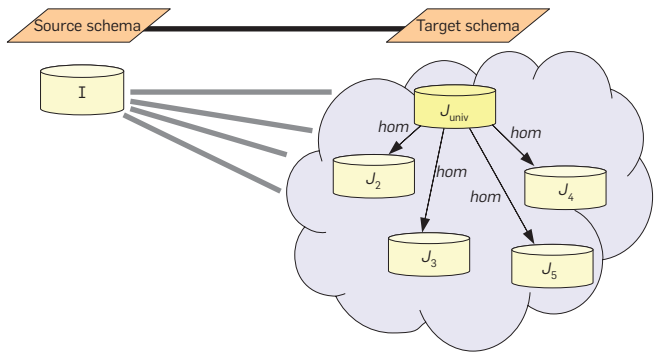
1. J is a solution for I.
2. For each target instance J' that is a solution for I, there is a homomorphism $h: J \to J'$ that is constant on adom (I).

The intuition behind this concept, which is illustrated in Figure 4, is that a universal solutions is a "most general" solution in the sense that it contains no more and no less information than that specified by the given schema mapping.

Example 3.1. Consider the source instance I in Figure 1. One solution for I with respect to the given schema mapping is the target instance J_1. Note that N_1 is a value that does not occur in I, and therefore is interpreted as a labeled null. Another solution for I is the target instance J_2, which is the same as J_1 except that labeled null value N_1 has been replaced by UCLA. However, J_2 contains information that is not implied by the schema mapping, namely, that the customer of the order on May 3, 2009 is UCLA, and, consequently, it is not a universal solution. Indeed, there is a homomorphism from J_1 to J_2 constant on the active domain of I (N_1 is mapped to UCLA), but not vice versa.

Queries and Certain Answers. Informally, a *query* is a question that a user poses against a database. More formally, a *query* q takes as input an instance K and returns as output a relation $q(K)$ of fixed arity with values from the active domain of K. Suppose now that we have a schema mapping between a source schema and target schema. Suppose also that a query q over the target schema is posed and that a source instance I is given. What does answering q using the source instance I mean? As seen earlier, there may be infinitely many solutions J for a given source instance I; furthermore, if q is evaluated on different solutions J for I, it is

Figure 4. A universal solution.



possible that different answers are obtained each time. This ambiguity raises the conceptual problem of giving precise semantics to query answering in data exchange and data integration. The approach taken by the research community is to adopt the *certain answers* semantics, a semantics that originated in the study of incomplete databases (see van der Meyden[19]).

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and q a query over the target schema $\mathbf{T}$. If I is source instance, then *the certain answers of q on I with respect to $\mathcal{M}$*, denoted $\text{certain}_{\mathcal{M}}(q)(I)$, is the set

$$\text{certain}_{\mathcal{M}}(q)(I) = \cap\{q(J) : J \text{ is a solution for } I \text{ w.r.t. } \mathcal{M}\}.$$

The certain answers semantics provides the guarantee that if a tuple $\mathbf{t}$ belongs to $\text{certain}_{\mathcal{M}}(q)(I)$, then $\mathbf{t}$ belongs to the result $q(J)$ of q on *every* solution J for I. It is easy to see that every tuple in $\text{certain}_{\mathcal{M}}(q)(I)$ is a tuple of values from I. Thus, every schema mapping $\mathcal{M}$ induces a (semantic) transformation $\text{certain}_{\mathcal{M}}$ from queries over the target schema to queries over the source schema, so that if q is query over the target schema, then this transformation associates to it the query $\text{certain}_{\mathcal{M}}(q)$ over the source schema that has the same arity as q and is defined by $\text{certain}_{\mathcal{M}}(q)(I) = \cap\{q(J) : J \text{ is a solution for } I \text{ w.r.t. } \mathcal{M}\}$.

On the face of it, the certain answers semantics is non-effective, since evaluating $\text{certain}_{\mathcal{M}}(q)(I)$ may entail computing the intersection of infinitely many sets. For many frequently asked queries, however, efficient algorithms for evaluating their certain answers exist.

