Nested Dependencies: Structure and Reasoning

Phokion G. Kolaitis
UC Santa Cruz & IBM
kolaitis@cs.ucsc.edu

Reinhard Pichler
TU Vienna
pichler@dbai.tuwien.ac.at

Emanuel Sallinger
TU Vienna
sallinger@dbai.tuwien.ac.at

Vadim Savenkov
TU Vienna
savenkov@dbai.tuwien.ac.at

ABSTRACT

During the past decade, schema mappings have been extensively used in formalizing and studying such critical data interoperability tasks as data exchange and data integration. Much of the work has focused on GLAV mappings, i.e., schema mappings specified by source-to-target tuple-generating dependencies (s-t tgds), and on schema mappings specified by second-order tgds (SO tgds), which constitute the closure of GLAV mappings under composition. In addition, nested GLAV mappings have also been considered, i.e., schema mappings specified by nested tgds, which have expressive power intermediate between s-t tgds and SO tgds.

Even though nested GLAV mappings have been used in data exchange systems, such as IBM's Clio, no systematic investigation of this class of schema mappings has been carried out so far. In this paper, we embark on such an investigation by focusing on the basic reasoning tasks, algorithmic problems, and structural properties of nested GLAV mappings. One of our main results is the decidability of the implication problem for nested tgds. We also analyze the structure of the core of universal solutions with respect to nested GLAV mappings and develop useful tools for telling apart SO tgds from nested tgds. By discovering deeper structural properties of nested GLAV mappings, we show that also the following problem is decidable: given a nested GLAV mapping, is it logically equivalent to a GLAV mapping?

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*

General Terms

Theory, Languages, Algorithms

Keywords

Schema mappings, data integration, data exchange, nested dependencies, second-order dependencies

1. Introduction

Schema mappings are high-level specifications, typically expressed in some logical formalism, that describe the relationship between two database schemas, called the source schema and the target schema. During the past decade, schema mappings have been extensively used in formalizing and studying such critical data interoperability tasks as data exchange and data integration. Much of the work has focused on two classes of schema mappings: GLAV mappings and mappings specified by SO tgds. A GLAV mapping is specified by a finite set of source-to-target tuple-generating dependencies (s-t tgds), which are first-order formulas of the form $\forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\,\psi(\vec{x},\vec{y}))$ with $\varphi(\vec{x})$ a conjunction of atoms over the source schema and $\psi(\vec{x},\vec{y})$ a conjunction of atoms over the target schema. As the name suggests, a second-order tuple-generating dependency (SO tgd) is a second-order formula; it starts with a string of existential function quantifiers that is followed by a conjunction of first-order formulas that resemble s-t tgds, but allow function terms in atomic formulas and also equalities between such terms. As shown in [8], SO tgds are the *right* language for expressing the composition of GLAV schema mappings. The study of GLAV mappings and mappings specified by SO tgds has spanned a wide range of problems, from expressive power and algorithms to optimization and structural properties; for recent overviews of the literature, see [1, 13].

In addition to GLAV mappings and mappings specified by SO tgds, two other classes of schema mappings of intermediate expressive power have also been considered. The first is the class of nested GLAV mappings that are specified by finitely many nested tgds, that is, first-order formulas that, informally, are obtained by a finite "nesting" of s-t tgds inside other s-t tgds. For example, the expression

$$\forall x_1 x_2 (S(x_1,x_2) \rightarrow \exists y\,(S(y,x_2) \land \\ \forall x_3(S(x_1,x_3) \rightarrow R(y,x_3))))$$

is a nested tgd. The second is the class of plain SO tgds, which consists of those SO tgds that contain no nested terms (i.e., no functional terms that have other functional terms as arguments) and no equalities between terms. For example, the expression

$$\exists f\,\forall x \forall y\,(S(x,y) \rightarrow R(f(x),f(y)))$$

is a plain SO tgd. We now describe the different reasons and motivation that led to the introduction of these two classes of schema mappings.

Nested GLAV mappings were introduced in [10] and demonstrated in [12] as an enhancement of the specification language of the Clio system, which, at that time, was being developed at the IBM Almaden Research Center, and is now part of IBM's InfoSphere BigInsights suite. Clio is a system that supports both the automatic or semi-automatic derivation of schema mappings from a visual specification and the subsequent generation of executable transformations for exchanging data between source and target. The main argument in favor of nested GLAV mappings over GLAV mappings given in [10, 12] is that they produce specifications that are more compact and also reflect more accurately the correlations between data; moreover, since they are specified in first-order logic, nested GLAV mappings give rise to transformations that, like those arising from GLAV mappings, can be implemented using SQL queries.

Plain SO tgds were introduced and studied in depth quite recently in [2] with a very different motivation in mind. Specifically, the goal was to find a "good" language for handling both composition and inversion of GLAV mappings. The results in [2] make a strong case that plain SO tgds form the *right* language for handling CQ-composition and inversion of GLAV mappings, where CQ-composition is a variant of the composition operator in which two schema mappings are considered to be equivalent if they give rise to the same certain answers for conjunctive queries (the notion of CQ-composition was introduced in [16]).

In terms of expressive power, nested GLAV mappings are strictly more expressive than GLAV mappings and strictly less expressive than mappings specified by plain SO tgds. As a matter of fact, it is known that the nested tgd given earlier is not logically equivalent to any finite set of s-t tgds, while the plain SO tgd given earlier is not logically equivalent to any finite set of nested tgds. Nested GLAV mappings and plain SO tgds share several desirable structural properties, such as admitting universal solutions and being closed under target homomorphisms [17, 2]. These similarities notwithstanding, it should be kept in mind that nested tgds and SO tgds belong to intrinsically different logical formalisms (first-order logic vs. second-order logic), a fact that may translate to different algorithmic behavior. For instance, the data complexity of the model checking problem of nested tgds is in LOGSPACE, while the data complexity of plain SO tgds is NP-complete.

Even though nested GLAV mappings were introduced several years ago and were incorporated into data exchange systems, no systematic investigation of nested tgds in their own right has been carried out to date. Our goal in this paper is to embark on such an investigation by focusing on the basic reasoning tasks, algorithmic problems, and deeper structural properties of nested GLAV mappings. Our first main result is that the implication problem (and, hence, the equivalence problem) for nested tgds is decidable. This should be contrasted with the state of affairs for SO tgds, for which the logical equivalence problem (hence also the implication problem) is undecidable. In fact, there is no algorithm even for deciding whether a given SO tgd is logically equivalent to a given finite set of s-t tgds, see [3, 9]. As for plain SO tgds, it is not known whether the implication problem and the logical equivalence problem are decidable.

Our decision procedure for the implication problem for nested tgds is rather elaborate and entails a delicate analysis of the properties of the chase procedure for nested tgds.

After this, we address the problem of telling apart nested tgds from s-t tgds. To that end, we show that the following problem is decidable: given a nested GLAV mapping, is it logically equivalent to some GLAV mapping? The situation is less clear regarding the problem of telling apart plain SO tgds from nested tgds. Indeed, at present, it is not known whether or not the following problem is decidable: given a plain SO tgd, is it logically equivalent to some nested GLAV mapping? Even though we do not settle the decidability of this problem here, we succeed in providing useful and easy-to-use sufficient conditions for telling that a given plain SO tgd is not logically equivalent to a nested GLAV mapping.

The aforementioned algorithm for telling apart nested tgds from s-t tgds, as well as the aforementioned sufficient conditions for telling apart a plain SO tgd from nested tgds, are derived by analyzing the structure of the cores of universal solutions with respect to nested GLAV mappings. In carrying out this analysis, we discover several deeper properties of nested tgds that enable their comparison with both s-t tgds and plain SO tgds. We believe that these properties are of interest in their own right and may play a role in structural characterizations of schema-mapping languages.

Finally, we study settings where key dependencies or, more generally, equality generating dependencies (egds) over the source schema are present. By revisiting the fundamental decision problems of logical equivalence and of telling apart schema mappings in different formalisms, we unveil further significant differences between nested tgds and plain SO tgds. In [9], the logical equivalence problem for plain SO tgds was shown undecidable if the source schema contains key dependencies. In contrast, here we show that the implication problem (and, hence, the logical equivalence problem) for nested tgds remains decidable even in the presence of arbitrary source egds. Likewise, we show that the problem of deciding if a given nested GLAV mapping is logically equivalent to some GLAV mapping remains decidable if arbitrary source egds are allowed. Again, this is in sharp contrast to plain SO tgds, for which we prove undecidability of the following problems in the presence of source key dependencies: given a plain SO tgd, is it logically equivalent to a GLAV mapping (or to a nested GLAV mapping, respectively)?

The remainder of the paper is organized as follows. Section 2 contains the definitions of the basic concepts and background material. Section 3 is devoted to the implication problem for nested tgds. Section 4 contains the analysis of the core of the universal solutions with respect to nested GLAV mappings, and the applications of this analysis to differentiating nested tgds from s-t tgds, and also to differentiating plain SO tgds from nested tgds. Section 5 revisits the problems studied in earlier sections when source key constraints are also present in the specification of the schema mappings at hand. Finally, the paper concludes with a discussion of open problems and directions for future research.

