Schema Mappings, Data Exchange, and Metadata Management

Phokion G. Kolaitis*
IBM Almaden Research Center
kolaitis@almaden.ibm.com

ABSTRACT

Schema mappings are high-level specifications that describe the relationship between database schemas. Schema mappings are prominent in several different areas of database management, including database design, information integration, data exchange, metadata management, and peer-to-peer data management systems. Our main aim in this paper is to present an overview of recent advances in data exchange and metadata management, where the schema mappings are between relational schemas. In addition, we highlight some research issues and directions for future work.

1. Introduction

Schema mappings are specifications that describe the relationships between schemas at a high level. These specifications are typically given in a logical formalism that captures the interaction between schemas at a logical level without spelling out implementation details relevant to the physical level. Schema mappings are widely used in all data management applications that involve data sharing or data transformation. In particular, schema mappings are essential building blocks in information integration, data exchange, metadata management, and peer-to-peer data managements systems. In this paper, we present an overview of recent advances in data exchange and metadata management, where the schema mappings are between relational schemas. We also highlight a number of research issues and suggest directions for future work. Most of the results presented here are based on joint work with Ronald Fagin, Renée J. Miller, Lucian Popa, and Wang-Chiew Tan reported in [13, 14, 15]. In some respects, this paper can be construed as a companion to Lenzerini's paper [23] from his invited tutorial on data integration in PODS 2002, even though it does not aspire to be as comprehensive and encyclopedic as Lenzerini's survey.

Data exchange, also known as data translation, is the problem of taking data structured under a schema, called the source schema, and transforming it into data structured under another schema, called the target schema. Data exchange has been described as the "oldest database problem" [7]. An early low-level tool for data exchange between hierarchical databases was the EXPRESS system, developed at the IBM San Jose Research Laboratory in the 1970s [34]. Data exchange has been a recurrent problem that has taken a new significance with the advent of semi-structured data and the resulting need to exchange data between heterogeneous schemas.

There are obvious similarities, but also clear differences, between data integration and data exchange. In both frameworks, schema mappings are used to specify the relationships between the schemas involved. In data integration, the goal is to synthesize data from different sources into a unified view under a global schema; this view is virtual, in that the data remain in the sources and are accessed by users symbolically via the global schema. In data exchange, the goal is to take a given source instance and transform it to a target instance such that it satisfies the specifications of the schema mapping and also "reflects" the given source data as accurately as possible; unlike data integration, this target instance is a materialized instance, not a virtual view.

Consider a schema mapping between a source schema and a target schema. It is often the case that, given a source instance, there may be multiple target instances, called *s*olutions, that satisfy the specifications of the schema mapping under consideration. This state of affairs gives rise to certain fundamental questions about the semantics and the algorithmics of data exchange. Given a source instance, which solutions are "better" than others? Which solution should one choose to materialize and how difficult is it to compute such a good solution? What is the semantics of target queries and how difficult is it to evaluate such queries? Both information integration and data exchange use the concept of the *c*ertain answers as the standard semantics of query answering, a concept that originated in the study of incomplete databases [35]. The two frameworks, however, adopt different approaches to obtain the certain answers of queries. In data integration, queries posed against the global schema are usually processed via rewriting to queries posed against the source schemas. In data exchange, however, it is natural to try to process target queries by making use of the materialized target instance; furthermore, this may be the only reasonable approach in cases in which the source instance becomes inaccessible af-

*On leave from UC Santa Cruz.

PODS 2005 June 13-15, 2005, Baltimore, Maryland.

ter the exchange has taken place. In turn, this raises the question: for which target queries can the certain answers be obtained by evaluating them on a good solution?

Schema mappings are *metadata*. Bernstein [6] has made a compelling case for the importance of developing both the theory and the practice of metadata management. To this effect, Bernstein has introduced a conceptual framework in which metadata is managed by combining certain basic generic operators on schema mappings, such as *composition*, *merge*, *match*, and *inverse*. Complex transformations on schema mappings can be obtained by repeated combinations of these basic operators; moreover, schema evolution can be dissected and analyzed using the same operators.

The first main challenge in metadata management is to develop rigorous semantics for each of the basic operators. Once this is achieved, the next challenge is to investigate the properties of these operators for different schema-mapping languages. A prominent issue in this investigation has to do with the *closure* properties of schema-mapping specification languages. For instance, is a given schema-mapping specification language closed under composition? In other words, can the composition of two schema mappings be expressed in the same language used to express each of the components of the composition? Another important issue has to do with the algorithmic properties of the basic operators and the schema-mapping specification languages used to express them. In particular, for which schema-mapping specification languages are the basic operators efficiently computable?

The remainder of this paper is organized as follows. In Section 2, we introduce schema mappings and the data exchange problem. In Section 3, we present an overview of results about data exchange with schema mappings specified by tuple-generating dependencies between relational schemas. In Section 4, we focus on the semantics and the computational complexity of query answering in data exchange. In Section 5, we give an account of results about the composition operator as a case study in metadata management. Finally, in Section 6 we conclude with some brief remarks on the connection of this work to Clio, a schema mapping and data exchange tool developed at the IBM Almaden Research Center.

2. Schema Mappings & Data Exchange

A *relational schema* or, simply, a *s*chema, is a finite sequence $\mathbf{R} = \langle R_1, \ldots, R_k \rangle$ of relation symbols, each of which has a fixed arity. An *instance* I *over* $\mathbf{R}$ is a sequence $\langle R_1^I, \ldots, R_k^I \rangle$ such that each R_i^I is a finite relation of the same arity as R_i. To keep the notation simple and when no confusion arises, we will use R_i to denote both the relation symbol and the relation R_i^I that interprets it. Given a tuple t occurring in a relation R, denote by $R(t)$ the association between R and t, and call it a *fact*. Clearly, an instance can be identified with the collection of its facts.

Let $\mathbf{S} = \langle S_1, \ldots, S_n \rangle$ and $\mathbf{T} = \langle T_1, \ldots, T_m \rangle$ be two schemas with no relation symbols in common. We write $\langle \mathbf{S}, \mathbf{T} \rangle$ for the schema $\langle S_1, \ldots, S_n, T_1, \ldots, T_m \rangle$. If I is an instance over $\mathbf{S}$ and J is an instance over $\mathbf{T}$, then we write $\langle I, J \rangle$ for the instance K over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$ such that $S_i^K = S_i^I$ and $T_j^K = T_j^J$, for $1 \leq i \leq n$ and $1 \leq j \leq m$.

DEFINITION 2.1. Let $\mathbf{S}$ and $\mathbf{T}$ be two schemas with no relation symbols in common.

A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ such that Σ is a set of formulas of some logic $\mathcal{L}$ over $\langle \mathbf{S}, \mathbf{T} \rangle$. In such a schema mapping, $\mathbf{S}$ is called the *source* schema, and $\mathbf{T}$ is called the *target* schema.

DEFINITION 2.2. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping.

- An *instance of* $\mathcal{M}$ is an instance $\langle I, J \rangle$ over $\langle \mathbf{S}, \mathbf{T} \rangle$ that satisfies every formula in the set Σ.
- We write $\mathrm{Inst}(\mathcal{M})$ to denote the space of all instances $\langle I, J \rangle$ of $\mathcal{M}$.
- Let I be an instance over $\mathbf{S}$. We say that an instance J over $\mathbf{T}$ is a *solution for* I *under* $\mathcal{M}$ if $\langle I, J \rangle \in \mathrm{Inst}(\mathcal{M})$. We write $\mathtt{SOL}(\mathcal{M}, I)$ for the collection of all solutions for I under $\mathcal{M}$.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping. Intuitively, the formulas in Σ express constraints between the source schema $\mathbf{S}$ and the target schema $\mathbf{T}$. We assume that the satisfaction relation between formulas and instances of the logic $\mathcal{L}$ used to specify the constraints of schema mappings is *preserved under isomorphisms*; this means that if an instance satisfies a formula of $\mathcal{L}$, then every isomorphic instance also satisfies that formula. Clearly, this property is shared by all standard logical formalisms, such as first-order logic, second-order logic, and Datalog. It follows that the set $\mathrm{Inst}(\mathcal{M})$ of instances of $\mathcal{M}$ is *closed under isomorphisms*, that is, if $\langle I, J \rangle \in \mathrm{Inst}(\mathcal{M})$ and $\langle I', J' \rangle$ is isomorphic to $\langle I, J \rangle$, then we also have $\langle I', J' \rangle \in \mathrm{Inst}(\mathcal{M})$.

Note that a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is not a mapping in the mathematical sense; in effect, it is a schema $\langle \mathbf{S}, \mathbf{T} \rangle$ partitioned into two parts $\mathbf{S}$ and $\mathbf{T}$, together with a set Σ of constraints. Nonetheless, every schema mapping $\mathcal{M}$ gives rise to a mathematical mapping such that, for every source instance I, it returns the collection $\mathtt{SOL}(\mathcal{M}, I)$ of all target instances J that are solutions for I under $\mathcal{M}$.

It should be pointed out that a source instance I may not have any solutions; furthermore, if I has solutions, then these solutions need be unique up to isomorphism. As a matter of fact, a source instance may have an arbitrary finite number or an infinite number of non-isomorphic solutions. For example, consider a schema mapping $\mathcal{M}$ in which the source schema $\mathbf{S}$ contains a binary relation symbol E, the target schema $\mathbf{T}$ contains a binary relation symbol H, and the set Σ consists of the first-order formula

$$\forall x \forall y (E(x, y) \rightarrow \exists z (H(x, z) \wedge H(z, y))).$$

Intuitively, this schema mapping transforms edges in E to paths of length 2 in H. Clearly, every source instance has a solution; in fact, for every source instance, there are infinitely many non-isomorphic solutions since solutions are preserved by augmenting H with an arbitrary number of new edges. With these considerations in mind, we are now ready to introduce the basic problems in schema mappings and data exchange.