A conjunctive query is a first-order formula of the form $\exists\mathbf{w}\chi(\mathbf{x}, \mathbf{w})$, where $\chi(\mathbf{x}, \mathbf{w})$ is a conjunction of atoms and/or equalities. Conjunctive queries are the most frequently asked queries in relational databases. They are also known as *project select-join* (SPJ) queries because they are precisely the queries that are expressible in relational algebra using the selection, projection, and join operations; in particular, conjunctive queries are easily expressible in SQL using the SELECT FROM WHERE construct. A *union of conjunctive queries* is a finite disjunction of conjunctive queries; equivalently, they are the queries expressible in relational algebra using the selection, projection, join, and union operations.

In information integration, the main approach to computing the certain answers is to try to rewrite queries over the target to queries over the source. In data exchange, one would like to take advantage of the materialized solution and use it to obtain the certain answers of target queries. Both approaches give rise to polynomial-time algorithms for computing the certain answers of unions of conjunctive queries. In particular, as shown in Fagin et al.,[7] the certain answers of unions of conjunctive queries can be obtained using universal solutions. Specifically, let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping such that Σ is a finite set of s-t tgds and let q be a union of conjunctive queries over $\mathbf{T}$. If I is a source instance and J is a universal solution for I, then $\text{certain}_{\mathcal{M}}(q)(I) = q(J)_{\downarrow}$, where $q(J)_{\downarrow}$ is the result obtained by first computing $q(J)$ and then keeping only those tuples that contain values from the active domain of I only.

EXAMPLE 3.2. Returning to the example schema mapping $\mathcal{M}$ and source instance I from Figure 1, consider the conjunctive query q over the target schema given by

$$q(x, y) = \exists name, date, n, m(Sales(name, date, x, n) \wedge Sales(name, date, y, m))$$

It asks for all pairs of products (x, y), such that some customer bought x and y (in some quantities) on the same date. It is not hard to see that, for every solution J of I with respect to $\mathcal{M}$, the pair (Quadcore-9950-PC,TFT-933SN-Wide) belongs to $q(J)$. In other words, this tuple belongs to the certain answers of q in I with respect to $\mathcal{M}$. It turns out that the certain answers of q on a source instance I are precisely the answers to $q'(I)$, where q' is the following union of conjunctive queries over the source schema:

$$\begin{aligned} q'(x,y) = (&\exists cid_1, cid_2, name, addr_1, addr_2, date, n, m \\ &(DirectOrder(cid_1, date, x, n) \wedge DirectOrder(cid_2, date, y, m) \wedge \\ &DirectCustomer(cid_1, name, addr_1) \wedge \\ &DirectCustomer(cid_2, name, addr_2))) \\ &\vee (x = y \wedge \exists sid, date, n\, Retail(sid, date, x, n)) \end{aligned}$$

Note that the *Retail* relation does not provide information about the name of the buyer, and therefore, can only contribute identity pairs to the certain answers of q.

Alternatively, the certain answers of q on I can be computed by evaluating q on the universal solution J_1 of I that we discussed in Example 3.1, and keeping only those tuples that contain only values from the active domain of I.

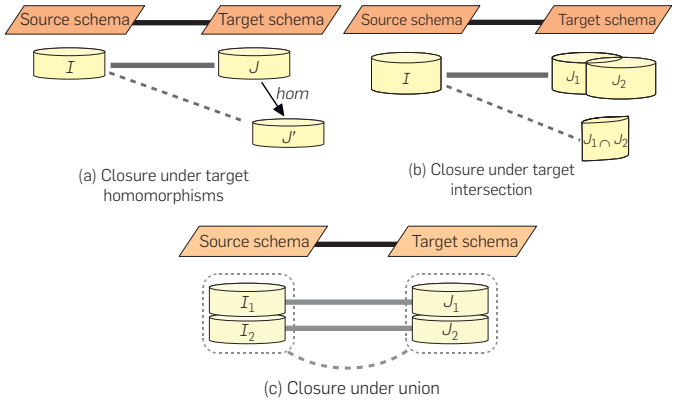
Structural Properties of Schema Mappings. Recall that a schema mapping $\mathcal{M}$ can be viewed as a syntactic object or as a semantic one. As a syntactic object, $\mathcal{M}$ is given by a triple $(\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a set of sentences in some logical formalism; as a semantic object, $\mathcal{M}$ is given by a triple $(\mathbf{S}, \mathbf{T}, \mathcal{W})$, where $\mathcal{W}$ is a set of pairs (I, J) with I a source instance and J a target instance. In what follows, whenever we write that $(I, J) \in \mathcal{M}$, we mean that $(I, J) \models \Sigma$ or that $(I, J) \in \mathcal{W}$ depending on whether $\mathcal{M}$ is given as a syntactic object or a semantic one.