2. Preliminaries

Schemas, Instances, and Homomorphisms. A *schema* $\mathbf{R}$ is a finite sequence $\langle R_1, \ldots, R_k \rangle$ of relation symbols, where each R_i has a fixed arity. An *instance* I over $\mathbf{R}$, or an $\mathbf{R}$-*instance*, is a sequence $(R_1^I, \ldots, R_k^I)$, where each R_i^I is a finite relation of the same arity as R_i. We will often use R_i to denote both the relation symbol and the relation R_i^I that instantiates it. A *fact* of an instance I (over $\mathbf{R}$) is an expression $R_i^I(v_1,\ldots,v_m)$ (or simply $R_i(v_1,\ldots,v_m)$), where R_i is a relation symbol of $\mathbf{R}$ and $(v_1,\ldots,v_m) \in R_i^I$.

Let $\mathbf{S}$ and $\mathbf{T}$ be two schemas with no relation symbols in common. We refer to $\mathbf{S}$ as the *source schema*, and $\mathbf{T}$ as the *target schema*. Similarly, we refer to $\mathbf{S}$-instances as *source instances*, and $\mathbf{T}$-instances as *target instances*. We assume the presence of two kinds of values, namely *constants* and *(labeled) nulls*. We also assume that the active domains of source instances consists of constants; the active domains of target instances may consist of constants and nulls.

Let J be a target instance. The *Gaifman graph of facts* of J is the graph whose nodes are the facts of J and there is an edge between two facts if they have a null in common. We say that a target instance J is *connected* if the Gaifman graph of facts of J is connected. A *fact block (f-block)* of J is a connected component of the Gaifman graph of facts of J. The *fact block size (f-block size)* of J is the maximum cardinality of the f-blocks of J.

Let J_1 and J_2 be two target instances. A function h is a *homomorphism* from J_1 to J_2 if the following hold: (i) for every constant c, we have that $h(c) = c$; and (ii) for every relation symbol R in $\mathbf{R}$ and every tuple $(a_1,\ldots,a_n) \in R^{J_1}$, we have that $(h(a_1),\ldots,h(a_n)) \in R^{J_2}$. We use the notation $J_1 \to J_2$ to denote that there is a homomorphism from J_1 to J_2. We say that J_1 is *homomorphically equivalent* to J_2, written $J_1 \leftrightarrow J_2$, if $J_1 \to J_2$ and $J_2 \to J_1$. The *core* of an instance J, denoted *core*(J), is the smallest subinstance of J that is homomorphically equivalent to J. If there are multiple cores of J, then they are all isomorphic [11].

Schema mappings. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S},\mathbf{T},\Sigma)$, where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, and Σ is a set of constraints (typically, formulas in some logic) that describe the relationship between $\mathbf{S}$ and $\mathbf{T}$. We say that $\mathcal{M}$ is *specified by* Σ; often, we will use the set Σ of constraints to denote the mapping $\mathcal{M}$ specified by Σ.

If I is a source instance and J is a target instance such that the pair (I,J) satisfies Σ (written $(I,J) \models \Sigma$), then we say that J is a *solution* of I w.r.t. $\mathcal{M}$. We say that J is a *universal solution for I w.r.t. $\mathcal{M}$* if J is a solution for I and for every solution J' for I, we have $J \to J'$. If $\mathcal{C}$ is a class of schema mappings, we say that $\mathcal{C}$ *admits universal solutions* if for every schema mapping $\mathcal{M}$ in $\mathcal{C}$ and every source instance I, a universal solution for I w.r.t. $\mathcal{M}$ exists.

s-t tgds. A *source-to-target tuple-generating dependency* (in short, *s-t tgd*) is a first-order sentence of the form $\forall\vec{x}(\varphi(\vec{x}) \to \exists\vec{y}\psi(\vec{x},\vec{y}))$, where $\varphi(\vec{x})$ is a conjunction of atoms over $\mathbf{S}$, each variable in $\vec{x}$ occurs in at least one atom in $\varphi(\vec{x})$, and $\psi(\vec{x},\vec{y})$ is a conjunction of atoms over $\mathbf{T}$ with variables in $\vec{x}$ and $\vec{y}$. For simplicity, we will often suppress writing the universal quantifiers $\forall\vec{x}$ in the above formula. Another name for s-t tgds is *global-and-local-as-view* (GLAV) constraints (see [14]). We refer to a schema mapping specified entirely by a finite set of GLAV constraints as a *GLAV mapping.* As shown in [5], the class of GLAV mappings admits universal solutions. Moreover, if $\mathcal{M}$ is a GLAV mapping, then given a source instance I, a canonical universal solution *chase*$(I,\mathcal{M})$ can be produced via the *oblivious chase procedure.* That is, whenever the antecedent of an s-t tgd in $\mathcal{M}$ becomes true, fresh null values are introduced and facts involving these nulls are added to *chase*$(I,\mathcal{M})$ so that the conclusion of the s-t tgd becomes true.

SO tgds and Plain SO tgds. *Second-Order tgds*, or *SO tgds*, were introduced in [8], where it was shown that SO tgds are exactly the dependencies needed to specify the composition of an arbitrary number of GLAV mappings. Before we formally define SO tgds, we need to define *terms*. Given collections $\vec{x}$ of variables and $\vec{f}$ of function symbols, a *term (based on $\vec{x}$ and $\vec{f}$)* is defined recursively as follows: (1) Every variable in $\vec{x}$ is a term; (2) If f is a k-ary function symbol in $\vec{f}$ and $t_1,\ldots,t_k$ are terms, then $f(t_1,\ldots,t_k)$ is a term.

Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ a target schema. A *second-order tuple-generating dependency (SO tgd)* is a formula of the form:

$$\exists\vec{f}((\forall\vec{x}_1(\varphi_1 \to \psi_1)) \wedge ... \wedge (\forall\vec{x}_n(\varphi_n \to \psi_n))), \text{where}$$

(1) Each member of $\vec{f}$ is a function symbol. (2) Each φ_i is a conjunction of (i) atoms $S(y_1,...,y_k)$, where S is a k-ary relation symbol of schema $\mathbf{S}$ and $y_1,\ldots,y_k$ are variables in $\vec{x}_i$, not necessarily distinct, and (ii) equalities of the form $t = t'$ where t and t' are terms based on $\vec{x}_i$ and $\vec{f}$. (3) Each ψ_i is a conjunction of atoms $T(t_1,...,t_l)$, where T is an l-ary relation symbol of schema $\mathbf{T}$ and $t_1,\ldots,t_l$ are terms based on $\vec{x}_i$ and $\vec{f}$. (4) Each variable in $\vec{x}_i$ appears in some atom formula of φ_i. As an example, the formula

$$\begin{aligned}&\exists f(\forall e(\mathit{Emp}(e) \to \mathit{Mgr}(e,f(e))) \ \wedge \\ &\qquad \forall e(\mathit{Emp}(e) \wedge (e = f(e)) \to \mathit{SelfMgr}(e)))\end{aligned}$$

expresses the property that every employee has a manager, and if an employee is the manager of himself/herself, then this employee is a self-manager.

Note that SO tgds allow for nested terms and for equalities between terms. A nested term is a functional term which contains a functional term as an argument. A *plain SO tgd* is an SO tgd that contains no nested terms and no equalities. For example, the preceding SO tgd is not plain, while the following SO tgd is plain

$$\exists f\forall x\forall y(S(x,y) \to R(f(x),f(y)))$$

The properties of plain SO tgds were recently investigated in [2]. It is easy to see that every GLAV schema mapping is logically equivalent to a plain SO tgd. Moreover, as shown in [8], the class of SO tgds admits universal solutions, hence the same holds true for the class of plain SO tgds. In fact, the chase procedure can be extended to SO tgds, so that if σ is an SO tgd and I is a source instance, then *chase*(I,σ) is a canonical universal solution for I w.r.t. σ.

In what follows, we will often suppress writing the existential second-order quantifiers and the universal first-order quantifiers in front of SO tgds.

Nested tgds. Fix a partition of the set of first-order variables into two disjoint infinite sets, X and Y. A *nested tgd* is a first-order sentence that can be generated by the following recursive definition:

$$\chi ::= \alpha \mid \forall\vec{x}\,(\beta_1 \wedge \ldots \wedge \beta_k \to \exists\vec{y}\,(\chi_1 \wedge \ldots \wedge \chi_\ell))$$

where each $x_i \in X$, each $y_i \in Y$, α is an atomic formula over the target schema, and each β_j is an atomic formula over the source schema containing only variables from X, such that each x_i occurs in some β_j. As an example, the formula

$$\forall x_1 x_2 (S(x_1,x_2) \rightarrow \exists y\, (R(y,x_2) \wedge \\ \forall x_3 (S(x_1,x_3) \rightarrow R(y,x_3))))$$

is a nested tgd. It is known that this nested tgd is not logically equivalent to any finite set of s-t tgds (see [8, 17]).