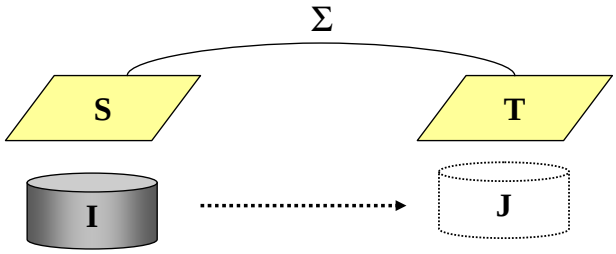

Figure 1: The Data Exchange Problem

DEFINITION 2.3. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping.

- The *existence-of-solutions problem for* $\mathcal{M}$, denoted by $\texttt{SOL}(\mathcal{M})$ is the following decision problem: given a source instance I, does a solution J for I under $\mathcal{M}$ exist? In other words, given I, is $\texttt{SOL}(\mathcal{M}, I) \neq \emptyset$?
- The *data exchange problem associated with* $\mathcal{M}$ is the following function problem: given a source instance I, find a solution J for I under $\mathcal{M}$, provided such a solution exists.

Several remarks are in order now. First, note that, in defining the existence-of-solutions problem and the data exchange problem, we have kept the schema mapping $\mathcal{M}$ fixed, so that the input to these two problems is just a source instance I. Although we will not pursue this here, it is also meaningful to consider the variants of these problems in which both a schema mapping $\mathcal{M}$ and a source instance I are part of the input. In Vardi's [36] taxonomy of problems in database theory, this means that here we are focusing on the *data complexity* of these two problems, instead of their *combined complexity*.

Up to this point, we have not been explicit about the logic $\mathcal{L}$ used to specify schema mappings. As a guiding principle, we want to use logics that are powerful enough to express interesting constraints occurring in applications, while at the same time are well-behaved enough so that the existence-of-solutions problem is tractable or, at the very least, decidable. It is not hard to see that if arbitrary formulas of first-order logic are allowed in the specification of schema mappings, then there are schema mappings for which the existence-of-solutions problem is undecidable. Indeed, as described in [15], we can write a fixed first-order sentence specifying a schema mapping $\mathcal{M}^*$ such that a solution J exists for a source instance I if and only if I is the encoding of a Turing machine and J is the encoding of a terminating computation on some input. Consequently, the existence-of-solutions problem for $\mathcal{M}^*$ is undecidable. This implies that, instead of full first-order logic or some extension of it, we will have to use suitably restricted fragments of first-order logic or some other more tractable logical formalism to specify schema mappings.

Finally, let us comment on the data exchange problem associated with a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$. As formulated in Definition 2.3, this problem asks, given a source instance I, to find a solution J for I under $\mathcal{M}$. As noted earlier, a source instance I may have many (in fact, infinitely many) non-isomorphic solutions. In such cases, which is the "right" solution to return as the answer to the data exchange problem? What criteria should we use to differentiate between solutions? How can we select a solution J to materialize that, in addition to merely satisfying the constraints in Σ, represents the source instance I as faithfully as possible?

The issues raised in the preceding remarks pave the road for the work that will be presented in the next section.

3. Data Exchange with Tuple- & Equality-Generating Dependencies

Starting with Codd's [9] work on functional dependencies, a large variety of constraints in relational databases, called *dependecies*, were investigated in depth during the 1970s and the 1980s. Most of the constraints studied during this period can be expressed by formulas of first-order logic, and fall into one of two classes: *tuple-generating dependencies* and *equality-generating dependencies*. Taken together, these two classes have the same expressive power as the class of *embedded implicational dependencies* [12] (see [22, Section 3] for a discussion about the taxonomy of constraints in relational databases).

A *tuple-generating dependency*, in short *tgd*, is a first-order formula of the form

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

where $\varphi(\mathbf{x})$ is a conjunction of atoms[1] such that the variables of each atom are among those in $\mathbf{x}$, and each variable in $\mathbf{x}$ occurs in at least one of the atoms of $\varphi(\mathbf{x})$; furthermore, $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms with variables among those in $\mathbf{x}$ and $\mathbf{y}$. In effect, every tgd expresses the containment of one conjunctive query in another conjunctive query. Both inclusion dependencies and multivalued dependencies are special cases of tgds. An important subclass of tgds is the collection of *full tgds*; by definition, these are the tgds whose right-hand side of the implication has no existential quantifiers, that is, they tgds of the form

$$(\forall \mathbf{x})(\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x})).$$

For example,

$$(\forall x_1 \forall x_2 \forall x_3)(E(x_1, x_2) \wedge E(x_2, x_3) \rightarrow E(x_1, x_3))$$

is a full tgd constraining E to be a transitive relation, while

$$\forall x \forall y(E(x, y) \rightarrow \exists z(H(x, z) \wedge H(z, y)))$$

is a tgd, but not a full one.

An *equality-generating dependency*, in short *egd*, is a first-order formula of the form

$$(\forall \mathbf{x})(\varphi(\mathbf{x}) \rightarrow (x_i = x_j)),$$

where x_i, x_j are among the variables in $\mathbf{x}$, and $\varphi(\mathbf{x})$ is a conjunction of atoms such that the variables of each atom are among those in $\mathbf{x}$, and each variable in $\mathbf{x}$ occurs in at least

[1] An *atom* is an expression $R(\mathbf{t})$, where R is a relation symbol and $\mathbf{t}$ is a tuple of variables. A *conjunctive query* $q(\mathbf{x})$ 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.

one of the atoms of $\varphi(\mathbf{x})$. Clearly, functional dependencies are a special case of egds.

In what follows, we will drop the universal quantifiers in front of tgds and egds, and will implicitly assume such quantification; all existential quantifiers, however, will be written explicitly.

Tuple-generating dependencies have been extensively used in data integration to specify constraints between the source schema and the global schema (see [23] for an overview). Furthermore, as we will describe next, tuple-generating dependencies and equality-generating dependencies were used in a systematic study of data exchange [13, 14].

Let $\mathbf{S}$ and $\mathbf{T}$ be two schemas with no relation symbols in common.

- A *source-to-target tuple-generating dependency*, in short *s-t tgd*, is a tgd
$$\varphi(\mathbf{x}) \rightarrow (\exists \mathbf{y})\psi(\mathbf{x}, \mathbf{y})$$
such that $\varphi(\mathbf{x})$ is a conjunction of atoms with relation symbols from $\mathbf{S}$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms with relation symbols from $\mathbf{T}$.
- A *target tuple-generating dependency*, in short *target tgd*, is a tgd
$$\varphi(\mathbf{x}) \rightarrow (\exists \mathbf{y})\psi(\mathbf{x}, \mathbf{y})$$
such that both $\varphi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atoms with relation symbols from $\mathbf{T}$.
- A t*arget equality-generating dependency*, in short *target egd*, is an egd
$$\varphi(\mathbf{x}) \rightarrow (x_i = x_j)$$
such that $\varphi(\mathbf{x})$ is a conjunction of atoms from $\mathbf{T}$.
Note that, as indicated earlier, we have dropped the universal quantifiers in front of dependencies.

It should be noted that s-t tgds are the same as GLAV (*global-and-local-as-view*) constraints expressing sound views in data integration. As such, they generalize sound views in LAV (*local-as-view*) and sound views in GAV (*global-as-view*). Indeed, sound views in LAV are s-t tgds of the form $R(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y})$, where R is a relation symbol in the source schema $\mathbf{S}$ and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms with relation symbols from the global schema; similarly, sound views in GAV are s-t tgds of the form $\varphi(\mathbf{x}) \rightarrow P(\mathbf{x})$, where $\varphi(\mathbf{x})$ is a conjunction of atoms with relation symbols from the source schema and P is a relation symbol in the target schema $\mathbf{T}$.

In what follows, we will focus on schema mappings $\mathcal{M}$ of the form $(\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds. The data exchange problem for such schema mappings was first investigated in [13] in a study motivated by the following considerations. There are many situations in which source data have to be translated to target data in such a way that "directional" constraints are satisfied, that is, a condition on the source implies a condition on the target. This is modelled using

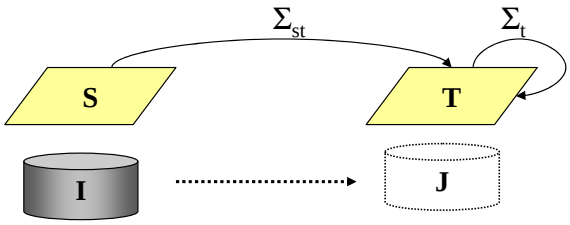

Figure 2: Data Exchange with Source-to-Target Dependencies Σ_{st} and Target Dependencies Σ_t

s-t tgds in the schema mapping. Furthermore, the data in the target may have to obey additional constraints, such as key constraints or inclusion dependencies. In turn, this is modelled using target egds and target tgds.

It goes without saying that one could expand the framework to include source constraints, target-to-source tgds, or more complex constraints, such as tgds $\varphi(\mathbf{x}) \rightarrow (\exists \mathbf{y})\psi(\mathbf{x}, \mathbf{y})$ in which both $\varphi(\mathbf{x})$ and $\psi(\mathbf{x}, \mathbf{y})$ are conjunctions of atoms over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$. As regards source constraints, it is natural to assume that the source data to be exchanged has already been preprocessed, and so the data satisfies all underlying source constraints; moreover, as will be seen later on, source constraints have no direct role in defining the semantics of data exchange. As regards target-to-source constraints or more complex constraints, their presence undoubtedly enhances the modelling power of the framework, but it is usually accompanied by a steep increase in the computational complexity of algorithmic problems in data exchange (see, for instance, [17]).