We are now ready to present the structural properties of schema mappings that will play a key role in our characterization results. We begin with three such properties that have been widely used in both data exchange and data integration.

DEFINITION 3.3. *Let $\mathcal{M}$ be a schema mapping.*

- **Closure under target homomorphisms**: *We say that* $\mathcal{M}$ is closed under target homomorphisms *if for all* $(I, J) \in \mathcal{M}$ *and for all homomorphisms* $h: J \to J'$ *that are constant on* $adom(I)$, *we have that* $(I, J') \in \mathcal{M}$. (*see Figure 5a*).
- **Admitting universal solutions**: *We say that* $\mathcal{M}$ admits universal solutions *if for each source instance I there is a universal solution J for I w.r.t.* $\mathcal{M}$.
- **Allowing for conjunctive query rewriting**: *We say that* $\mathcal{M}$ allows for conjunctive query rewriting *if for each union q of conjunctive queries over the target schema, the*

Figure 5. Closure properties of schema mappings.



certain answers query $\text{certain}_{\mathcal{M}}(q)$ *is definable by a union of conjunctive queries over the source schema. In other words, there is a union q′ of conjunctive queries over the source schema such that* $\text{certain}_{\mathcal{M}}(q)(I) = q'(I)$, *for every source instance I.*

The first two conditions of closure under target homomorphisms and admitting universal solutions go very well together. As was observed in Fagin et al.,[7] if a schema mapping is closed under target homomorphisms and admits universal solutions, then, for every source instance I, the (typically infinite) space of all solutions of I w.r.t. $\mathcal{M}$ can be completely described by just a single target instance J, namely, by any universal solution J for I. This is so because if J is universal for I and $\mathcal{M}$ is closed under target homomorphisms, then for every target instance J', we have that J' is a solution for I if and only if there is a homomorphism $h: J \to J'$ that is constant on $adom(I)$. We mention in passing that these two conditions together also imply the existence of *core* universal solutions, which are the smallest universal solutions (see Fagin et al.[8]). Thus, these two conditions lie at the foundation of *data exchange*. The third condition of allowing for conjunctive query rewriting is important in the context of *data integration*, since it implies that the certain answers of unions of conjunctive queries over the target are computable in polynomial time (in the sense of data complexity).

It is well known that all three conditions of closure under target homomorphisms, admitting universal solutions, and allowing for conjunctive query rewriting are possessed by every schema mapping $\mathcal{M}$ definable by a finite set of s-t tgds. Closure under homomorphisms follows easily from the definitions; admitting universal solutions was shown in Fagin et al.[7] using the *chase procedure*. In the case of GAV dependencies, a union of conjunctive queries over the target is easily transformed to a union of conjunctive queries over the source by simply replacing each target relation symbol P by a union of conjunctive queries over the source that defines P. In the case of arbitrary s-t tgds, allowing for conjunctive query rewriting is proved by first "decomposing" the given s-t tgds into GAV dependencies and to LAV dependencies, and then applying results from Abiteboul and Duschka[1] and Duschka and Genesereth.[6] We collect these results into one theorem.

THEOREM 3.4. *Every schema mapping definable by a finite set of s-t tgds is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting.*

Next, we define three additional properties of schema mappings.