A nested tgd σ contains a number of *parts* σ_i. Informally, σ_i is an implicational formula that corresponds to the recursive option of the production rule for χ, where each χ_i is the conjunction of atoms given by the non-recursive option. Thus, σ_i is syntactically similar to an s-t tgd, but may have free variables; moreover, the conclusion may be an empty conjunction, in which case it evaluates to $\top$ (true). As an example, the parts of the preceding nested tgd are

- $\forall x_1 x_2 (S(x_1,x_2) \rightarrow \exists y\, R(y,x_2))$ and
- $\forall x_3 (S(x_1,x_3) \rightarrow R(y,x_3))$

In our examples, we refer to parts of nested tgds using labels. The way of inline labeling of parts which we use throughout this paper is illustrated below by a nested tgd σ with four parts $\sigma_1, \ldots, \sigma_4$:

$$\begin{array}{ll}
\sigma_1: & \forall x_1 \big(S_1(x_1) \rightarrow \exists y_1 \qquad\qquad\qquad (*) \\
\sigma_2: & \quad \big(\forall x_2 (S_2(x_2) \rightarrow R_2(y_1,x_2)) \wedge \\
\sigma_3: & \quad\; \forall x_3 (S_3(x_1,x_3) \rightarrow (R_3(y_1,x_3) \wedge \\
\sigma_4: & \qquad\quad \forall x_4 (S_4(x_3,x_4) \rightarrow \exists y_2 R_4(y_2,x_4))))\big)
\end{array}$$

For $i > 1$, by $parent(\sigma_i)$ we denote the part where σ_i is nested: for instance, for the above dependency σ, we have $parent(\sigma_2) = parent(\sigma_3) = \sigma_1$, and $parent(\sigma_4) = \sigma_3$. The ancestors $anc(\sigma_i)$ of σ_i are defined via the transitive closure of *parent*. For example $anc(\sigma_4) = \{\sigma_1, \sigma_3\}$. Symmetrically, we define $child(\sigma_i)$ to be the set of parts nested directly under (σ_i). For example, we have $child(\sigma_1) = \{\sigma_2, \sigma_3\}$ for the dependency σ above. Again, we define the descendants $desc(\sigma_i)$ to be the transitive closure of *child*, e.g., $desc(\sigma_1) = \{\sigma_2, \sigma_3, \sigma_4\}$.

It is sometimes convenient to consider *Skolemized nested tgds*, in which every existential variable y is replaced by the Skolem term $f(\vec{x})$ where f is a fresh function symbol and $\vec{x}$ is the vector of universally quantified variables in the part σ_i in which $\exists y$ occurs, and in the ancestors of σ_i. Note that we assume existential variables in different parts to be renamed apart. The Skolemized version of σ has the following form:

$$\begin{array}{ll}
\sigma_1: & \forall x_1 \big(S_1(x_1) \rightarrow \\
\sigma_2: & \quad \big(\forall x_2 (S_2(x_2) \rightarrow R_2(f(x_1),x_2)) \wedge \\
\sigma_3: & \quad\; \forall x_3 (S_3(x_1,x_3) \rightarrow (R_3(f(x_1),x_3) \wedge \\
\sigma_4: & \qquad\quad \forall x_4 (S_4(x_3,x_4) \rightarrow R_4(g(x_1,x_3,x_4),x_4))))\big)
\end{array}$$

We write $\forall \vec{x} (\varphi(\vec{x},\vec{x}_0) \rightarrow \psi(\vec{x},\vec{x}_0))$ to express a part σ_i of a Skolemized nested tgd, where $\vec{x}_0$ is the vector of universally quantified variables stemming from $anc(\sigma_i)$. For instance, the part σ_4 can be expressed as $\forall \vec{x} (\varphi_4(\vec{x},\vec{x}_0) \rightarrow \psi_4(\vec{x},\vec{x}_0))$ where $\vec{x} = \langle x_4 \rangle$ and $\vec{x}_0 = \langle x_1, x_3 \rangle$.

It is easy to see that every Skolemized nested tgd is a plain SO tgd. Thus, the class of nested tgds contains the class of s-t tgds and is contained in the class of plain SO tgds. A *nested GLAV mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of nested tgds.

3. The Implication Problem

Let Σ and Σ' be two finite sets of source-to-target constraints expressed in some logical formalism (e.g., SO tgds, nested tgds, s-t tgds). We say that Σ implies Σ', denoted by $\Sigma \models \Sigma'$, if for every source instance I and every target instance J such that $(I,J) \models \Sigma$, we have that $(I,J) \models \Sigma'$. The *implication problem* asks: given two finite sets Σ and Σ' of source-to-target constraints, does $\Sigma \models \Sigma'$ hold? Analogously, the *(logical) equivalence problem* asks if $\Sigma \equiv \Sigma'$ holds, i.e., if Σ and Σ' are satisfied by exactly the same pairs (I,J) of source and target instances.

Note that, since all instances considered are finite, this is the implication (and equivalence) problem *in the finite*. The main result of this section is as follows.

Theorem 3.1 *The implication problem for nested tgds is decidable.*

The decision procedure behind Theorem 3.1 requires the introduction of several key concepts and the development of new technical tools that are presented in what follows.

Chase Forest. We begin by introducing the *chase forest* of a nested tgd, which represents the process of chasing a source instance with a nested tgd.

Let σ be a nested tgd and I a source instance. The oblivious chase $chase(I,\sigma)$ of I with σ can be described as a sequence of recursive *triggerings*: Each triggering t is associated with a part $\sigma_i : \forall \vec{x} \varphi(\vec{x},\vec{x}_0) \rightarrow \psi(\vec{x},\vec{x}_0)$ and with the variable assignment $\vec{a}$ to the variables in $\vec{x}$.

If the part σ_i is the top-level part of σ, then the vector $\vec{x}_0$ is empty and t is called a *root triggering*. Otherwise, the triggering t of σ_i has a unique *parent triggering* t', associated with the part $parent(\sigma_i)$ and the partial assignment to the variables in $\vec{x}_0$. The transitive closure over parent triggerings gives the set of *ancestor triggerings* of t. All variables in $\vec{x}_0$ are bound in ancestor triggerings. The corresponding assignment $\vec{a}_0$ for $\vec{x}_0$ is called the *input assignment* of t.

Let $\forall \vec{x} (\varphi(\vec{x},\vec{x}_0) \rightarrow \psi(\vec{x},\vec{x}_0))$ be a part σ_i of σ, and suppose that there exists a chain of triggerings of parts in $anc(\sigma_i)$ that has bound all variables in $\vec{x}_0$ yielding an assignment $\vec{a}_0$. A necessary and sufficient condition to activate the triggering t of σ_i is $I \models \varphi(\vec{a},\vec{a}_0)$, for some assignment $\vec{a}$ for $\vec{x}$. The result of t is the instantiation $\psi(\vec{a},\vec{a}_0)$ of the conclusion atoms of σ_i, which are then added to the instance $chase(I,\sigma)$. In this instantiation, Skolem terms are considered as null labels. The parts in $child(\sigma_i)$ are then triggered recursively. The set of all triggerings recursively called from t is denoted by $rec(t)$.

The collection of the triggerings in the chase of I with a finite set Σ of nested tgds constitutes the *chase forest of I with σ*: root triggerings t_r are associated with the top-level parts of nested tgds in Σ, and each set of triggerings called recursively from t_r constitutes the *chase tree* rooted at t_r.

An immediate consequence of the definition of the chase is that triggerings in distinct chase trees produce facts which share no nulls. This is one of the key underpinnings of our decidability result: namely, reasoning about nested tgds may be restricted to source instances that give rise to a single chase tree. The second underpinning is that only chase trees of bounded fanout need to be considered, as we explain next.

Patterns. The algorithm behind Theorem 3.1 is described in the displayed decision procedure Implies. We now introduce the notions used in this procedure.

Definition 3.2 (Pattern) Let σ be a nested tgd and $\mathcal{T}$ a chase tree of some source instance I with σ.

A *pattern* of σ is a tree whose nodes are labeled by identifiers of tgd parts in such a way that the parent-child re-

```
Procedure IMPLIES(Σ,σ)
  Data: Set Σ of nested tgds, nested tgd σ
  Result: true if Σ ⊨ σ, false otherwise
1  Skolemize σ and Σ in a standard way;
2  Let $v_\sigma$ be the number of distinct Skolem functions in σ;
3  Let $w_\Sigma$ be the maximum number of universally
   quantified variables in a nested tgd in Σ;
4  Let $k = v_\sigma \cdot w_\Sigma + 1$;
5  Let $\mathcal{P}_k(\sigma)$ be the set of k-patterns of σ;
6  for each k-pattern $p_k \in \mathcal{P}_k(\sigma)$ do
7      Produce $I_{p_k}$ and $J_{p_k}$, the canonical source and,
       respectively, the canonical target instances of $p_k$;
8      if no homomorphism $J_{p_k} \to chase(I_{p_k}, \Sigma)$ exists
       then
9          return false;
10     end
11 end
12 return true;
```

lationship between nodes coincides with the nesting of the tgd parts at the labels of respective nodes.

The *pattern of chase tree* $\mathcal{T}$ is the tree obtained from $\mathcal{T}$ by ignoring the assignments to the universal variables and using solely the identifiers of tgd parts as node markers. ◁

We also use the notion of subtree in a pattern and the "cloning" operation on subtrees, defined in an intuitive way.

Definition 3.3 (Subtree, Cloning, k-pattern) Let σ be a nested tgd and let p be a pattern of σ. By a *subtree* of a pattern p, we always mean a subtree closed under child relation. That is, there is a single subtree rooted at each node n, namely the one containing all descendant nodes of n.

A subtree t' is called a *clone* of a subtree t if the roots of t' and t are siblings in p and the two subtrees are isomorphic. Appending a clone of a subtree t to the parent of its root node is called *cloning* t.