3.1 Universal Solutions in Data Exchange

Before proceeding further, we should become specific about the values that occur in source instances and in target instances. Assume that $\underline{\mathsf{Const}}$ is the set of all values that may occur in source instances; we call these values *constants*. Let $\underline{\mathsf{Var}}$ be an infinite set of values, called *labelled nulls*, such that $\underline{\mathsf{Const}} \cap \underline{\mathsf{Var}} = \emptyset$. We stipulate that the target instances created in data exchange have values in $\underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}$. If J is a target instance, then we write $\underline{\mathsf{Var}}(J)$ for the set of all labelled nulls occurring in relations in J.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds. Recall that the data exchange problem associated with $\mathcal{M}$ asks: given a source instance I, find a solution J for I under $\mathcal{M}$, that is, a target instance J such that $\langle I, J \rangle$ satisfies $\Sigma_{st} \cup \Sigma_t$.

It is possible that, for a given source instance I, no solution exists. For example, let $\mathcal{M}$ be a schema mapping such that

$$\begin{aligned} \Sigma_{st} &= \{E(x, y) \wedge E(y, z) \rightarrow H(x, z)\} \\ \Sigma_t &= \{H(x, y) \wedge H(x, z) \rightarrow y = z\}. \end{aligned}$$

If I is a source instance containing the facts $E(1, 2)$, $E(2, 3)$, $E(1, 4)$, $E(4, 5)$, then no solution for I exists, since every solution must contain the facts $H(1, 3)$ and $H(1, 5)$, thus vi-

olating the target egd in Σ_t.

At the other extreme, it is easy to see that if $\Sigma_t = \emptyset$, then solutions always exist. Moreover, in this case, every source instance I has infinitely many non-isomorphic solutions, since if extra facts are added to a solution for I, then the resulting instance is still a solution for I.

EXAMPLE 3.1. Let $\mathcal{M}$ be a schema mapping such that

$$\begin{aligned} \Sigma_{st} &= \{E(x,y) \rightarrow (\exists z)(H(x,z) \wedge H(z,y))\} \\ \Sigma_t &= \emptyset. \end{aligned}$$

Let I be the source instance consisting of just the fact $E(1,2)$. The set $\mathtt{SOL}(\mathcal{M}, I)$ of solutions for I under $\mathcal{M}$ contains, among others, the following target instances:

$$\begin{aligned} J_1 &= \{H(1,1), H(1,2)\} \\ J_2 &= \{H(1,2), H(2,2)\} \\ J_3 &= \{H(1,u), H(u,2)\} \\ J_4 &= \{H(1,u), H(u,2), H(u,u)\} \\ J'_n &= \{H(1,v_i) : 1 \leq i \leq n\} \cup \{H(v_i,2) : 1 \leq i \leq n\}, \end{aligned}$$

where u and v_i, $1 \leq i \leq n$, are labelled nulls in $\underline{\mathsf{Var}}$. □

Which member of $\mathtt{SOL}(\mathcal{M}, I)$ should we choose to materialize and return as answer to the data exchange problem on input I? What properties should a solution possess that would make it more desirable for data exchange than other solutions? Intuitively, although every solution for I satisfies the specifications of the schema mapping $\mathcal{M}$, we would like to materialize a solution that carries *no more and no less* information than is required for data exchange. But how can this intuition be turned into a precise concept?

In some respects, this state of affairs is reminiscent of the unification problem in logic programming [2]. Two given terms may or may not be unifiable; if they are, then they can have more than one unifier. In the latter case, the preferred unifier is the *most general unifier*, which is a unifier with the property that every unifier can be obtained from it via a *substitution*.

In [13], the concept of a *universal* solution was introduced, and a case was made that universal solutions are the preferred solutions in data exchange because they are the "most general" solutions in data exchange. The concept of universal solution makes use of the concept of *homomorphism*; the precise definitions of these concepts are as follows.

DEFINITION 3.2. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds.

Let J_1 and J_2 be two target instances. A *homomorphism* $h : J_1 \rightarrow J_2$ is a function $h : \underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(J_1) \rightarrow \underline{\mathsf{Const}} \cup \underline{\mathsf{Var}}(J_2)$ with the following two properties.

1. For every constant $c \in \underline{\mathsf{Const}}$, we have that $h(c) = c$.
2. For every fact $P(\mathbf{t})$ of J_1, we have that $P(h(\mathbf{t}))$ is a fact of J_2 (where, if $\mathbf{t} = (c_1, \ldots, c_m)$, then $h(\mathbf{t}) = (h(c_1), \ldots, h(c_m))$).

DEFINITION 3.3. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds.

If I is a source instance, then a *universal solution for* I is a solution J for I such that for every solution J' for I, there is a homomorphism $h : J \rightarrow J'$.

Intuitively, a homomorphism between two relational structures is a mapping that preserves facts. The definition of homomorphism we just gave is essentially the standard definition of homomorphism between relational structures in which some elements (in the present case, the constants) have been distinguished and can be mapped only to themselves. The concept of homomorphism is a fundamental algebraic concept that been extensively studied in graph theory [21] and in constraint satisfaction [16]. Many basic NP-complete problems can be cast as homomorphism problems. As an illustration, it is easy to verify that a graph $\mathbf{G} = (V, E)$ is 3-colorable if and only if there is a homomorphism from $\mathbf{G}$ to $\mathbf{K}_3$, the complete graph with 3 nodes. Furthermore, homomorphisms are tightly connected to conjunctive-query processing and optimization, since conjunctive-query containment can also be cast as a homomorphism problem [8].

By definition, universal solutions can be mapped homomorphically to every solution; consequently, every solution can be obtained from a given universal solution via a homomorphism and an augmentation with extra facts (tuples in relations). In this sense, universal solutions are indeed the "most general" solutions in data exchange: they carry no more and no less information than is needed for data exchange purposes.

Let us illustrate this intuition by examining the solutions listed in Example 3.1. Solution J_1 is not universal, since no homomorphism from J_1 to J_2 exists (or to any of the other solutions listed). Although J_1 satisfies the s-t tgd $E(x,y) \rightarrow (\exists z)(H(x,z) \wedge H(z,y))$, it contains extra information, as it uses the constant 1 to witness the existential quantifier $\exists z$. Similarly solution J_2 is not universal, since no homomorphism from J_2 to J_1 exists. In contrast, solution J_3 is a universal solution for I. For instance, there is a homomorphism h from J_3 to J_1 with $h(u) = 1$, and a homomorphism g from J_3 to J'_n with $g(u) = v_1$. Solution J_4 is not universal, even though it contains the universal solution J_3. Intuitively, the reason is that J_4 also contains the self-loop $H(u,u)$, a fact not specified in the data exchange. Formally, every homomorphism maps self-loops to self-loops, thus there is no homomorphism from J_4 to J_3 (or to any of the other solutions listed). Finally, each J'_n is a universal solution. For instance, there is a homomorphism h_n from J'_n to J_3 with $g(v_i) = u$, for every $i \leq n$.

In logic programming, a most general unifier of two unifiable terms is known to be unique up to isomorphism. In contrast, the universal solutions for a given source instance I need not have this property. For instance, in Example 3.1, all solutions J'_n are universal and pairwise non-isomorphic, as they have different sizes. Nonetheless, it follows immediately from Definition 3.3 that all universal solutions for a given source instance I are *homomorphically equivalent*, which means that if J and J' are universal solutions for I, then there is a homomorphism h from J to J', and a homomorphism h' from J'

to J. Furthermore, according to the next result from [13]), universal solutions embody in a certain sense the entire space of solutions; this is analogous to the most general unifier of two terms encapsulating the entire space of unifiers of the two terms.

THEOREM 3.4. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds. Assume that I and I' are two source instances, J is a universal solution for I, and J' is a universal solution for I'. The following statements are equivalent:*

1. $\texttt{SOL}(\mathcal{M}, I) \subseteq \texttt{SOL}(\mathcal{M}, I')$.
2. *There is a homomorphism $h : J' \rightarrow J$.*

Consequently, $\texttt{SOL}(M, I) = \texttt{SOL}(M, I')$ if and only if J and J' are homomorphically equivalent.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target tgds and target egds. We mentioned earlier that if $\Sigma_t = \emptyset$, then every source instance I has a solution. Moreover, in this case, every source instance I has a universal solution. In general, however, a given source instance I may have no solution. Furthermore, it is possible that a solution exists for a given source instance I, but no universal solution for I exists [13]. This state of affairs dictates that restrictions have to imposed on the target constraints used, so that the existence of solutions implies the existence of universal solutions (the converse is always trivially true). At the same time, we would like to have restrictions that yield polynomial-time algorithms for testing whether a solutions exists and for computing a universal solution, whenever a solution exists.

The *chase procedure* is an indispensable algorithmic tool for reasoning about dependencies [26, 4, 5] and, in particular, for testing whether a given set of dependencies logically implies another given dependency. As it turns out, the chase procedure is versatile enough to be adapted for productive use in data exchange. Specifically, in [13], the concept of a *weakly acyclic set* of target tgds was introduced and then used to show that a variant of the chase procedure yields a polynomial-time algorithm for testing for solutions and for computing universal solutions in every schema mappings $\mathcal{M}$ in which the set Σ_t of target constraints is the union of a set of target egds with a weakly acyclic set of target tgds.

DEFINITION 3.5. [Weakly acyclic set of tgds] Let Σ be a set of tgds over a schema $\mathbf{T}$. Construct the following directed graph, called the *dependency graph*:

- The nodes of the dependency graph are pairs (R, A) such that R is a relation symbol in $\mathbf{T}$ and A is an attribute of R; call such a pair a *position*.
- The edges of the dependency graph are created according to the following rules. For every tgd $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ, for every x in $\mathbf{x}$ that occurs in ψ, and for every occurrence of x in ϕ in position (R, A_i):
  1. For every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).
  2. In addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$ (if it does not already exist).

We say that the set Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge.