DEFINITION 3.5. *Let $\mathcal{M}$ be a schema mapping.*

- **Closure Under Target Intersection**: *We say that* $\mathcal{M}$ is closed under target intersection *if for all source instances* I *and all target instances* J_1, J_2 *if* $(I, J_1) \in \mathcal{M}$ and $(I, J_2) \in \mathcal{M}$, *then also* $(I, J_1 \cap J_2) \in \mathcal{M}$. (*See Figure 5b.*)
- **Closure Under Union**: *We say that* $\mathcal{M}$ *is* closed under union *if* $(\emptyset, \emptyset) \in \mathcal{M}$, *and for all* $(I_1, J_1) \in \mathcal{M}$, *and* $(I_2, J_2) \in \mathcal{M}$, *we have that also* $(I_1 \cup I_2, J_1 \cup J_2) \in \mathcal{M}$. (*See Figure 5c.*)
- ***n*-Modularity**: *Let n be a positive integer. We say that* $\mathcal{M}$ *is* n-modular *if whenever a pair* (I, J) *does not belong to* $\mathcal{M}$, *there is a sub-instance* $I' \subseteq I$ *such that* $|adom(I')| \leq n$ *and* (I', J) *does not belong to* $\mathcal{M}$.

Intuitively, a schema mapping is closed under union if solutions can be constructed in a "modular" fashion, i.e., on a tuple-by-tuple basis. Similarly, n-modularity asserts that if $(I, J) \notin \mathcal{M}$, then there is a concise explanation for this fact; this property can also be viewed as a relaxation of closure under union.

We now give several useful propositions about the properties we just introduced.

PROPOSITION 3.6. *Let $\mathcal{M}$ be a schema mapping.*

- *If* $\mathcal{M}$ *is definable by a finite set of GAV dependencies, then* $\mathcal{M}$ *is closed under target intersection.*
- *If* $\mathcal{M}$ *is definable by a finite set of LAV dependencies, then* $\mathcal{M}$ *is closed under union.*
- *If* $\mathcal{M}$ *is definable by a finite set of s-t tgds, then* $\mathcal{M}$ *is n-modular for some* $n \geq 1$.

PROOF. The first two parts follow easily from the definitions. For the third part, assume that $\mathcal{M}$ is a schema mapping definable by a finite set Σ of s-t tgds. Let n be the maximum number of variables occurring in the left-hand side of the s-t tgds in Σ. We claim that $\mathcal{M}$ is n-modular. Assume that $(I, J) \notin \mathcal{M}$. Then there is some s-t tgd $\forall \mathbf{x}(\phi(\mathbf{x}) \to \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ from Σ and a tuple $\mathbf{a}$ of values from $adom(I)$ such that $(I, J) \models \phi(\mathbf{a}) \wedge \neg\exists \mathbf{y}\psi(\mathbf{a}, \mathbf{y})$. Now, let I' be the sub-instance of I containing only the values $\mathbf{a}$. Then, it is still the case that $(I', J) \models \phi(\mathbf{a}) \wedge \neg\exists \mathbf{y}\psi(\mathbf{a}, \mathbf{y})$, and hence $(I', J) \notin \mathcal{M}$. ❑

4. LANGUAGE CHARACTERIZATIONS

This section contains the main technical results of the paper, which yield structural characterizations of the various schema-mapping languages considered in earlier sections. We begin with schema mappings specified by LAV

dependencies.

THEOREM 4.1. *For all schema mappings $\mathcal{M}$, the following are equivalent:*

1. *$\mathcal{M}$ is definable by a finite set of LAV dependencies.*
2. *$\mathcal{M}$ is closed under target homomorphisms, admits universal solutions, allows for conjunctive query rewriting, and is closed under union.*

PROOF. The implication (1) ⇒ (2) follows from Theorem 3.4 and Proposition 3.6. We now prove the implication (2) ⇒ (1). The idea behind the proof is that, since $\mathcal{M}$ is closed under union, universal solutions for source instances I can be constructed out of universal solutions for parts of I. This implies that, in defining our schema mapping, we only need to take into account of finite number of source instances up to isomorphism, namely, those that contain precisely one tuple. In what follows, we will make this idea precise.