Let $\mathcal{C}_t$ denote the set of all clones of t in p. If for each subtree t, $|\mathcal{C}_t| \leq k$, we call p a *k-pattern*. The set of all k-patterns of σ is denoted $\mathcal{P}_k(\sigma)$. ◁

Example 3.4 We point out that not every pattern of a nested tgd can be realized in a chase forest. Consider the nested tgd $\forall x_1\ S_1(x_1) \to ((S_2(x_1) \to T_2(x_1))$ with a single nested part. This tgd can only generate chase trees with patterns having two nodes. This is because the assignment of the only variable x_1 is determined by the root triggering and thus only a single triggering of the nested part is possible, using the same assignment. ◁

Ignoring realizability of patterns simplifies the presentation of the decision procedure and can be shown not to affect its correctness.

We now show how to enumerate k-patterns. We identify trees with the pairs $\langle \sigma_j, \mathcal{T}^\mu \rangle$ where σ_j is a part of σ associated with the root of the tree and $\mathcal{T}^\mu$ is a *multiset* of subtrees nested under σ_j, given by the set $\mathcal{T}$ of distinct subtrees and the multiplicity function $\mu : \mathcal{T} \to 1 \ldots k$. We now define the set $\mathcal{P}^*_k(\sigma_j)$ associated with a part σ_j of σ as follows:

- If *child*(σ_j) is empty, define $\mathcal{P}^*_k(\sigma_j) = \{\langle \sigma_j, \emptyset \rangle\}$.
- Otherwise, assume *child*$(\sigma_j) = \{\sigma_{i_1}, \ldots \sigma_{i_\ell}\}$; define

$$\mathcal{P}^*_k(\sigma_j) = \{\langle \sigma_j, \textstyle\bigcup_{\alpha=1}^{\ell} P_\alpha^{\mu_\alpha} \rangle \mid P_\alpha \subseteq \mathcal{P}^*_k(\sigma_{i_\alpha}) \text{ and } \mu_\alpha \text{ is a function } P_\alpha \to 1 \ldots k\}.$$

The inductive step of the above definition constructs the set of all possible trees rooted at a node labeled with σ_j and having as subtrees at most k clones of some trees in $\mathcal{P}^*(\sigma_{i_\alpha})$, for every part σ_{i_α} nested at σ_j.

Proposition 3.5 *Let k be an integer, σ be a nested tgd and let σ_1 be the top-level part of σ. Then, the set $\mathcal{P}_k(\sigma)$ of k-patterns of σ coincides with $\mathcal{P}^*_k(\sigma_1)$.*

Example 3.6 Recall the nested tgd σ with four parts from Section 2 marked with (*). The set $\mathcal{P}_1(\sigma) = \{p_1, \ldots, p_8\}$ containing 1-patterns of σ is shown in Figure 1. ◁

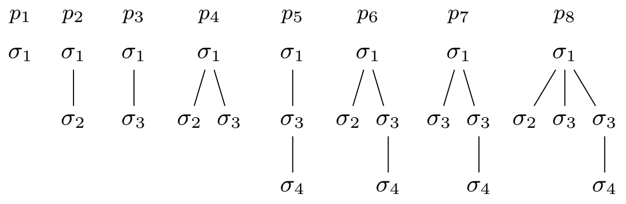


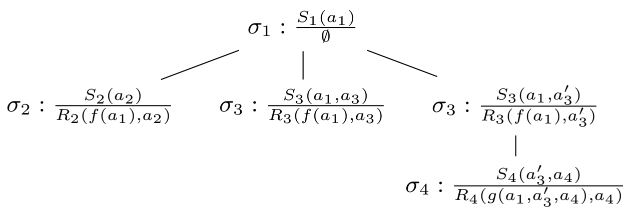
Figure 1: 1-patterns of the tgd σ from Section 2.

It follows from the definition of the set $\mathcal{P}^*_k(\sigma_j)$ used in Proposition 3.5 that for each fixed k, the maximum size of k-patterns in $\mathcal{P}_k(\sigma)$ is bounded, albeit non-elementary in the nesting depth of σ, and so is the size of $\mathcal{P}_k(\sigma)$ itself.

The only missing component for the decision procedure IMPLIES is now the notion of the canonical instance of a pattern. It is defined next.

Definition 3.7 (Canonical instances of a pattern) Let p be a pattern representing a sequence of recursive triggerings of a nested tgd σ. The *canonical source instance I_p of p* and the *canonical target instance J_p of p* are obtained by adding, for each node associated with the part $\sigma_i : \forall \vec{x}(\varphi(\vec{x}, \vec{x}_0) \to \psi(\vec{x}, \vec{x}_0))$, the atoms of $\varphi(\vec{a}, \vec{a}_0)$ to I_p and the atoms of $\psi(\vec{a}, \vec{a}_0)$ to J_p, where $\vec{a}$ assigns distinct fresh constants to the variables of $\vec{x}$, and $\vec{a}_0$ is the assignment used to instantiate the parts in *anc*(σ_i). ◁

Note that we speak of *the* canonical source and target instances, even though the constants used to create them can be arbitrary. The justification is that such instances are unique up to renaming of constants and thus are indistinguishable for nested tgds, which have no constants, according to the definition in Section 2.



$\sigma_1 : \frac{S_1(a_1)}{\emptyset}$
$\sigma_2 : \frac{S_2(a_2)}{R_2(f(a_1),a_2)}$
$\sigma_3 : \frac{S_3(a_1,a_3)}{R_3(f(a_1),a_3)}$
$\sigma_3 : \frac{S_3(a_1,a'_3)}{R_3(f(a_1),a'_3)}$
$\sigma_4 : \frac{S_4(a'_3,a_4)}{R_4(g(a_1,a'_3,a_4),a_4)}$

Figure 2: Facts of the canonical source instance I_{p_8} (above the bars) and the canonical target instance J_{p_8} (below the bars) of the pattern p_8.

Example 3.8 The canonical source instance and the canonical target instance of the 1-pattern p_8 from Example 3.6 are respectively I_{p_8} and J_{p_8} arranged in a tree in Figure 2. $\triangleleft$

The next example shows the canonical source instance of a pattern containing clones of subtrees.

Example 3.9 One possible 3-pattern based on the 1-pattern p_8 from Example 3.8, in which one clone of the node σ_2 and two clones of the node σ_4 are added, is shown in Figure 3, along with the facts of its canonical source instance. $\triangleleft$

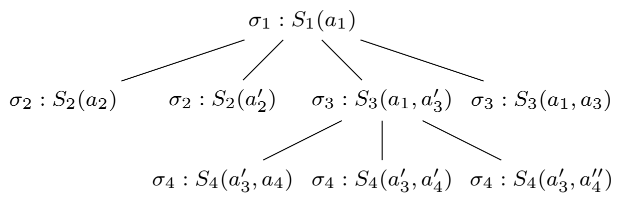

$\sigma_1 : S_1(a_1)$
$\sigma_2 : S_2(a_2)$
$\sigma_2 : S_2(a_2')$
$\sigma_3 : S_3(a_1, a_3')$
$\sigma_3 : S_3(a_1, a_3)$
$\sigma_4 : S_4(a_3', a_4)$
$\sigma_4 : S_4(a_3', a_4')$
$\sigma_4 : S_4(a_3', a_4'')$

Figure 3: A 3-pattern and the facts constituting its canonical source instance.

Finally, we have the stage set to put the procedure Implies into action.

Example 3.10 Consider the nested tgd τ and the s-t tgds τ' and τ'':

$\tau : \forall x_1\, (S_1(x_1) \rightarrow \exists y(\forall x_2 S_2(x_2) \rightarrow R(x_2, y)))$

$\tau' : \forall x_2\, (S_2(x_2) \rightarrow \exists z R(x_2, z))$

$\tau'' : \forall x_1 \forall x_2\, (S_1(x_1) \wedge S_2(x_2) \rightarrow R(x_2, x_1))$

It is easy to see that $\tau' \not\models \tau$ and $\tau'' \models \tau$. We now verify that procedure Implies yields the same answers. The Skolemization of the nested tgd τ has the form:

$$\begin{array}{ll} \tau_1 : & \forall x_1\big(S_1(x_1) \rightarrow \\ \tau_2 : & \quad \forall x_2(S_2(x_2) \rightarrow R_2(x_2, f(x_1)))\big) \end{array}$$

According to line 4 of the procedure, the bound k on the number of clones should be 2 for testing $\tau' \models \tau$ and 3 for testing $\tau'' \models \tau$, since we have $v_\sigma = 1$, $w_{\{\tau'\}} = 1$, and $w_{\{\tau''\}} = 2$. The set $\mathcal{P}_3(\tau)$ has two 1-patterns $\{p', p''\}$, of which only p'' has a non-empty canonical target instance. Based on p'', the 2-pattern p_2'' and the 3-pattern p_3'' can be obtained. One can then check that the set $\{p', p'', p_2'', p_3''\}$ is actually the complete set of 3-patterns of τ.

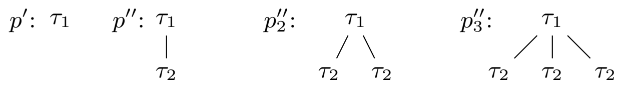


Figure 4: Patterns used to test $\tau' \models \tau$ and $\tau'' \models \tau$.