Clearly, if Σ is a set of full tgds, then Σ is weakly acyclic, since the absence of existential quantifiers in the tgds of Σ implies that the dependency graph has no special edges. It is also easy to verify that if Σ is an *acyclic set of inclusion dependencies* [10], then Σ is weakly acyclic as well. In contrast, the singleton set $\Sigma = \{H(x_1, x_2) \rightarrow \exists y H(x_2, y)\}$ is not weakly acyclic, because the dependency graph has a self-loop with a special edge, namely $(H, B) \stackrel{*}{\rightarrow} (H, B)$. Note also that each of the two singleton sets $\Sigma_1 = \{H(x_1, x_2) \rightarrow \exists y H(x_1, y)\}$ and $\Sigma_2 = \{H(x_1, x_2) \rightarrow \exists y H(y, x_2)\}$ is weakly acyclic, but not acyclic; however, their union $\Sigma_1 \cup \Sigma_2$ is not weakly acyclic as it contains the cycle $(H, A) \stackrel{*}{\rightarrow} (H, B) \stackrel{*}{\rightarrow} (H, A)$

The intuition behind the dependency graph and the concept of a weakly set of tgds is as follows. Suppose we try to construct a target instance that satisfies a set Σ_t of target tgds by using the chase procedure. The non-special edges in the dependency graph keep track of the fact that a value may propagate from position (R, A_i) to position (S, B_j) during the chase. The special edges keep track of the fact that propagation of a value from position (R, A_i) to position (S, B_j) also creates a labelled null in every position that has an existentially quantified variable. If a cycle goes through a special edge, then a labelled null appearing in a certain position during the chase may determine the creation of another labelled null, in the same position, at a later chase step. Thus, this process may continue for ever. In contrast, if the set Σ_t is weakly acyclic, it can be shown that the chase procedure will terminate after polynomially many steps. Moreover, if target egds are also present, then the only extra complication is that the chase may fail, but this failure will be detected within polynomially many steps.

The next result from [13] yields a broad sufficient condition for the tractability of the existence-of-solutions problem in data exchange and for the efficient computation of a universal solution, whenever a solution exists.

THEOREM 3.6. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds.*

- *Given a source instance I, a universal solution for I exists if and only if a solution for I exists.*
- *There is a polynomial-time algorithm based on the chase procedure such that, given a source instance I, it tests whether a solution for I exists and, if so, it produces a universal solution solution J for I.*

To appreciate the wide applicability of the preceding Theorem 3.6, note that its hypothesis is fulfilled by every schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ such that Σ_{st} is a set of s-t tgds and Σ_t satisfies one of the following properties:

- Σ_t is a set of target egds (that is, Σ_t contains no target tgds).
- Σ_t is the union of a set of target egds with a set of target full tgds.
- Σ_t is the union of a set of target egds with an acyclic set of inclusion dependencies.

It remains an interesting open problem to identify other broad classes of schema mappings that have the following properties: (1) for a given instance, universal solutions exist if and only if solutions exist; (2) there are polynomial-time algorithms for determining, given a source instance, whether a solution exists and for computing a universal solution, whenever a solution exists.

3.2 The Core: The Smallest Universal Solution

We have made a case that universal solutions are the preferred solutions in data exchange. At the same time, we pointed out that, although all universal solutions for a given source instance are homomorphically equivalent to each other, they need not be isomorphic. In particular, the source instance I in Example 3.1 has infinitely many non-isomorphic universal solutions, since, for every $n \geq 1$, the target instance J'_n has size (number of facts) $2n$ and is a universal solution for I. This raises the question: are some universal solutions "better" than others? Is there a "best" universal solution that we should choose to materialize and, if so, is it efficiently computable?

The pursuit of the "best" universal solution was undertaken in [14] using a *small is beautiful* approach. The main finding of this investigation is that there is a *smallest* universal solution, which is thus the most economical one to materialize in terms of size. This smallest universal solution is unique up to isomorphism and coincides with the *core* of all universal solutions. Furthermore, although computing the core of arbitrary relational structures is an intractable problem, it turns out that there are broad classes of schema mappings for which the core of the universal solutions is computable in polynomial time. In the remainder of this section, we present an overview of these findings.

Let $\mathbf{G} = (V, E)$ be an (undirected) *graph*, where V is the set of its nodes and E is the set of its edges; this means that E is a non-empty binary relation on V that is symmetric and irreflexive (no self-loops). A *subgraph* of $\mathbf{G}$ is a graph $\mathbf{G}' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$; it is a *proper* subgraph if E' is a proper subset of E.

DEFINITION 3.7. Let $G = (V, E)$ be a graph.

- A subgraph $\mathbf{G}' = (V', E')$ is a *core of* $\mathbf{G}$ if the following two conditions hold:
  1. There is homomorphism from $\mathbf{G}$ to $\mathbf{G}'$.
  2. There is no homomorphism from $\mathbf{G}'$ to a proper subgraph of $\mathbf{G}'$.
- $\mathbf{G}$ is a *core* if it is a core of itself.

Cores have been studied in depth by graph theorists [20, 21]. For every $n \geq 2$, the complete graph $\mathbf{K}_n$ with n nodes is a core; for every $n \geq 1$, the odd cycle $\mathbf{C}_{2n+1}$ is also a core. It is also easy to see that a graph $\mathbf{G}$ is 2-colorable if and only if $\mathbf{K}_2$ is a core of $\mathbf{G}$. In particular, $\mathbf{K}_2$ is a core of every even cycle $\mathbf{C}_{2n}$. There is an abundance of cores. As a matter of fact, it has been shown that almost all finite graphs are cores [24], which means that, under the uniform probability measure, the asymptotic probability of a finite graph being a core is equal to 1. The next proposition summarizes some well known and easy to prove properties of cores of finite graphs.

PROPOSITION 3.8. *Let* $\mathbf{G}$ *be a finite graph.*

- $\mathbf{G}$ *has a core.*
- *The cores of* $\mathbf{G}$ *are pairwise isomorphic; thus, we can talk about the core of* $\mathbf{G}$*, denoted by* $\underline{\mathsf{core}}(\mathbf{G})$*.*
- *There is one-to-one homomorphism from* $\underline{\mathsf{core}}(\mathbf{G})$ *to* $\mathbf{G}$*.*
- $\mathbf{G}$ *is homomorphically equivalent to its core. It follows that two graphs are homomorphically equivalent if and only if their cores are isomorphic.*

It should be noted that there are infinite graphs that have no cores; thus, the hypothesis that $\mathbf{G}$ is finite in Proposition 3.8 is of the essence. The concept of a core is perfectly meaningful for directed graphs and, more generally, for arbitrary relational structures. Proposition 3.8 easily extends to every finite relational structure, so that every finite relational structure has a core, which is unique up to isomorphism. Cores of relational databases have been studied in the context of conjunctive-query processing [8], since minimizing a conjunctive query amounts to finding the core of the *canonical* database associated with the query (this is the database whose facts are the conjuncts of the conjunctive query).

For our purposes here, we will focus on cores of target instances in schema mappings $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ in which Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds and target tgds. The definition of the core of a target instance is entirely analogous to that of the core of a graph, except that the homomorphisms considered must obey Definition 3.3 and, thus, map constants to themselves. This means, that we consider only homomorphisms h between target instances such that $h(c) = c$, for every $c \in \underline{\mathsf{Const}}$. In what follows, if J is a target instance, we will write $\underline{\mathsf{core}}(J)$ to denote the (unique up to isomorphism) core of J.

Let I be a source instance for which universal solutions exist. As we saw earlier, all universal solutions for I are homomorphically equivalent. Consequently, Proposition 3.8 (extended to target instances) implies that the cores of the universal solutions for I are isomorphic; in other words, all universal solutions for I have the same core up to isomorphism. In

[14], it was shown that the core of a solution for a source instance I is itself a solution for I. It follows that the core of universal solutions is itself a universal solution, hence it is the *smallest* universal solution. We collect these facts into a proposition.

PROPOSITION 3.9. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping in which Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds and target tgds.*

1. *If I is a source instance and J is a solution for I, then $\underline{\mathsf{core}}(J)$ is a solution for I.*
2. *If I is a source instance and J is a universal solution for I, then also $\underline{\mathsf{core}}(J)$ is a universal solution.*
3. *If I is source instance for which a universal solution exists, then there is a universal solution J_0 having the following properties:*
   - *J_0 is a core and is isomorphic to the core of every universal solution for I.*
   - *If J is a universal solution for I, there is a one-to-one homomorphism from J_0 to J. It follows that $|J_0| \leq |J|$, where $|J_0|$ and $|J|$ are the sizes of J_0 and J; hence, J_0 is the smallest universal solution.*

Note that if $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is a schema mapping in which Σ is a set of arbitrary dependencies (say, arbitrary first-order formulas), then the core of a solution for a source instance I need not be a solution for I. Thus, the first part of Proposition 3.9 depends crucially on the assumption that the dependencies in the schema mapping $\mathcal{M}$ are s-t tgds, target egds, and target tgds. The other parts of Proposition 3.9 follow from the first part, the definitions, and Proposition 3.8. Returning to Example 3.1, we see that solution J_3 is the smallest universal solution for I, and so is the isomorphic solution J_1'.

The core of the universal solutions is the preferred universal solution to materialize in data exchange, since it is the unique most compact universal solution. In turn, this raises the question of how to compute cores of universal solutions. Theorem 3.6 asserts that universal solutions can be computed in polynomial time using the chase, provided the set of target tgds is weakly acyclic. However, the result of the chase procedure, while a universal solution, need not be the core of the universal solutions. Consequently, different algorithmic tools are needed for computing the core of the universal solutions in data exchange.