Suppose that $\mathcal{M}$ satisfies the listed conditions. Let $R_1, \ldots, R_n$ be the relations of the source schema, and let D be a set consisting of k distinct values, with $k = \max_{i \le n} \mathsf{arity}(R_i)$. Let facts be the set of all possible facts, of the form $R_i(d_1, \ldots, d_l)$ with $i \le n$, $l = \mathsf{arity}(R_i)$, and $d_1, \ldots, d_l \in D$. For each $\alpha \in$ facts, let I_α be a the source instance containing only the fact α, and let J_α be a universal solution for I_α. Let $PosDiag_{I_\alpha}(\mathbf{x})$ be the *positive diagram* of I_α, i.e., the conjunction of all facts true in I (which consists of precisely one fact) and let $PosDiag_{J_\alpha}(\mathbf{x}, \mathbf{y})$ be the positive diagram of J_α where $\mathbf{x}$ are as many variables as there are elements of $adom(I_\alpha)$ and $\mathbf{y}$ as many variables as there are elements of $adom(J_\alpha) \setminus adom(I_\alpha)$. Let ϕ_α be the following LAV dependency: $\forall \mathbf{x}(PosDiag_{I_\alpha}(\mathbf{x}) \rightarrow \exists \mathbf{y} PosDiag_{J_\alpha}(\mathbf{x}, \mathbf{y}))$. Finally, let $\Sigma = \{\phi_\alpha | \alpha \in \mathsf{facts}\}$. We claim that Σ defines $\mathcal{M}$.

First, we prove *soundness*: every $(I, J) \in \mathcal{M}$ satisfies Σ. Suppose $(I, J) \in \mathcal{M}$, and take any $\phi_\alpha \in \Sigma$. Furthermore, suppose that the antecedent of ϕ_α is satisfied in (I, J) under some variable assignment h. In other words, h is a homomorphism from I_α to I. Let q be the certain answer query of the conjunctive query $\exists \mathbf{y} PosDiag_{J_\alpha}(\mathbf{x}, \mathbf{y})$. Since $\mathcal{M}$ allows for conjunctive query rewriting, q is definable by a union of conjunctive queries. Moreover, since $\exists \mathbf{y} PosDiag_{J_\alpha}(\mathbf{x}, \mathbf{y})$ is satisfied in J_α, which is a universal solution of I_α, it is satisfied in all solutions of I_α. In other words, q is satisfied in I_α, and hence, in I under the assignment h. Hence, (I, J) satisfies ϕ_α.

Next, we prove *completeness*: every pair (I, J) satisfying Σ belongs to $\mathcal{M}$. Suppose (I, J) satisfies Σ. Write I as $I = I_1 \cup \cdots \cup I_n$ where each I_i contains only a single fact. Then each (I_i, J) still satisfies Σ. Since I_i contains a single fact, it must be isomorphic to I_α for some $\alpha \in$ facts. Using the fact that (I_i, J) satisfies ϕ_α, we can show that there is a homomorphism from a universal solution of I_i to J, constant on $adom(I_i)$, hence, by closure under target homomorphisms, $(I_i, J) \in \mathcal{M}$. It follows by closure under union that $(I, J) \in \mathcal{M}$. ❑

Our next result characterizes schema mappings specified by GAV dependencies.

THEOREM 4.2. *For all schema mappings $\mathcal{M}$, the following are equivalent*:

1. *$\mathcal{M}$ is definable by a finite set of GAV dependencies.*
2. *$\mathcal{M}$ is closed under target homomorphisms, admits universal solutions, allows for conjunctive query rewriting, and is closed under target intersection.*

PROOF. (Hint) The implication (1) ⇒ (2) follows from Theorem 3.4 and Proposition 3.6. For the implication (2) ⇒ (1), we first show that every schema mapping $\mathcal{M}$ satisfying (2) is n-modular for some $n > 0$. For each target relation R, let $q_R = \mathsf{certain}_{\mathcal{M}}(R\mathbf{y})$, where $\mathbf{y}$ is a sequence of distinct fresh variables of appropriate length. Note that, since $\mathcal{M}$ allows for conjunctive query rewriting, q_R can be written as a union of conjunctive queries. Now, let n be the maximum of the number of variables occurring in each q_R. Using the hypothesis that $\mathcal{M}$ admits universal solutions, is closed under target homomorphisms, and is closed under target intersection, it can be shown that $\mathcal{M}$ is n-modular.

After this, the implication (2) ⇒ (1) is established along the same lines as the proof of Theorem 4.1. Instead of considering all source instances consisting of one tuple, we consider all source instances I with $|adom(I)| \le n$. There are only finitely many such source instances up to isomorphism. Moreover, it can be shown, using closure under intersection, that each has a null-free universal solution, and hence only *full* s-t tgds are needed to describe them. ❑

We now focus on schema mappings specified by arbitrary s-t tgds. As seen in Theorem 3.4, every schema mapping defined by a *finite* set of s-t tgds is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting. The next result asserts that any schema mapping satisfying these conditions is definable by an *infinite* set of s-t tgds.