Let I_p and J_p denote the canonical source resp. canonical target instance of a pattern p of τ with $p \in \{p', p'', p_2'', p_3''\}$. Let Σ be one of $\{\tau'\}$ or $\{\tau''\}$. To test if $\Sigma \models \tau$, the procedure Implies checks the existence of a homomorphism $J_p \rightarrow chase(I_p, \Sigma)$ for the patterns p', p'', p_2'' and p_3'' (for the case $\Sigma = \{\tau''\}$). For p', this check is trivial since p' has an empty canonical target instance. We illustrate the check for the pattern p_2''. The canonical source and target instances for this pattern are as follows:

$I_{p_2''} = \{S_1(a_1), S_2(a_2), S_2(a_2')\}$

$J_{p_2''} = \{R_2(a_2, f(a_1)), R_2(a_2', f(a_1))\}$

Let us check $\tau' \not\models \tau$. The Skolemization of τ' is

$\tau' : \forall x_2\, (S_2(x_2) \rightarrow R(x_2, g(x_2)))$

The chase of $I_{p_2''}$ with τ' results in $J_{\tau'} = \{R(a_2, g(a_2)), R(a_2', g(a_2'))\}$. Due to the absence of a homomorphism from $J_{p_2''}$ to $J_{\tau'}$, Implies($\{\tau'\}$,τ) outputs *false*, as it should.

We now check that $\tau'' \models \tau$ holds. The chase of $I_{p_2''}$ with τ'' results in $J_{\tau''} = \{R(a_2, a_1), R(a_2', a_1)\}$. The mapping $[f(a_1) \mapsto a_1]$ is a homomorphism $J_{p_2''} \rightarrow J_{\tau''}$, and thus the check in the Implies procedure for the pattern p_2'' passes successfully. One can verify that so does the check for the patterns p'' and p_3''. Therefore, Implies($\{\tau''\}$,τ) outputs *true*, as it should. $\triangleleft$

Proof of Theorem 3.1 (Idea). Two ideas underlie the correctness of the decision procedure Implies, and thus of Theorem 3.1. The first is a well-known property of schema mappings which are closed under target homomorphisms and for which a chase procedure producing universal solutions exists. Namely, $\Sigma \models \sigma$ if and only if for every source instance I, we have that $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds, which is the case when every f-block of $chase(I, \sigma)$ can be homomorphically embedded in $chase(I, \Sigma)$ [7]. The second idea is specific to nested tgds. It uses the fact that $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds for arbitrary I if for every k-pattern $p_k \in \mathcal{P}_k(\sigma)$, the homomorphism $J_{p_k} \rightarrow chase(I_{p_k}, \Sigma)$ exists, where k is a constant depending on σ and Σ, as defined at line 4 of the decision procedure Implies, and I_{p_k} and J_{p_k} are the canonical source instance and, respectively, the canonical target instance of p_k.

The rationale for choosing k is based on the following claim. Let t_1 and t_2 be two triggerings in a chase tree of σ such that neither triggering is an ancestor of the other one. Then, the facts generated by t_1 and t_2 can only share nulls that instantiate Skolem terms based on variables bound in the common ancestor triggerings of t_1 and t_2. Based on that, one can show that any large f-block B generated by a nested tgd via chase must be "stitched together" from small fragments having a small number of common nulls. Namely, there are at most v_σ such nulls, where v_σ is the number of distinct Skolem terms in σ. A homomorphism $h : B \rightarrow chase(I, \Sigma)$ maps every such null either to a constant or to a null created by Σ. In the latter case, this null corresponds to some Skolem term in Σ. A Skolem term can be based on at most w_Σ variables, where w_Σ is the maximum number of variables in any tgd in Σ. Now, let B contain $k = v_\sigma \cdot w_\Sigma + 1$ fragments corresponding to clones of some pattern subtree. Using the pigeonhole principle one can show that if facts corresponding to yet further clones of the same subtree are added to B, the homomorphism h can be extended to such an increased f-block. Thus, the property $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ for arbitrary I can be ensured by inspecting f-blocks of canonical instances corresponding to chase trees with at most k clones of any subtree. Such chase tree patterns are among the k-patterns of σ. $\square$

We conclude this section by discussing an immediate consequence of Theorem 3.1.

Corollary 3.11 *The logical equivalence problem for nested tgds is decidable.*

In contrast, it is known that the logical equivalence problem for SO tgds is undecidable, according to Theorem 1 in [9], which builds on [3]. As a matter of fact, an examination of the proof of Theorem 1 in [9] reveals that the following problem is undecidable: given an SO tgd σ and a finite set Σ' of s-t tgds, is $\sigma \equiv \Sigma'$? Hence, the following problem is undecidable as well: given an SO tgd σ and a finite set Σ' of nested tgds, is $\sigma \equiv \Sigma'$? Therefore, Corollary 3.11 contributes significantly to the delineation of the boundary between decidability and undecidability for the logical equivalence problem.

4. The Structure of the Core and Applications

As mentioned in Section 2, the class of nested tgds contains the class of s-t tgds and is, in turn, contained in the class of plain SO tgds. Moreover, it is known that both containments are proper. In this section, we produce powerful tools that allow us to tell apart nested tgds from s-t tgds, and also plain SO tgds from nested tgds. The main result of this section is an algorithm for telling whether or not a given finite set of nested tgds is logically equivalent to a finite set of s-t tgds. In addition, we give useful and easy-to-apply sufficient conditions for a plain SO tgd to be not logically equivalent to any finite set of nested tgds.

The results in this section are obtained by analyzing the structure of the core of the universal solutions of nested GLAV mappings. We embark on this analysis next, which we believe is of interest in its own right.

4.1 Nested GLAV Mappings vs. GLAV Mappings

Recall that every schema mapping $\mathcal{M}$ specified by an SO-tgd admits universal solutions. Moreover, for every source instance I, a canonical universal solution $\mathit{chase}(I, \mathcal{M})$ for I w.r.t. $\mathcal{M}$ can be obtained via the chase procedure. Since all universal solutions for a given source instance I are homomorphically equivalent, it follows that their cores are unique up to isomorphism, hence we can take $\mathit{core}(\mathit{chase}(I, \mathcal{M}))$ as *the* core of the universal solutions for I w.r.t. $\mathcal{M}$ [7]. Note that, in general, $\mathit{core}(\mathit{chase}(I, \mathcal{M}))$ need not be a universal solution for I w.r.t. $\mathcal{M}$ [6]. However, if $\mathcal{M}$ is specified by a plain SO tgd, then $\mathit{core}(\mathit{chase}(I, \mathcal{M}))$ is a universal solution for I w.r.t. $\mathcal{M}$. The reason for this is that, as shown in [2], every schema mapping $\mathcal{M}$ specified by a plain SO tgd is *closed under target homomorphisms*, which means that if J is a solution for I w.r.t. $\mathcal{M}$ and if there is a homomorphism from J to J' that is the identity on constants, then J' is also a solution for I w.r.t. $\mathcal{M}$. Moreover, $\mathit{core}(\mathit{chase}(I, \mathcal{M}))$ is the smallest universal solution for I w.r.t. $\mathcal{M}$. In particular, the above facts hold true for nested GLAV mappings (hence also for GLAV mappings).

We will make extensive use of the following notion, which was introduced in [6]. A schema mapping $\mathcal{M}$ specified by an SO tgd has *bounded f-block size* if there is a positive integer b such that for every source instance I, the f-block size of $\mathit{core}(\mathit{chase}(I, \mathcal{M}))$ is at most b; otherwise, we say that $\mathcal{M}$ has *unbounded f-block size*. The following result follows immediately from Proposition 3.14 and Theorem 4.10 in [6].

Theorem 4.1 ([6]) *A schema mapping $\mathcal{M}$ specified by a plain SO tgd is logically equivalent to a GLAV schema mapping if and only if $\mathcal{M}$ has bounded f-block size. In particular, this holds true for every nested GLAV schema mapping.*

The preceding Theorem 4.1 will be used to prove the main result in this section, which we now state formally.

Theorem 4.2 *The following problem is decidable: given a nested GLAV mapping $\mathcal{M}$, is there a GLAV mapping $\mathcal{M}'$ such that $\mathcal{M}$ is logically equivalent to $\mathcal{M}'$?*

In view of Theorem 4.1, it suffices to give an algorithm that, given a nested GLAV schema mapping $\mathcal{M}$, determines whether or not $\mathcal{M}$ has bounded f-block size. To this end, we introduce a crucial property of mappings and show that nested GLAV mappings have this property.

Definition 4.3 Let $\mathcal{C}$ be a class of schema mappings. We say that $\mathcal{C}$ has *effective threshold for f-block size* if there exists a recursive function $f : \mathcal{C} \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, s.t. every mapping $\mathcal{M} \in \mathcal{C}$ either has f-block size at most $f(\mathcal{M})$ or has unbounded f-block size. $\triangleleft$

Theorem 4.4 *The class of nested GLAV mappings has effective threshold for f-block size.*

Proof (Idea). We show that if the size of an f-block in the core is above a certain threshold that depends on the maximum size of a 1-pattern, then two siblings in the chase tree of that f-block have isomorphic subtrees (up to variable bindings). After that, we "clone" a third subtree, and show that this strictly increases the f-block size. Finally, we show that the claimed f-block with increased size indeed persists in the core of an extended source instance. It is then clear that, by successively increasing the size of this f-block, the size can increase beyond any bound. □

The following statement was claimed in [4].