Let us consider for a moment the computational complexity of computing the core of a graph. As mentioned earlier, a graph $\mathbf{G}$ is 3-colorable if and only if there is a homomorphism from $\mathbf{G}$ to $\mathbf{K}_3$, the complete graph with 3 nodes. From this, it follows that a graph $\mathbf{G}$ is 3-colorable if and only if $\underline{\mathsf{core}}(\mathbf{G} \oplus \mathbf{K}_3) = \mathbf{K}_3$, where $\mathbf{G} \oplus \mathbf{K}_3$ is the disjoint union of $\mathbf{G}$ and $\mathbf{K}_3$. Consequently, unless P = NP, there is no polynomial-time algorithm for computing the core of a given structure. Indeed, if such an algorithm existed, then we could determine in polynomial time whether a graph is 3-colorable by first running the algorithm to compute the core of $\mathbf{G} \oplus \mathbf{K}_3$ and then checking if the answer is equal to $\mathbf{K}_3$. This intractability of computing the core of graphs and, more generally, of finite relational structure was already realized by Chandra and Merlin [8] in their work on conjunctive-query minimization. Later on, Hell and Nešetřil [20] showed that the following problem, called CORE RECOGNITION, is coNP-complete: given a graph $\mathbf{G}$, is it a core? Finally, in [14], it was shown that the following problem, called CORE IDENTIFICATION, is DP-complete: given two graphs $\mathbf{G}$ and $\mathbf{H}$, is $\mathbf{H}$ the core of $\mathbf{G}$? The class DP consists of all decision problems that can be written as an intersection of NP-problem and a coNP-problem [31, 32]. Since DP contains both NP and coNP as subclasses, DP-complete problems are regarded as "harder" than NP-complete problems.

The preceding complexity-theoretic results reveal that, unless P = NP, computing the core of a graph (or, of a finite relational structure) is an intractable problem. In data exchange, however, the goal is to compute the core of a universal solution, rather than the core of an arbitrary instance. Therefore, the intractability of computing the core of an arbitrary instance does not automatically imply the intractability of computing the core of universal solutions. In fact, as we are about to see, for certain large classes of schema mappings, polynomial-time algorithms for computing the core of universal solutions do exist.

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds (no target tgds). For such schema mappings, a polynomial-time algorithm, called the *blocks* algorithm, for computing the core of universal solutions was given in [14]. A conceptually simpler polynomial-time algorithm, called the *greedy* algorithm, for the same task was subsequently given in the full version of [14]. Intuitively, given a source instance I, the greedy algorithm first determines whether solutions for I exist, and then, if solutions exist, computes the core of the universal solutions for I by successively removing tuples from a universal solution for I, as long as I and the instance resulting in each step satisfy the s-t tgds in Σ_{st}.

Before describing the greedy algorithm, let us recall that a *fact* is an expression of the form $R(t)$ indicating that the tuple t belongs to the relation R; moreover, every instance can be identified with the set of all facts arising from the relations of that instance.

ALGORITHM 3.10. **Greedy Algorithm**

Parameter: Schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds.

Input: source instance I.

Output: the core of the universal solutions for I, if solutions exist; "failure", otherwise.

1. Chase I with Σ_{st} to produce a target instance J that is a universal solution for I under the schema mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma_{st})$.
2. Chase J with Σ_t; if the chase fails, then stop and return "failure"; otherwise, let J' be the universal solution for I produced by the chase.

3. Initialize J^* to be J'.
4. While there is a fact $R(t)$ in J^* such that $\langle I, J^* - \{R(t)\}\rangle$ satisfies Σ_{st}, set J^* to be $J^* - \{R(t)\}$.
5. Return J^*.

The following result about the greedy algorithm was established in the full version of [14].

THEOREM 3.11. *Assume that $(\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is a data exchange setting such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds. Then Algorithm 3.10 is a correct, polynomial-time algorithm for testing for the existence of solutions and for computing the core of universal solutions, if solutions exists.*

In [14], it was left as an open problem to determine whether there are polynomial-time algorithms for computing the core in richer schema mappings. In the meantime, Gottlob [18] obtained a number of results concerning the computation of cores in data exchange. In particular, he established that if $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is set of target egds and target full tgds, then the core of universal solutions can be computed in polynomial time using a sophisticated extension of the blocks algorithm. It remains to be seen whether these tractability results extend to schema mappings in which the set of target constraints is the union of a set of target egds with a weakly acyclic set of target tgds.

4. Query Answering in Data Exchange

Suppose that a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ is used to exchange data from a source schema $\mathbf{S}$ to a target schema $\mathbf{T}$. Suppose also that a query q over the target schema $\mathbf{T}$ is posed. What does answering this query using source data mean? As we have seen earlier, given a source instance I, there may be infinitely many solutions J for I; furthermore, if q is evaluated on different solutions J for I, it is possible that different answers are produced. This ambiguity raises the conceptual problem of giving precise semantics to query answering in data exchange.

Similar conceptual problems were encountered much earlier in the study of *incomplete* databases; they were addressed by introducing the concept of *the certain answers* as the semantics of query answering (see [35] for a survey). The certain answers were also adopted as the standard semantics of query answering in information integration (see [1, 23]). What indefinite databases and information integration have in common is that queries are posed not against a single database, but rather against the set of all *possible* databases in certain contexts, that is, all databases satisfying a certain specification. By definition, the certain answers of a query q are the tuples that occur in the intersection of all $q(J)$'s, as J ranges over all databases satisfying the specification at hand. This concept is also perfectly meaningful in data exchange, where the schema mapping can be viewed as a specification of the solutions for a given source instance. Thus, in data exchange, given a source instance I, the *possible* databases are the solutions for I.

DEFINITION 4.1. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let q be 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 $certain_{\mathcal{M}}(q, I)$, is the set
$$certain_{\mathcal{M}}(q, I) = \cap\{q(J) : J \in \mathtt{SOL}(\mathcal{M}, I)\}.$$
- *Computing the certain answers of q* is the following decision problem: given a source instance I and a tuple t of constants from I, is t in $certain_{\mathcal{M}}(q, I)$?

On the face of it, the definition of the certain answers is highly non-effective, since evaluating $certain_{\mathcal{M}}(q, I)$ entails computing the intersection of infinitely many sets. In information integration, the main approach to computing the certain answers is to try to rewrite queries over the target to queries over the sources. In data exchange, one would like to take advantage of the materialized solution and use it to obtain the certain answers of target queries. As shown in [13], the certain answers of *unions of conjunctive queries* can be obtained using a universal solution. Recall that 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. A *union of conjunctive queries* is a finite disjunction of conjunctive queries. The following result is from [13].

THEOREM 4.2. *Assume that $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds and target tgds. Let q be a union of conjunctive queries over the target schema $\mathbf{T}$.*

- *If I is a source instance and J is a universal solution for I, then*
$$certain_{\mathcal{M}}(q, I) = q(J)_{\downarrow},$$
where $q(J)_{\downarrow}$ is the set of all "null-free" tuples in $q(J)$, that is, all tuples $\mathbf{t}$ in $q(J)$ such that every value in t is a constant in Const.
- *Assume further that Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds. Then there is a polynomial-time algorithm for computing the certain answers of q.*

The preceding Theorem 4.2 provides further evidence for the goodness of universal solutions in data exchange. The proof of the first part of this result uses the existence of homomorphisms from a universal solution to every solution, and the preservation of conjunctive queries under homomorphisms. The second part follows immediately by combining the first part with Theorem 3.6 and the fact that every fixed conjunctive query can be evaluated in polynomial time. As an illustration of Theorem 4.2, let us consider again the schema mapping $\mathcal{M}$ and the source instance $I = \{E(1, 2)\}$ in Example 3.1. Let $q(x)$ be the conjunctive query $\exists w H(x, w)$. Using the definitions, it is easy to verify that $certain_{\mathcal{M}}(q, I) = \{1\}$. Recall that the target instance $J_3 = \{H(1, u), H(u, 2)\}$ is a universal solution for I. Clearly, $q(J_3) = \{1, u\}$, hence $q(J_3)_{\downarrow} = \{1\} = certain_{\mathcal{M}}(q, I)$, as predicted by Theorem 4.2.

Conjunctive queries with inequalities ($\neq$) form one of the most extensively studied extensions of conjunctive queries. By definition, a *conjunctive query with inequalities* 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 inequalities $u \neq v$; a *union of conjunctive query with inequalities* is a finite disjunction of conjunctive query with inequalities. Conjunctive queries with inequalities are more expressive than conjunctive queries; this increase in expressive power, however, often comes at a price. In particular, as we are about to see, Theorem 4.2 does not extend to unions of conjunctive queries with inequalities.

First, it is easy to see that the certain answers of conjunctive queries with inequalities cannot be obtained by simply evaluating them on some universal solution and then discarding all tuples containing nulls. Intuitively, the reason for this is that a conjunctive query with inequalities need not be preserved under homomorphisms; thus, if it holds on some universal solution, then it need not hold on every solution. Concretely, let $\mathcal{M}$ be the schema mapping in Example 3.1, let $p(x)$ be the query $\exists w(H(x, w) \wedge w \neq x)$, and let I_0 be the source instance consisting of just the fact $E(1, 1)$. Clearly, $certain_{\mathcal{M}}(p, I_0) = \emptyset$, since $J_0 = \{H(1, 1)\}$ is a solution for I_0 and $p(J_0) = \emptyset$. At the same time, the target instance $J_5 = \{(1, u), (u, 1)\}$ is a universal solution for I_0 (in fact, it is the core of the universal solutions for I_0), and $p(J_5)_{\downarrow} = \{1\}$.

The next result from [13] pinpoints the computational complexity of computing the certain answers of unions of conjunctive queries with inequalities; in particular, it shows that this problem can be intractable, unless at most one inequality is allowed in every conjunctive query in the union.