PROPOSITION 4.3. *If a schema mapping $\mathcal{M}$ is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting, then $\mathcal{M}$ is definable by an infinite set of s-t tgds.*

PROOF. (Hint) Assume that $\mathcal{M}$ satisfies the listed properties. Consider a source instance I and a target instance J such that J is a universal solution for I with respect to $\mathcal{M}$. For each element of $adom(I)$, introduce a distinct variable x_i, and for each element of $adom(J) \setminus adom(I)$, introduce a distinct variable y_j. Define $PosDiag_I(\mathbf{x})$ to be the conjunction of all atomic formulas in $\mathbf{x}$ true in I (under the chosen assignment) and define $PosDiag_J(\mathbf{x}, \mathbf{y})$ likewise. Finally, let Σ be the set of all s-t tgds $\phi_{I,J}$ of the form $\forall \mathbf{x}(PosDiag_I(\mathbf{x}) \rightarrow \exists \mathbf{y}\ PosDiag_J(\mathbf{x}, \mathbf{y}))$, where I is a source instance and J is a universal solution for I w.r.t. $\mathcal{M}$. Using an argument analogous to the one used in the proof of Theorem 4.1, it can be shown that Σ defines $\mathcal{M}$. ❑

Can Proposition 4.3 be strengthened to a characterization of schema mappings specified by a *finite* set of s-t tgds? The next result, which was proved in Fagin et al.,[9] shows that this is not possible.

PROPOSITION 4.4. *The schema mapping defined by the first-order sentence* $\forall x \exists y \forall z(Rxz \rightarrow Syz)$ *is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting, but is not definable by any finite set of s-t tgds.*

Can Proposition 4.3 be turned to a characterization of schema mappings specified by an *infinite* set of s-t tgds? The next observation shows that this is not possible.

PROPOSITION 4.5. *The schema mapping defined by the following infinite set of s-t tgds does not admit universal solutions:*

$$\{\forall x(Px \rightarrow \exists y_1 \cdots y_n(Rxy_1 \wedge Ry_1y_2 \wedge \cdots \wedge Ry_{n-1}y_n)) \mid n \geq 0\}$$

PROOF. (Hint) It is easy to see that no solution for $I = \{Pa\}$ can be universal. Here, the assumption that all instances (hence all solutions) are finite is of the essence. ❑

Proposition 4.4 implies that additional properties must be considered in order to characterize the schema mappings that are definable by a finite set of s-t tgds. It turns out that the addition of n-modularity, for some $n > 0$, yields such a characterization.

THEOREM 4.6. *For all schema mappings* $\mathcal{M}$, *the following are equivalent:*

1. $\mathcal{M}$ *is definable by a finite set of s-t tgds.*
2. $\mathcal{M}$ *is closed under target homomorphisms, admits universal solutions, allows for conjunctive query rewriting, and is n-modular for some* $n > 0$.

We conclude this section by commenting briefly on additional characterizations presented in the previous version of this paper.[17] Observe that the condition of *allowing for conjunctive query rewriting* differs in nature from the other structural conditions: while the latter are model-theoretic conditions, the former refers to a certain syntactically defined class of queries. Thus, it is natural to ask: can the condition of allowing for conjunctive query rewriting be replaced by a condition of model-theoretic character? To this effect, we consider the notion of *reflecting source homomorphisms*, which states, roughly, that every homomorphism between source instances I, I' extends to a homomorphism from any universal solution of I to any universal solution of I'. We show that the structural characterizations of LAV dependencies in Theorem 4.1 and of s-t tgds in Theorem 4.6 hold with the condition of allowing for conjunctive query rewriting replaced by that of reflecting source homomorphisms. Further, we pursue characterizations where one assumes that the schema mapping is first-order definable to start with. We establish that, for all schema mappings $\mathcal{M}$ definable by a first order sentence, $\mathcal{M}$ is definable by a finite set of GAV dependencies if and only if $\mathcal{M}$ is closed under target homomorphisms, admits universal solutions, reflects source homomorphisms, and is closed under target intersection. The proof makes essential use of the sophisticated machinery developed by Rossman[16] for proving the preservation-under-homomorphisms theorem in the finite.