Claim 4.5 *There is an algorithm for the following problem: Given an SO tgd σ and a positive integer b, is the f-block size of σ bounded by b?*

Our desired Theorem 4.2 would follow immediately by combining Theorem 4.1 and Theorem 4.4 with Claim 4.5. Alas, while the algorithm for Claim 4.5 presented in [4, Theorem 5.2] appears to be correct, the proof of correctness of the algorithm given there has a flaw, which will be pointed out in the sequel. It should be noted that the above claim would also follow from Theorem 3 in [16] together with Theorem 4.10 in [6]; however, Theorem 3 in [16] is stated without proof. In view of this state of affairs, we prove that Claim 4.5 indeed holds for nested GLAV mappings. For this purpose, we introduce the following concept.

Definition 4.6 A schema mapping $\mathcal{M}$ is said to have a *bounded anchor* if there exists an integer a such that for every source instance I and for every connected target instance J with $J \subseteq \mathit{core}(\mathit{chase}(I, \mathcal{M}))$, there are a source instance I' and a connected target instance J' such that

- $|I'| \leq a|J|$;
- $|J'| \geq |J|$ and $J' \subseteq \mathit{core}(\mathit{chase}(I', \mathcal{M}))$.

We say that *the bounded anchor of $\mathcal{M}$ is witnessed by a.* $\triangleleft$

We extend this notion to classes of schema mappings.

Definition 4.7 Let $\mathcal{C}$ be a class of schema mappings. We say that $\mathcal{C}$ has *effective bounded anchor*, if there exists a recursive function $a : \mathcal{C} \to \mathbb{N}$ such that every schema mapping $\mathcal{M}$ in $\mathcal{C}$ has bounded anchor witnessed by $a(\mathcal{M})$. $\triangleleft$

For understanding the intuition behind the concept of bounded anchor, let us consider an example that was brought to our attention by R. Fagin.

Example 4.8 Let σ be the following plain SO tgd:

$$\exists f\, \forall x \forall y\, (S(x,y) \to R(f(x), f(y)) \wedge R(f(y), f(x)))$$

Suppose we want to determine whether σ has a bounded anchor. Let $I_n = \{S(1,2), S(2,3), \ldots, S(n,1)\}$ be the source instance consisting of a directed cycle of length n. Then $chase(I_n, \sigma)$ is the undirected cycle of length n. Let n be an odd number. It follows that $core(chase(I_n, \sigma))$ is also the undirected cycle of length n, which we depict on the left side of Figure 5 for $n > 5$ (each arc, whether it is solid or dashed, denotes an R-atom).

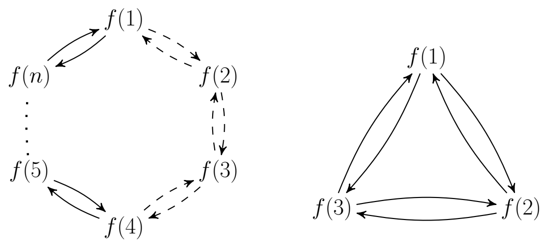


Figure 5: The undirected cycle of length n on the left side, and of length 3 on the right side.

We now take J to be the subinstance of $core(chase(I_n, \sigma))$ consisting of the dashed edges on the left side of Figure 5 (i.e., 6 R-atoms denoted by the 6 dashed arcs). Intuitively, the definition of bounded anchor requires us to find a "small" source instance I' that gives rise to a connected J' of size at least $|J| = 6$ such that J' is contained in $core(chase(I', \sigma))$. Here, "small" means that the size of I' may depend on the size of J but not on n. Now observe that no such small source instance can be constructed using the atoms of I_n: if I' is *any* proper subinstance of I_n, then $core(chase(I', \sigma))$ is just an undirected R-edge. However, we can meet the requirements in the definition of bounded anchor by taking $I' = I_3$ (note that for each $n > 3$, $I_3 \not\subseteq I_n$ holds). Indeed, $core(chase(I_3, \sigma))$ is the undirected cycle of size 3, as depicted on the right side of Figure 5. ◁

Note that Example 4.8 yields a counter-example to a step in the proof of correctness of the algorithm in Theorem 5.2 in [4], where the search for I' and J' was confined to subsets of the given instances I and J.

Example 4.8 also illustrates that it is not always easy to find a bounded anchor. However, we show next that the class of nested tgds indeed has effective bounded anchor.

Theorem 4.9 *The class of nested GLAV mappings has effective bounded anchor.*

Proof (Idea). Given I and J, we first construct an overestimation I_b and J_b, where J_b fulfills the lower bound of the bounded anchor definition, but I_b is too large. Intuitively, this J_b is of the size of the f-block B in which the connected subinstance J is contained. From this overestimation, we compute an underestimation I_0 and J_0, where I_0 fulfills the upper bound of the bounded anchor definition, but J_0 is too small. I_0 and J_0 are, respectively, the canonical source and target instances of a k-pattern from which the pattern of J_b can be obtained by cloning of subtrees. Here, k is defined as in the procedure IMPLIES in Section 3. From this underestimation, we compute our final I' and J' according to the definition of bounded anchor by a suitable cloning of subtrees. Based on the ideas underpinning the proof of Theorem 3.1, J' can be shown to satisfy the required properties of an anchor. □

The computation of the function for the effective bounded anchor in the preceding Theorem 4.9, as well as the function witnessing effective threshold in Theorem 4.4, utilize the concept of a k-pattern introduced in Section 3. Hence, by the considerations in Section 3, we have that both are non-elementary in the depth of the nested tgd.

Having shown that the class of nested GLAV mappings has effective bounded anchor, we can now prove that Claim 4.5 indeed holds for the class of nested GLAV mappings.

Theorem 4.10 *Let $\mathcal{C}$ be a class of schema mappings that has effective bounded anchor. Then the following problem is decidable: given a schema mapping $\mathcal{M}$ in $\mathcal{C}$ and a positive integer b, is the f-block size of $\mathcal{M}$ at most b?*

Proof (Sketch). Let a be a witness of the bounded anchor of $\mathcal{M}$. We test for all source instances I with $|I| \leq a(b+1)$ whether the f-block size of $core(chase(I, \Sigma))$ is at most b. There are finitely many such instances (up to isomorphism) and each test itself is decidable by computing and inspecting the core. If at least one of these tests returns an f-block size greater than b, we return that the f-block size is greater than b. Otherwise, we return that the f-block size is at most b. □

Now, by exploiting the fact that nested GLAV mappings have both effective threshold and effective bounded anchor, there is an algorithm for deciding whether the f-block size of a nested GLAV mapping is bounded.

Theorem 4.11 *Let $\mathcal{C}$ be a class of schema mappings having both effective threshold for f-block size and effective bounded anchor. Then the following problem is decidable: given a schema mapping $\mathcal{M}$ in $\mathcal{C}$, does $\mathcal{M}$ have bounded f-block size?*

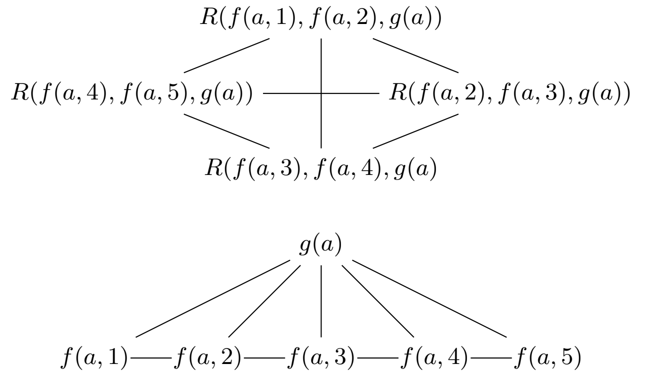
Proof Let f be the recursive function providing the effective threshold for f-block size for schema mappings in $\mathcal{C}$. Consider the following algorithm: given a mapping $\mathcal{M}$ in $\mathcal{C}$, compute the bound $b = f(\mathcal{M})$ for the effective threshold for f-block size. Since $\mathcal{C}$ has effective bounded anchor, we can use the algorithm in Theorem 4.10 to test whether $\mathcal{M}$ has f-block size bounded by b. If it does, we return that $\mathcal{M}$ has bounded f-block size; otherwise, we return that $\mathcal{M}$ has unbounded f-block size. □

By assembling all the preceding machinery, we can now prove the main result of this section.

Proof of Theorem 4.2 By Theorem 4.4 and Theorem 4.9, the class of nested GLAV mappings has both effective threshold for f-block size and effective bounded anchor. Therefore, by Theorem 4.11, the following problem is decidable: given a nested GLAV mapping, does it have bounded f-block size? Thus, together with Theorem 4.1, we get the decidability result stated in Theorem 4.2. □

4.2 Plain SO tgds vs. Nested GLAV Mappings

We have just seen that there is an algorithm to differentiate between nested GLAV mappings and GLAV mappings. It is not known, however, whether there is an algorithm to differentiate between plain SO tgds and nested GLAV mappings. In other words, it is not known whether or not the



$R(f(a,1), f(a,2), g(a))$
$R(f(a,4), f(a,5), g(a))$
$R(f(a,2), f(a,3), g(a))$
$R(f(a,3), f(a,4), g(a)$
$g(a)$
$f(a,1)$
$f(a,2)$
$f(a,3)$
$f(a,4)$
$f(a,5)$

Figure 6: Gaifman graph of facts (top) and Gaifman graph of nulls (bottom) of Example 4.14 for a successor relation of length 5.