THEOREM 4.3. *Assume that $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is the union of a set of target egds with a weakly acyclic set of target tgds.*

- *If q is a union of conjunctive queries with at most one inequality per conjunctive query, then the certain answers of q are polynomial-time computable.*
- *If q is a union of conjunctive queries with inequalities, then the certain anwers of q is a* coNP *problem.*
- *Computing the certain answers of unions of conjunctive queries with inequalities can be a* coNP*-complete problem, even if the union consists of two conjunctive queries each of which has at most two inequalities, and the schema mapping has no target constraints.*

Abiteboul and Duschka [1] showed that there is a single conjunctive query with seven inequalities and a schema mapping with no target constraints for which computing the certain answers in a coNP-complete problem. In [13], it was conjectured that there is a single conjunctive query with two inequalities and a schema mapping with no target constraints for which computing the certain answers is a coNP-complete problem. This conjecture has recently been proved by Madry [27]. In fact, the s-t tgds in the schema mapping constructed by Madry (as well as in the schema mapping constructed by Abiteboul and Duschka), are sound views in LAV, which means that they are s-t tgds of the form $R(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ with R a relation symbol in the source schema $\mathbf{S}$. Combined with Theorem 4.3, these results yield a rather complete picture of the complexity of the certain answers of conjunctive queries with inequalities.

We saw earlier that the certain answers of conjunctive queries with inequalities cannot always be obtained by evaluating them on some universal solution and then discarding all tuples containing a null value. In fact, this fails even if the universal solution chosen is the core of the universal solutions. Nonetheless, among all universal solutions, the core gives the *best approximation* to the certain answers of unions of conjunctive queries. This is made precise in the next proposition, which is proved using the fact that the core has one-to-one homomorphisms to every universal solution.

PROPOSITION 4.4. *Assume that $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is a schema mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a set of target egds and target tgds. Assume also that I is a source instance for which universal solutions exist and let J_0 be the core of the universal solutions for I. If q is a union of conjunctive queries with inequalities, then*

- *$q(J_0) \subseteq q(J)$, for every universal solution J for I;*
- *$q(J_0)_{\downarrow} = \cap\{q(J) : J$ is universal for $I\} \subseteq certain_{\mathcal{M}}(q, I)$.*

The concept of the certain answers in data exchange was arrived at by taking the solutions for an instance to be the *possible* databases. We have made a case, however, that the universal solutions are the preferred solutions in data exchange. This suggests an alternative semantics of query answering in data exchange by taking the universal solutions for an instance to be the *possible* databases in the definition of the certain answers. The following concept of the *universal-certain answers*, in short *u-certain answers*, was introduced in the full version of [14].

DEFINITION 4.5. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping and let q be a query over the target schema $\mathbf{T}$.

- If I is source instance, then *the u-certain answers of q on I with respect to $\mathcal{M}$*, denoted $u\text{-}certain_{\mathcal{M}}(q, I)$, is the set
$$u\text{-}certain_{\mathcal{M}}(q, I) = \cap\{q(J) : J \text{ is universal for } I\}.$$
- *Computing the u-certain answers of q* is the following decision problem: given a source instance I and a tuple t of constants from I, does $t \in u\text{-}certain_{\mathcal{M}}(q, I)$?

From the definitions, it follows immediately that
$$certain_{\mathcal{M}}(q, I) \subseteq u\text{-}certain_{\mathcal{M}}(q, I).$$
Furthermore, if q is a union of conjunctive queries and I is a source instance for which universal solutions exist. then Theorem 4.2 implies that
$$certain_{\mathcal{M}}(q, I) = u\text{-}certain_{\mathcal{M}}(q, I).$$

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schema mapping such that $\Sigma_$ is a set of s-t tgds and Σ_t is a set of target egds. By Theore 3.11, given a source instance I, we can decide in polynomi time whether a solution for I exists and, if it does, construc the core J_0 of the universal solutions for I in polynomi time. From this fact and Proposition 4.4, it follows that q is a union of conjunctive queries with inequalities, the $u\text{-}certain_{\mathcal{M}}(q, I)$ can be evaluated in polynomial time, sinc $u\text{-}certain_{\mathcal{M}}(q, I) = q(J_0)_{\downarrow}$. Thus, we have established th following result.

COROLLARY 4.6. *Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ be a schem mapping such that Σ_{st} is a set of s-t tgds and Σ_t is a s of target egds. If q is a union of conjunctive queries with in equalities, then there is a polynomial-time algorithm for com puting $u\text{-}certain_{\mathcal{M}}(q, I)$.*

To appreciate the preceding Corollary 4.6, note that it not at all clear how this result can be proved without usin the concept of the core and Theorem 3.11. Note also tha by Gottlob's recent results [18], Corollary 4.6 extends to a schema mappings $\mathcal{M}$ in which Σ_t is a set of target egds an target full tgds.

We saw that the u-certain answers semantics coincides with the certain answers semantics on unions of conjunctive queries, but the two may digress on unions of conjunctive queries with inequalities. The u-certain answers semantics has a definite computational advantage over the certain answers semantics, as the former is polynomial-time computable in settings in which the latter is coNP-complete. Several questions merit further investigation. How do these two semantics compare from a pragmatic point of view? In other words, which of the two captures better the intent of the user? How do they compare on queries that more expressive than the ones considered here? More broadly, are there are other semantics that are meaningful for query answering in data exchange? And what criteria can we develop to compare the quality of query answering in data exchange under different semantics?

5. Composing Schema Mappings

Suppose that we have two schema mappings such that the target schema of the first is the source schema of the second. We would like to have a *composition operator* that takes two such schema mappings as input and produces a third schema mapping that has the same effect as applying the two original schema mappings one after the other. Such an operator could be a powerful component in a data exchange tool that is able to automatically synthesize several consecutive schema mappings into a single schema mapping that captures the combined effect of all schema mappings in the sequence. Indeed, the resulting single schema mapping could then be used during the run-time phase for direct data exchange or for query answering, potentially yielding performance benefits. The concept of the composition operator occupies also a central place in Bernstein's metadata management framework [6]. The reason for this is that composition is a key building block in constructing more powerful metadata operators. In particular, schema evolution can be analyzed via repeated applications of composition.

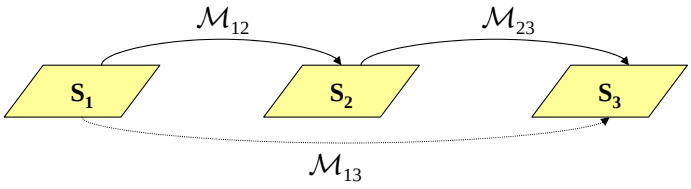

Figure 3: Composing Schema Mappings

More formally, let $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$ be three schemas with no relation symbols in common pairwise. Assume also that $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ are two schema mappings in which Σ_{12} and Σ_{23} are sets of formulas of some logic $\mathcal{L}$ over $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$ and $\langle \mathbf{S}_2, \mathbf{S}_3 \rangle$, respectively. The goal is to have a *composition operator* that, given two such schema mappings, produces a schema mapping $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ that is "equivalent" to the successive applications of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. It is not clear, however, what being "equivalent" exactly means in this context. Therefore, the first step in the study of the composition operator on schema mappings is to develop rigorous semantics for this operator.

Madhavan and Halevy [25] were the first to propose and investigate a precise semantics for the composition operator. Since one of their primary motivations was query answering in peer-to-peer data management systems, their definition carries a class $\mathcal{Q}$ of queries over the schema $\mathbf{S}_3$ as a parameter. Specifically, it is stipulated that, for every query q in $\mathcal{Q}$, the certain answers of q with respect to the composition $\mathcal{M}_{13}$ coincide with the certain answers obtained by successively applying $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. The dependence on the class of queries results to inequivalent semantics for different classes of queries. Indeed, as shown in [15], the semantics of the composition w.r.t. conjunctive queries is different from those w.r.t. conjunctive queries with inequalities. Also, even for a fixed class of queries, the composition of two schema mappings need not be unique up to logical equivalence.

A different semantics for the composition operator was given and thoroughly investigated in [15]. This semantics has a set-theoretic flavor and is obtained by simply composing the spaces $\mathrm{Inst}(\mathcal{M}_{12})$ and $\mathrm{Inst}(\mathcal{M}_{23})$ of the instances (recall Definition 2.2) of the schema mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$.

DEFINITION 5.1. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two schema mappings such that the schemas $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$ have no relation symbols in common.

A schema mapping $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ is a *composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$* if for every instance I_1 over $\mathbf{S}_1$ and every instance I_3 over $\mathbf{S}_3$, the following are equivalent:

1. $\langle I_1, I_3 \rangle \models \Sigma_{13}$;
2. There is an instance I_2 over $\mathbf{S}_2$ such that $\langle I_1, I_2 \rangle \models \Sigma_{12}$ and $\langle I_2, I_3 \rangle \models \Sigma_{23}$.

In symbols, $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ is a *composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$* if and only if

$$\mathrm{Inst}(\mathcal{M}_{13}) = \mathrm{Inst}(\mathcal{M}_{12}) \circ \mathrm{Inst}(\mathcal{M}_{23}),$$

where $\circ$ denotes the set-theoretic composition of two binary relations.

The preceding Definition 5.1 can be construed as the natural definition of the *operational* semantics of the composition operator on schema mappings. The following basic facts are immediate consequences of this definition.

If both $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}_1, \mathbf{S}_3, \Sigma')$ are compositions of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, then the sets Σ and Σ' are logically equivalent. For this reason, from now on, we will talk about *the composition* of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, and we will write $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ to denote it.

As stated in Section 2, the spaces $\mathrm{Inst}(\mathcal{M}_{12})$ and $\mathrm{Inst}(\mathcal{M}_{23})$ are closed under isomorphism. It follows that the space $\mathrm{Inst}(\mathcal{M}_{12}) \circ \mathrm{Inst}(\mathcal{M}_{23})$ of their composition as binary relations is also closed under isomorphism. Thus, $\mathrm{Inst}(\mathcal{M}_{12}) \circ \mathrm{Inst}(\mathcal{M}_{23})$ can be identified with the following query, called the *composition query of* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$: given two instances I_1 and I_3, is $\langle I_1, I_3 \rangle$ in $\mathrm{Inst}(\mathcal{M}_{12}) \circ \mathrm{Inst}(\mathcal{M}_{23})$? Clearly, a schema mapping $(\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ is the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ if and only if the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is definable by the set Σ_{13}. From now on, if Σ_{13} is a set of formulas such that $(\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ is the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, we will say that the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is *definable* by Σ_{13}.