5. COMPLEXITY OF DEFINABILITY

Our characterizations provide tools for testing whether a schema mapping defined in one language can also be defined in another language. For example, our results imply that a schema mapping defined by a finite set of s-t tgds is definable by a finite set of GAV dependencies if and only if it is closed under target intersection: furthermore, it is definable by a finite set of LAV dependencies if and only if it is closed under union. Here, we pinpoint the *computational complexity* of testing definability in the different languages.

First, assume that the input to the problem is a finite set of s-t tgds. The results are summarized in the following table.

Input schema mapping	Desired schema mapping	Complexity of definability
s-t tgds	GAV dependencies	NP-complete
s-t tgds	LAV dependencies	NP-complete
GAV dependencies	LAV dependencies	PTIME
LAV dependencies	GAV dependencies	NP-complete

The proofs also yield effective methods for constructing an equivalent schema mapping in the smaller language whenever it exists. Our proofs are based on reductions from definability problems to the *entailment problem for s-t tgds*; given two schema mappings $\mathcal{M}_1$, $\mathcal{M}_2$, specified by a finite set of s-t tgds, is it the case that whenever $(I, J) \in \mathcal{M}_1$ also $(I,J) \in \mathcal{M}_2$? We show that the latter problem in NP-complete; moreover, it is in PTIME if $\mathcal{M}_1$ is specified by LAV dependencies and $\mathcal{M}_2$ by GAV dependencies.

It is also natural to consider the problem of testing whether a schema mapping specified by a first-order sentence is definable in one of the schema-mapping languages studied here. This problem, however, turns out to be undecidable no matter what schema-mapping language we consider (s-t tgds, GAV dependencies, or LAV dependencies). This can be easily proved using the undecidability of satisfiability for first-order sentences in the finite.[18]

6. DISCUSSION AND OPEN PROBLEMS

The work presented here has methodological implications for the study of schema mappings. Concretely, our structural characterizations delineate the exact set of tools available in the study of schema mappings specified in particular languages. For example, consider the language of LAV dependencies. A perusal of the literature reveals that earlier results about schema mappings specified by a finite set of LAV dependencies made systematic use of the fact that such schema mappings are closed under target homomorphisms, admit universal solutions, allow for conjunctive query rewriting, and are closed under union. The structural characterization given in Theorem 4.1, in effect, turns the tables around and asserts that these

four properties are the *only* properties one needs to use in reasoning about schema mappings specified by LAV dependencies. On the computational side, the complexity-theoretic results in Section 5 quantify in precise terms the difficulty of determining whether a schema mapping specified in one language can also be specified in a different language.

There has also been an extensive study of schema mappings specified using languages richer than the language of s-t tgds. Consider, for instance, schema mappings specified by s-t tgds and target tgds, which were studied in Fagin et al.,[7] or schema mappings specified by second-order tgds (SO tgds), which arise when composing schema mappings specified by s-t tgds.[9] These languages are known to be strictly more expressive than the language of s-t tgds. Our results then predict that these languages lack at least one of the structural properties considered here. Indeed, schema mappings specified by s-t tgds and target tgds are closed under target homomorphisms and admit universal solutions, but need *not* allow for conjunctive query rewriting. Likewise, schema mappings specified by SO tgds admit universal solutions and allow for conjunctive query rewriting, but need *not* be closed under target homomorphisms. It remains an open problem to establish structural characterizations for such richer languages of dependencies. A particularly interesting question is whether there is a natural way to characterize *weakly acyclic* sets of target tgds,[7] a class of target dependencies that is of central importance in data exchange, as they guarantee termination of the chase procedure within a polynomial number of steps.