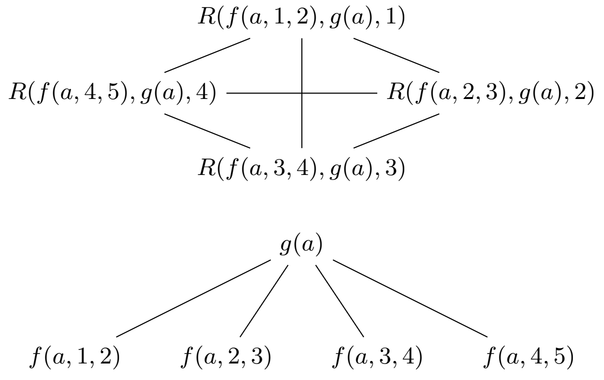


Figure 7: Gaifman graph of facts (top) and Gaifman graph of nulls (bottom) of Example 4.15 for a successor relation of length 5.

following problem is decidable: given a plain SO tgd σ, is there a nested GLAV mapping $\mathcal{M}$ such that σ is logically equivalent to $\mathcal{M}$?

What tools are there for showing that a particular plain SO tgd σ is not logically equivalent to any nested GLAV mapping? Since plain SO tgds are expressible in second-order logic while nested GLAV mappings are expressible in first-order logic, it suffices to show that σ is not first-order expressible. The standard method for doing this are Ehrenfeucht-Fraïssé games or locality methods (see [15]). In fact, essentially this method is behind the proof in [2] that the plain SO tgd

$$\exists f\, \forall x \forall y\, (S(x,y) \rightarrow R(f(x), f(y)))$$

is not logically equivalent to any nested GLAV mapping. In what follows, we take a totally different approach and give two different sufficient conditions for showing that a given SO tgd is not logically equivalent to any nested GLAV mapping. The idea behind these conditions is as follows. Suppose we suspect that a given plain SO tgd σ is not logically equivalent to any nested GLAV mapping. In this case, σ is not equivalent to any GLAV mapping either and, hence, σ has unbounded f-block size by Theorem 4.1. Now, a schema mapping may have unbounded f-block size for a number of different reasons. However, we will show that a nested GLAV mapping can have unbounded f-block size only for certain specific reasons that are not shared by all plain SO tgds. Therefore, if the given SO tgd σ falls in one of these categories, then we can conclude that indeed σ is not logically equivalent to any nested GLAV mapping.

Before stating the first result of this section, we need to relativize the notion of bounded f-block size to a class of source instances.

Assume that $\mathcal{M}$ is a schema mapping specified by an SO tgd and $\mathcal{C}$ is a class of source instances. We say that $\mathcal{M}$ has *bounded f-block size on* $\mathcal{C}$ if there is a positive integer b such that for every source instance I in $\mathcal{C}$, the f-block size of $core(chase(I, \mathcal{M}))$ is at most b; otherwise, we say that $\mathcal{M}$ has *unbounded f-block size on* $\mathcal{C}$. Clearly, $\mathcal{M}$ has bounded f-block size if it has bounded f-block size on the class of all source instances.

If G is an undirected graph and v is a node of G, then the *degree of* v is the number of edges incident to v. The *degree of* G is the maximum degree of its nodes. We say that a schema mapping has *bounded f-degree on* $\mathcal{C}$ if there is a positive integer d such that for every source instance I in $\mathcal{C}$, the degree of every f-block of $core(chase(I, \mathcal{M}))$ is at most d; otherwise, we say that $\mathcal{M}$ has *unbounded f-degree on* $\mathcal{C}$.

Theorem 4.12 *Let $\mathcal{M}$ be a nested GLAV mapping and $\mathcal{C}$ a class of source instances. Then $\mathcal{M}$ has bounded f-block size on $\mathcal{C}$ if and only if $\mathcal{M}$ has bounded f-degree on $\mathcal{C}$.*

Informally, the preceding theorem asserts that nested tgds can achieve unbounded f-block size on a class of source instances only because some null value appears unboundedly often in the core of the universal solutions of such instances. In contrast, plain SO tgds can achieve unbounded f-block size in more complex ways, as evidenced by the next result.

Proposition 4.13 *There is a plain SO tgd τ and a class $\mathcal{C}$ of source instances such that τ has unbounded f-block size on $\mathcal{C}$, but bounded f-degree on $\mathcal{C}$.*

Proof Let τ be the plain SO tgd

$$\exists f\, \forall x \forall y\, (S(x,y) \rightarrow R(f(x), f(y)))$$

and let $\mathcal{C}$ be the class of all source instances I such that S is a successor relation. If $I \in \mathcal{C}$, then $core(chase(I, \tau))$ consists of a single f-block of the same size as S in which no null occurs more than twice. Thus, the f-block size of τ on $\mathcal{C}$ is unbounded, but the f-degree of τ on $\mathcal{C}$ is 2. $\square$

It follows that the plain SO tgd τ in Proposition 4.13 is not logically equivalent to any nested GLAV mapping. Altogether, f-degree is an easy-to-use tool for showing that a schema mapping is not logically equivalent to a nested GLAV mapping. However, it is not always sufficient for dealing with for more complex schema mappings, as the next example shows.

Example 4.14 Consider the following plain SO tgd σ:

$$\exists f\, \forall x \forall y \forall z\, (S(x,y) \wedge Q(z) \rightarrow R(f(z,x), f(z,y), g(z)))$$

It will turn out that σ is not logically equivalent to any nested GLAV mapping. However, it is easy to see that each f-block is a clique, which implies that for every class $\mathcal{C}$ of source instances, σ has unbounded f-block size on $\mathcal{C}$ if and only if it has unbounded f-degree on $\mathcal{C}$. For example, if $\mathcal{C}$

is the class of source instances in which S is a successor relation and Q is a singleton, then each f-block is a clique of the form depicted in the upper part of of Figure 6. Thus, Theorem 4.12 cannot be used to show that σ is not logically equivalent to any finite set of nested tgds. $\triangleleft$

The preceding example shows that, in addition to Theorem 4.12, a different structural tool is needed to differentiate between plain SO tgds and nested GLAV mappings. To appreciate how delicate this differentiation can be, we note that a plain SO tgd and a nested tgd may have the same f-blocks on some classes of instances, yet the plain SO tgd is not logically equivalent to any finite set of nested tgds.

Example 4.15 Consider the following plain SO tgd σ':

$$\exists f \exists g\, \forall x \forall y \forall z\, (S(x,y) \wedge Q(z) \rightarrow R(f(z,x,y), g(z), x))$$

This dependency is logically equivalent to the following nested tgd:

$$\forall z\, (Q(z) \rightarrow \exists u\, (\forall x \forall y\, (S(x,y) \rightarrow \exists v\, R(v,u,x))))$$

For the source instances in which S is a successor relation, the f-blocks are the same as those for the plain SO tgd σ in Example 4.14, i.e., they are complete graphs (see Figure 6 and 7). Yet, as we are about to discover, σ is not logically equivalent to any finite set of nested tgds. $\triangleleft$

To cope with this situation, we need to look beyond the Gaifman graph of facts. Recall that the Gaifman graph of facts (in short, fact graph) is the graph whose nodes are the facts and there is an edge between two facts if they share a null. Let J be a target instance. The Gaifman graph of nulls of J (in short, null graph), is the graph whose nodes are the nulls of J, and there is an edge between two nulls if they occur in the same fact in J.

It turns out that properties of the null graph can be used to show inexpressibility in situations where the structure of the fact graph is of no help. More formally, the *path length* of an undirected graph G is the length of the longest simple path in G, where a simple path is a path that visits each node in G at most once. We say that a schema mapping $\mathcal{M}$ specified by an SO tgd has *bounded path length* if there is a positive integer l such that for every source instance I, the path length of the null graph of $core(chase(I, \mathcal{M}))$ is at most l; otherwise, we say that $\mathcal{M}$ has *unbounded path length.*

Theorem 4.16 *Every nested GLAV mapping has bounded path length.*

Equipped with Theorem 4.16, we now have a tool to show that the plain SO tgd σ of Example 4.14 is not logically equivalent to any nested GLAV mapping. This is so because σ has unbounded path length, which can be checked using successor relations in S as source instances (see the bottom part of Figure 6, where the null graph contains a simple path of length 4).

5. Adding Source Constraints

In the previous section, we showed that it is decidable whether a schema mapping based on nested tgds is equivalent to a GLAV mapping. The decidability of whether an SO tgd is equivalent to a GLAV mapping is still open. In this section, we give evidence that the problem may indeed be harder for SO tgds: It is undecidable whether a plain SO tgd is equivalent to a GLAV mapping in the presence of a single source key dependency. In contrast, for nested tgds and in the presence of arbitrary source egds, equivalence to GLAV is still decidable. Completing the picture, we also show that the implication problem of nested tgds discussed in Section 3 remains decidable in the presence of source egds.