Once the semantics of the composition operator has been put in place, several technical issues arise. As mentioned in the Introduction, the first key issue is the *closure* of schema-mapping specification languages under composition. More precisely, let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two schema mappings in which Σ_{12} and Σ_{23} are sets of formulas of some logic $\mathcal{L}$. Is the composition of these two schema mappings definable in $\mathcal{L}$? In other words, is there a set Σ_{13} of $\mathcal{L}$-formulas such that $\mathcal{M}_{12} \circ \mathcal{M}_{23} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$? If so, can such a set be effectively constructed? Last but not least, what is the computational complexity of the composition query of two given schema mappings?

5.1 Composing s-t tgds

The aforementioned issues were investigated in [15] for schema mappings of the form $(\mathbf{S}, \mathbf{T}, \Sigma_{st})$, where Σ_{st} is a set of s-t tgds (no target constraints). In a nutshell, it was shown there that the language of s-t full tgds is closed under composition, while on the contrary the language of s-t tgds is not. Furthermore, the composition query of two schema mappings specified by s-t tgds may be NP-complete. These findings are described in more detail in the next result.

THEOREM 5.2. *Let* $\mathbf{S}_1$, $\mathbf{S}_2$, $\mathbf{S}_3$ *be three schemas with no relation symbols in common, and let* $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ *and* $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ *be two schema mappings.*

- *If both* Σ_{12} *and* Σ_{23} *are finite sets of s-t full tgds, then the composition* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ *is definable by a finite set of s-t full tgds. Consequently, the composition query of* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$ *is a polynomial-time query.*
- *If* Σ_{12} *is a finite set of s-t full tgds and* Σ_{23} *is a finite set of s-t tgds, then the composition* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ *is definable by a finite set of s-t tgds. Consequently, the composition query of* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$ *is a polynomial-time query.*
- *If both* Σ_{12} *and* Σ_{23} *are finite sets of s-t tgds, then the composition query of* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$ *is in* NP.
- *There exist schema mappings* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$ *such that* Σ_{12} *is a finite set of s-t tgds,* Σ_{23} *is a finite set of s-t full tgds, and the following hold for the composition of these two schema mappings:*
  1. *The composition query of* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ *is* NP*-complete.*
  2. *The composition* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ *is not definable by any formula of least fixed-point logic* LFP*. In particular, the composition* $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ *is not definable by any formula of first-order logic or of Datalog.*

Several remarks are in order now. To begin with, note that the proof of the first part of Theorem 5.2 actually gives an algorithm for finding a set Σ_{13} of s-t full tgds that defines the composition; this set, however, can be exponentially larger than the sets Σ_{12} and Σ_{23}. The same state of affairs holds true for the second part of Theorem 5.2. In certain respects, these two parts contain the best positive results about the composition of finite sets of s-t tgds. Indeed, the last part of Theorem 5.2 implies that the composition of a finite set of s-t tgds with a finite set of s-t full tgds need not be definable by a finite set of s-t tgds. The next example from [15] illustrates this phenomenon.

EXAMPLE 5.3. Consider the following three schemas $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_3$: schema $\mathbf{S}_1$ consists of a single unary relation symbol `Emp` of employees; schema $\mathbf{S}_2$ consists of one binary relation symbol `Mgr'` associating each employee with a manager; schema $\mathbf{S}_3$ consists of a binary relation symbol `Mgr` and an additional unary relation symbol `SelfMgr`, intended to store employees who are their own managers. Consider now the schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$, where

$$\Sigma_{12} = \{\mathtt{Emp}(e) \rightarrow \exists m \mathtt{Mgr}'(e, m)\}$$
$$\Sigma_{23} = \{\mathtt{Mgr}'(e, m) \rightarrow \mathtt{Mgr}(e, m), \mathtt{Mgr}'(e, e) \rightarrow \mathtt{SelfMgr}(e)\}.$$

In [15], it is shown that the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is not definable by any set (finite or infinite) of s-t tgds. The reason for this is that every set of s-t tgds is preserved under homomorphisms of target instances, while the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is not. It is not hard, however, to show that the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ of these two schema mappings is definable by the following first-order formula:

$$\forall e(\mathtt{Emp}(e) \rightarrow \exists m \mathtt{Mgr}(e, m)) \wedge$$
$$\forall e((\forall x(\mathtt{Mgr}(e, x) \rightarrow (e = x))) \rightarrow \mathtt{SelfMgr}(e)).$$

Consequently, the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is a polynomial-time query. □

The last part of Theorem 5.2 asserts that the composition of a finite set of s-t tgds with a finite set of s-t tgds need not be first-order definable; moreover, the associated composition query may actually be NP-complete. This was proved in [15] via reduction from 3-COLORABILITY that we now give.

Consider the following three schemas $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_3$: schema $\mathbf{S}_1$ consists of a single binary relation symbol E; schema $\mathbf{S}_2$ consists of two binary relation symbols C and F; schema $\mathbf{S}_3$ consists of one binary relation symbol D. Consider now the schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$, where

$$\begin{aligned}\Sigma_{12} &= \{E(x,y) \rightarrow F(x,y),\\ &\quad E(x,y) \rightarrow \exists u C(x,u),\\ &\quad E(x,y) \rightarrow \exists v C(y,v)\}\\ \Sigma_{23} &= \{(C(x,u) \wedge C(y,v) \wedge F(x,y) \rightarrow D(u,v))\}.\end{aligned}$$

Given a graph $\mathbf{G} = (V, E)$, let I_1 be the instance over $\mathbf{S}_1$ consisting of the edge relation E of $\mathbf{G}$, and let I_3 be the instance over the schema $\mathbf{S}_3$ with

$$D = \{(r,g),(g,r),(b,r),(r,b),(g,b),(b,g)\}.$$

In words, D is the edge relation of the complete graph $\mathbf{K}_3$ on three nodes r, g, b. It is now easy to verify that $\mathbf{G}$ is 3-colorable if and only if $\langle I_1, I_3\rangle$ is in $\mathrm{Inst}(\mathcal{M}_{12}) \circ \mathrm{Inst}(\mathcal{M}_{23})$. Thus, the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is NP-complete. In addition, since the above reduction is expressible in first-order logic, results by Dawar [11] imply that the composition query of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ is not definable in least fixed-point logic LFP, a logic that is well known to subsume both first-order logic and Datalog.

Observe that each s-t tgd in Σ_{12} has at most one existential quantifier, while Σ_{23} consists of a single s-t full tgd. Thus, the preceding construction draws a rather sharp boundary on the definability of the composition of schema mappings specified by finite sets of s-t tgds. More precisely, the composition of a finite set of s-t full tgds with a finite set of s-t tgds is always definable by a first-order formula (and, in fact, definable by a finite conjunction of s-t tgds), while the composition of a finite set of s-t tgds having at most one existential quantifier with a set consisting of a single s-t full tgd may not even be LFP-definable (and, a fortiori, not first-order definable). Similarly, the computational complexity of the associated composition query may jump from solvable in polynomial time to NP-complete.

5.2 Second-order tgds

What is the "right" specification language for expressing the composition of schema mappings specified by finite sets of s-t tgds? There are two main desiderata in such a specification language. First, it should be powerful enough to express the composition of schema mappings specified by finite sets of s-t tgds, while at the same time it should itself be closed under composition. Second, it should enjoy good algorithmic properties for data exchange purposes.

A class of existential second-order formulas, called *second-order tuple-generating dependencies (*SO *tgds)*, was introduced and studied in [15]. In informal terms, an SO tgd is a source-to-target dependency suitably extended with existentially quantified function symbols. The composition of two finite sets of s-t tgds is always definable by an SO tgd. Moreover, the composition of two SO tgds is always definable by an SO tgd. Finally, SO tgds have good properties for data exchange, since the chase procedure can be extended to SO tgds so that it produces polynomial-time computable universal solutions in schema mapping specified by SO tgds. We now formally define SO tgds.

Definition 5.4. Let $\mathbf{S}$ and $\mathbf{T}$ be two schemas with no relation symbols in common. A *second-order tuple-generating dependency (SO tgd) over* $\langle \mathbf{S}, \mathbf{T}\rangle$ is a formula of the form

$$\exists f_1 \ldots \exists f_m\, (\forall \mathbf{x}_1(\phi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x}_n(\phi_n \rightarrow \psi_n)),$$

where

- Each f_i is a function symbol.
- Each ϕ_i is a conjunction of
  1. atomic formulas $R(y_1, \ldots, y_k)$, where R is a k-ary relation symbol of the schema $\mathbf{S}$ and $y_1, \ldots, y_k$ are variables in $\mathbf{x_i}$, not necessarily distinct, and
  2. equalities of the form $t = t'$, where t and t' are terms built from the function symbols f_i and the variables $\mathbf{x}_j$.
- Each ψ_i is a conjunction of atomic formulas $S(t_1, \ldots, t_l)$, where S is an l-ary relation symbol of the schema $\mathbf{T}$ and $t_1, \ldots, t_l$ are terms built from the function symbols f_i and the variables $\mathbf{x}_j$.
- Each variable in $\mathbf{x_i}$ appears in a relational atomic formula of ϕ_i.