Among the properties of schema mappings considered here, closure under target homomorphisms, admitting universal solutions, and allowing for conjunctive query rewriting are arguably the most fundamental ones for data exchange and data integration. Proposition 4.4 tells that there are schema mappings satisfying these properties that cannot be defined by any finite set of s-t tgds. Is there a natural extension of the language of s-t tgds that is characterized by these three properties? In order to express transformations involving grouping and data merging, a proper extension of the language of s-t tgds, called *nested s-t tgds*, was proposed in Fuxman et al.[11] In the previous version of this paper,[17] we showed that schema mappings specified by a finite set of nested s-t tgds are closed under target homomorphisms, admit universal solutions, allow for conjunctive query rewriting (but may not be n-modular for any $n > 0$). It is an open problem to determine whether all schema mappings satisfying these properties can be defined by nested s-t tgds.

Acknowledgments

Research on this paper was partially supported by NSF grants IIS-0430994 and ARRA 0905276, the Netherlands Organization for Scientific Research (NWO) grant 639.021.508, and the ERC Advanced Grant Webdam on Foundations of Web data management. Part of the work was carried out during a visit by Balder ten Cate at UC Santa Cruz and the IBM Almaden Research Center.

References

1. Abiteboul, S., Duschka, O.M. Complexity of answering queries using materialized views. In *ACM Symposium on Principles of Database Systems (PODS)* (1998) 254–263.
2. Avigad, J., Dean, E., Mumma, J. A formal system for Euclid's Elements. *Rev. Symb. Logic* (2009). To appear.
3. Bernstein, P.A., Haas, L.M. Information integration in the enterprise. *Commun. ACM 51*, 9 (2008), 72–79.
4. Codd, E.F. A relational model for large shared data banks. *Commun. ACM 13* (1970), 377–387.
5. Codd, E.F. Relational completeness of data base sublanguages. *Database Systems*. R. Rustin, ed. Prentice-Hall, 1972, 33–64.
6. Duschka, O.M., Genesereth, M.R. Answering recursive queries using views. *ACM Symposium on Principles of Database Systems (PODS)* (1997), 109–116.
7. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L. Data exchange: semantics and query answering. *Theor. Comp. Sci. 336*, 1 (2005), 89–124.
8. Fagin, R., Kolaitis, P.G., Popa, L. Data exchange: getting to the core. *ACM Trans. Database Syst. 30*, 1 (2005), 174–210.
9. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.-C. Composing schema mappings: second-order dependencies to the rescue. *ACM Trans. Database Syst. 30*, 4 (2005), 994–1055.
10. Fagin R., Vardi, M.Y. The theory of data dependencies—a survey. *Mathematics of Information Processing*, volume 34 of *Proceedings of Symposia in Applied Mathematics*, American Mathematical Society, 1986, 19–71.
11. Fuxman, A., Hernandez, M.A., Ho, H., Miller, R.J., Papotti, P., Popa, L. Nested mappings: schema mapping reloaded. *International Conference on Very Large Data Bases (VLDB)* (2006), 67–78.
12. Haas, L.M. Beauty and the beast: the theory and practice of information integration. *International Conference on Database Theory (ICDT)* (2007), 28–43.
13. Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Roth, M. Clio grows up: from research prototype to industrial t. *ACM International Conference on Management of Data (SIGMOD)* (2005), 805–810.
14. Kolaitis, P.G. Schema mappings, data exchange, and metadata management. *ACM Symposium on Principles of Database Systems (PODS)* (2005), 61–75.
15. Lenzerini, M. Data integration: a theoretical perspective. *ACM Symposium on Principles of Database Systems (PODS)* (2002), 233–246.
16. Rossman, B. Existential positive types and preservation under homomorphisms. *Symposium on Logic in Computer Science (LICS)* (2005), 467–476.
17. ten Cate, B., Kolaitis, P.G. Structural characterizations of schema-mapping languages. *International Conference on Database Theory (ICDT)* (2009), 63–72.
18. Trakhtenbrot, B. Impossibility of an algorithm for the decision problem on finite classes. *Dokl. Akad. Nauk. SSSR, 70* (1950), 569–572.
19. van der Meyden, R. Logical approaches to incomplete information: a survey. *Logics for Databases and Information Systems*. Kluwer, 1998, 307–356.

Balder ten Cate (balder.tencate@uva.nl), INRIA and ENS Cachan.

Phokion G. Kolaitis (kolaitis@cs.ucsc.edu), University of California, Santa Cruz and IBM Research–Almaden.