It is easy to see that the conjunction of finitely many SO tgds is logically equivalent to a single SO tgd. Note also that SO tgds subsume s-t tgds. As a matter of fact, every s-t tgd σ is equivalent to an SO tgd without equalities obtained by Skolemizing σ. Specifically, if σ is a s-t tgd of the form

$$\begin{aligned}&\forall x_1 \ldots \forall x_m(\varphi(x_1, \ldots, x_m) \rightarrow\\ &\qquad \exists y_1 \ldots \exists y_n \psi(x_1, \ldots, x_m, y_1, \ldots, y_n)),\end{aligned}$$

then σ is equivalent to the SO tgd

$$\begin{aligned}&\exists f_1 \ldots \exists f_n(\forall x_1 \ldots \forall x_m(\varphi(x_1, \ldots, x_m) \rightarrow\\ &\qquad \psi(x_1, \ldots, x_m, f_1(x_1, \ldots, x_m), \ldots, f_n(x_1, \ldots, x_m))).\end{aligned} \quad (1)$$

It follows that every finite set of s-t tgds is logically equivalent to a single SO tgd.

Next, consider the schema mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ used in the reduction from 3-Colorability. It is easy to verify that the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ is definable by the SO tgd

$$\exists f(\forall x \forall y(E(x,y) \rightarrow D(f(x), f(y)))). \quad (2)$$

Since this composition is not first-order definable, it follows that the above SO tgd is not logically equivalent to any finite set of s-t tgds. Thus, SO tgds properly subsume s-t tgds.

Finally, the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ of the schema mappings in Example 5.3 is definable by the SO tgd

$$\begin{aligned}&\exists f(\ \forall e(\mathtt{Emp}(e) \rightarrow \mathtt{Mgr}(e, f(e))) \wedge\\ &\qquad \forall e(\mathtt{Emp}(e) \wedge (e = f(e)) \rightarrow \mathtt{SelfMgr}(e))).\end{aligned} \quad (3)$$

As noted earlier, this composition is not definable by any set (finite or infinite) of s-t tgds, but is first-order definable.

Note that an equality between terms occurs in SO tgd (3), but not in SO tgds (1) and (2). In [15], it was shown that

allowing equalities in SO tgds is of the essence, because no SO tgd without equalities can define the composition of the two schema mappings in Example 5.3. This also suggests that formulating the concept of SO tgds was a rather delicate matter. For example, a naive way to arrive at second-order dependencies is to Skolemize s-t tgds. However, the fragment of existential second-order logic obtained this way lacks the expressive power to express the composition of finite sets of s-t tgds, since formulas in this fragment do not contain equalities.

As noted earlier, the conjunction of finitely many SO tgds is logically equivalent to a single SO tgd. Thus, instead of considering schema mappings specified by finite sets of SO tgds, it suffices to consider schema mappings specified by a single SO tgds. The next result from [15] asserts that such schema mappings are closed under composition.

THEOREM 5.5. SO *tgds are closed under composition, that is, if* $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ *and* $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ *are two schema mappings in which* Σ_{12} *and* Σ_{23} *are* SO *tgds, then there is an* SO *tgd* Σ_{13} *such that* $\mathcal{M}_{12} \circ \mathcal{M}_{23} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{13})$.

COROLLARY 5.6. *The composition of two finite sets of s-t tgds is definable by an* SO *tgd.*

In [15], an algorithm is presented for constucting an SO tgd that defines the composition of two given SO tgds. The algorithm runs in exponential time; it can be shown that this task requires exponential time. Also in [15], it is shown that, when it comes to data exchange, SO tgds have the same good algorithmic properties as s-t tgds.

THEOREM 5.7. *Let* $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ *be a schema mapping in which* Σ *is an* SO *tgd.*

- *There is a polynomial-time algorithm based on an extension of the chase procedure such that, given a source instance* I*, it tests whether a solution for* I *exists and, if so, it produces a universal solution* J *for* I.
- *If* q *is a union of conjunctive queries over* $\mathbf{T}$*, then there is a polynomial-time algorithm for computing the certain answers of* q.

The results presented in this section make a strong case that SO tgds possess the right balance between high expressive power and good algorithmic properties. On the one hand, they are powerful enough to express the composition of finite sets of s-t tgds, while being themselves closed under composition. On the other hand, they are amenable to the chase procedure, so that there are polynomial-time algorithms for constructing a universal solution and for computing the certain answers of unions of conjunctive queries.

Finally, note that the semantics of composition of schema mappings presented here has also been considered by Melnik [28] in his doctoral dissertation. In addition, Nash, Bernstein and Melnik [30] have recently investigated the composition of schema mappings specified by (first-order) embedded implicational dependencies that need not be s-t tgds.

6. Concluding Remarks

In this paper, we presented an overview of a body of work on data exchange and composition of schema mappings between relational schemas. This work was originally motivated by Clio, a schema mapping and data exchange system built at the IBM Almaden Research Center [29, 33]. Our initial goal was to provide a formal justification for certain engineering choices made in Clio; this led to the formulation of the concept of universal solutions and to the subsequent investigation of foundational and algorithmic issues in data exchange and metadata management. Some of the findings of this investigation influenced the development of Clio, which, in the meantime, has evolved from a prototype to an industrial-strength tool [19]. In particular, SO tgds form the core of Clio's Mapping Specification Language; moreover, the composition algorithm for SO tgds has been incorporated in Clio.

It should be noted that Clio actually supports data exchange between XML and relational schemas, in all four combinations. Many conceptual and technical challenges arise in the study of the foundations of data exchange and metadata management for XML schemas. Arenas and Libkin [3] have already addressed some of the challenges in data exchange between XML schemas.

Acknowledgments The work reported here is the result of a fruitful and enjoyable collaboration with my colleagues Ronald Fagin, Renée J. Miller, Lucian Popa, and Wang-Chiew Tan. I am grateful to Ronald Fagin for reading earlier versions of this paper and offering numerous valuable comments, corrections, and suggestions for improvement.

7. References

[1] S. Abiteboul and O. M. Duschka. Complexity of Answering Queries Using Materialized Views. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 254–263, 1998.

[2] K. R. Apt. Logic programming. In J. van Leeuween, editor, *Handbook of Theoretical Computer Science*, volume B, pages 493–474. The MIT Press/Elsevier, 1990.

[3] M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[4] C. Beeri and M. Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the Association for Computing Machinery (JACM)*, 31(4):718–741, 1984.

[5] C. Beeri and M. Y. Vardi. Formal Systems for Tuple and Equality Generating Dependencies. *SIAM J. on Computing*, 13(1):76–98, 1984.

[6] P. A. Bernstein. Applying Model Management to Classical Meta-Data Problems. In *Conference on Innovative Data Systems Research (CIDR)*, pages 209–220, 2003.

[7] P. A. Bernstein. Generic Model Management: A Database Infrastructure for Schema Manipulation. Keynote Address, IDM 2003 Workshop, Seattle, Washington, September 2003.

[8] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *ACM Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

[9] E. F. Codd. Further normalization of the data base relational model. In R. Rustin, editor, *Data Base Systems*, pages 33–64. Prentice-Hall, 1972.

[10] S. S. Cosmadakis and P. C. Kanellakis. Functional and Inclusion Dependencies: A Graph Theoretic Approach. In *Advances in Computing Research*, volume 3, pages 163–184. JAI Press, 1986.

[11] A. Dawar. A Restricted Second Order Logic for Finite Structures. *Information and Computation*, 143(2):154–174, 1998.

[12] R. Fagin. Horn Clauses and Database Dependencies. *Journal of the Association for Computing Machinery (JACM)*, 29(4):952–985, Oct. 1982.

[13] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *International Conference on Database Theory (ICDT)*, pages 207–224, 2003. Full version to appear in a Special Issue of *Theoretical Computer Science* with selected papers from ICDT 2003.

[14] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 90–101, 2003. Full version invited to *ACM Transactions on Database Systems* (TODS).

[15] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 83–94, 2004. Full version invited to *ACM Transactions on Database Systems* (TODS).

[16] T. Feder and M. Y. Vardi. The Computational Structure of Monotone Monadic SNP and Constraint Satisfaction: A Study through Datalog and Group Theory. *SIAM J. on Computing*, 28:57–104, 1998. Preliminary version in *Proc. 25th ACM Symp. on Theory of Computing*, May 1993, pp. 612–622.

[17] A. Fuxman, P. G. Kolaitis, R. Miller, and W.-C. Tan. Peer Data Exchange. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[18] G. Gottlob. Computing Cores for Data Exchange: New Algorithms and Practical Solutions. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[19] L. Haas, M. Hernandez, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2005.

[20] P. Hell and J. Nešetřil. The Core of a Graph. *Discrete Mathematics*, 109:117–126, 1992.

[21] P. Hell and J. Nešetřil. *Graphs and Homomorphisms.* Oxford University Press, 2004.

[22] P. C. Kanellakis. Elements of Relational Database Theory. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics*, pages 1073–1156. Elsevier and MIT Press, 1990.

[23] M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[24] T. Luczak and J. Nešetřil. A probabilistic approach to the dychotomy problem. Technical Report 640, Charles University, Prague, 2003.

[25] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *International Conference on Very Large Data Bases (VLDB)*, pages 572–583, 2003.

[26] D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing Implications of Data Dependencies. *ACM Transactions on Database Systems (TODS)*, 4(4):455–469, Dec. 1979.

[27] A. Mądry. Data exchange: on complexity of answering queries with inequalities. *Information Processing Letters*, 2005. In press.

[28] S. Melnik. *Generic Model Management: Concepts and Algorithms.* Lecture Notes in Computer Science 2967. Springer, 2004.

[29] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *International Conference on Very Large Data Bases (VLDB)*, pages 77–88, 2000.

[30] A. Nash, P. A. Bernstein, and S. Melnik. Composition of mappings given by embedded dependencies. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[31] C. Papadimitriou and M. Yannakakis. The Complexity of Facets and Some Facets of Complexity. In *ACM Symposium on Theory of Computing (STOC)*, pages 229–234, 1982.

[32] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[33] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 598–609, 2002.

[34] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *ACM Transactions on Database Systems (TODS)*, 2(2):134–174, 1977.

[35] R. van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

[36] M. Y. Vardi. The complexity of relational query languages. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 137–146, 1982.