Recall that Theorem 4.1 reduces the problem of whether a plain SO tgd is equivalent to a GLAV mapping to the problem of deciding whether it has bounded f-block size. This theorem, which is derived from Proposition 3.14 and Theorem 4.10 in [6], thus played an important role in Section 4. A close inspection of the proofs of Proposition 3.14 and Theorem 4.10 in [6] shows that these results (and therefore Theorem 4.1) still hold in the presence of source egds. We make use of this fact below.

Theorem 5.1 *It is undecidable whether a given plain SO tgd is equivalent to a GLAV mapping in the presence of a single source key dependency.*

Proof (Idea). By the above comments it suffices to show the undecidability of the problem if, given a plain SO tgd and a source key dependency, the mapping has bounded f-block size. Our proof is by reduction from the halting problem.

Thus for a given Turing machine, we construct an SO tgd that "simulates" the computation of the Turing machine. The basic structure for our construction is to represent a run of a Turing machine (state and tape configurations) together with a successor relation in the source instance. We construct a key dependency to ensure that in the supposed successor relation, each element has a unique predecessor. The SO tgd then guarantees that the f-block size is bounded if and only if the Turing machine halts.

The particular challenge of this reduction is how to handle incorrect and missing information in the source instance. For incorrect information, we define "guards" that lead to a collapse of f-block size. The more problematic part is missing information, for which we define a specific one-dimensional enumeration of the two-dimensional (time and tape space) structure of the Turing machine's run in the target. When we reach a certain point of this enumeration, we know that no essential information is missing up to that point.

The final challenge is how to handle the effects of unintended structure of the successor relation given in the source instance. While the single key dependency gives us some control over the structure, we define "traps" that address the effects of deviating from the successor relation in the target in ways not handled by that single key dependency. $\square$

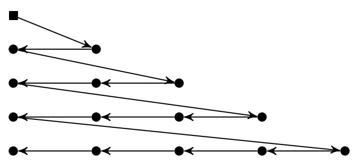


Figure 8: Intended enumeration represented in the target instance. Arrows denote N atoms.

Missing Information. To give a bit of the flavor of the construction in the proof of Theorem 5.1, we highlight one of the main challenges, namely how to handle missing information. The enumeration of the configurations we materialize in the target instance is illustrated in Figure 8. The vertical axis represents time and the horizontal axis the tape. Note that it is only necessary to represent this triangular part of the configuration matrix, as a Turing machine can in, e.g., 4 steps in time at most reach the 4^{th} tape cell.

The first key fact about this enumeration is that it uses the successor relation, both in space and in time, only in one direction (the "backwards" direction). This is necessary because with a single key dependency we can only guarantee that one direction allows correct navigation (in our case, we guarantee unique predecessors). The only other navigation step we can ensure to be correct is "jumping to the diagonal" (the time and space index coincide) as illustrated by the diagonal arrows in Figure 8.

What we have to show is that we can indeed, using the successor relation only in correct ways, generate this enumeration using SO tgds. Assume that S represents a successor relation and Z represents the initial element ("'zero") of that successor relation. During the construction, we define an abbreviation $\mathit{check}_{\pi_{good}}[x, y]$ that checks whether the Turing machine at time instant x and at tape position y represents a certain locally correct (π_{good}) configuration. It is a complex definition that does not give major insights, so we do not give it here. The crucial parts are the following two plain SO tgds:

$$\mathit{check}_{\pi_{good}}[x, y] \wedge S(y, y') \rightarrow N(f(x, y'), f(x, y))$$
$$\mathit{check}_{\pi_{good}}[x', x'] \wedge S(x, x') \wedge Z(y) \rightarrow N(f(x, y), f(x', x'))$$

The first SO tgd realizes the $\leftarrow$ step in Figure 8, while the second SO tgd realizes the $\searrow$ step (which can be seen by the term $f(x', x')$, the "diagonal"). Note that they are mutually exclusive as one checks for a predecessor via S, and the other one for the initial element using Z.

To sum up, while each step in the enumeration guarantees that locally the configuration is correct, the full unbroken enumeration thus guarantees that globally, the computation of the Turing machine is correct. In particular, if we have missing information, the enumeration will break. Finally, recall that we are interested in the f-block size. What our construction ensures through appropriate graph gadgets is that a part of the enumeration that is not connected to the origin (the square node in Figure 8) will collapse in the core, thus not contribute to the f-block size.
The Turing machine construction of Theorem 5.1 can be used to give an alternative proof of the undecidability of the equivalence of plain SO-tgds in the presence of source key dependencies, which was originally shown in [9] by a reduction from the domino problem.

Also, we note that the SO tgd simulating a Turing machine computation can produce a core with f-blocks of arbitrary size but with bounded f-degree if the Turing machine does not halt. In this case, by Theorem 4.12, the SO tgd cannot be equivalent to a nested GLAV mapping. This immediately gives us the following undecidability result:

Theorem 5.2 *It is undecidable whether a given plain SO tgd is equivalent to a nested GLAV mapping in the presence of a single source key dependency.*

Nested tgds. We now show that in contrast, the problem of deciding whether a set of nested tgds is equivalent to a GLAV mapping is decidable even in the presence of source egds.

The proof strategy of Theorem 4.2 is still valid, namely showing that the class of schema mappings has (1) effective threshold, (2) effective bounded anchor as well as that (3) a mapping is logically equivalent to a GLAV mapping iff f-block size is bounded. We already mentioned that (3) still holds in the presence of source egds. It thus remains to show that also (1) and (2) hold if source egds are allowed.

We now show that nested tgds have effective threshold for f-block size also in the presence of source egds. However, a straightforward extension of Theorem 4.4 is not possible, as the following example illustrates.

Example 5.3 Consider the following nested tgd σ:

$$\forall z\, (Q(z) \rightarrow \exists y\, \forall x_1 \forall x_2 \\ (P_1(z, x_1) \wedge P_2(z, x_2) \rightarrow R(y, x_1, x_2)))$$

and the set Σ_s of source dependencies given by

$$P_1(z, x_1) \wedge P_1(z, x_1') \rightarrow x_1 = x_1'$$

Now consider the source instance I given as

$$\{Q(a), P_1(a, b), P_2(a, b), P_2(a, c)\}$$

The proof of Theorem 4.4 depends on "cloning" parts of the source instance. Intuitively, in our example this means constructing a source instance $I' = I \cup I[b \mapsto d]$, where $[b \mapsto d]$ denotes replacing all occurrences of b by d. That is, we have $I' = I \cup \{Q(a), P_1(a, d), P_2(a, d), P_2(a, c)\}$. But while both I and $I[b \mapsto d]$ satisfy Σ_s, the combined instance I' does not. Indeed, $\{P_1(a, b), P_1(a, d)\}$ violates Σ_s. $\triangleleft$

Still, it is possible to show effective threshold also in this case. The key tool for this result and further results in this section is an adapted notion of canonical instances that takes source dependencies into account.

Definition 5.4 Consider a nested tgd σ defined for the source schema $\mathbf{S}$ and a target schema $\mathbf{T}$, such that a set of integrity constraints Σ_s consisting of egds is associated with $\mathbf{S}$. Given a pattern p of a chase tree with σ, we define the *legal canonical source resp. target instances* I_p^s, J_p^s as the instances obtained from the canonical source resp. target instances I_p and J_p of σ and p as follows: I_p^s results from chasing I_p with Σ_s, and J_p^s results from J_p by enforcing all equalities between constants of I_p implemented in I_p^s. $\triangleleft$

Similarly to the case without source dependencies, we speak of *the* legal canonical source resp. target instances, since irrespectively of the choice of constants used to produce such instances, they are unique up to constant renaming. We are now able to show our desired result.

Theorem 5.5 *The class of nested tgds with source egds has effective threshold for f-block size.*

Proof (Idea). This is shown by adapting the proof of Theorem 4.4 through using legal canonical instances. Additional claims are needed to show that the equalities present in a legal canonical target instance do not interfere with the "cloning" process needed to increase the f-block size. ☐

Furthermore, the argumentation in the proof of Theorem 4.9 that nested tgds have effective bounded anchor still holds in the presence of source egds, by using legal canonical instances. We can thus extend Theorem 4.2:

Theorem 5.6 *It is decidable whether a given nested GLAV mapping is equivalent to a GLAV mapping in the presence of source egds.*

Finally, we show that also the implication problem remains decidable in the presence of source egds.

Theorem 5.7 *The implication problem for nested tgds is decidable in the presence of source egds.*

Proof (Idea). It turns out that in the presence of source egds, all results of Section 3 required to show the correctness of the procedure Implies remain valid, with the provision that legal canonical instances are used instead of canonical instances. The adaptations necessary to ensure correctness complicate the proof considerably, but the intuition behind the machinery introduced in Section 3 persists. □

This result clearly separates the complexity of reasoning tasks for nested and plain SO tgds, since as shown in [9], equivalence — and thus also implication — of plain SO tgds is undecidable even in the presence of a single source key dependency.

6. Concluding Remarks

In this paper, we initiated the study of fundamental reasoning tasks and structural properties of nested tgds. On the positive side, we showed that the following problems are decidable: the implication problem (and hence the equivalence problem) of nested tgds, and the problem of deciding whether a given nested GLAV mapping is equivalent to some GLAV mapping. We also showed that these problems remain decidable even if source egds are allowed. In contrast, we established that the problem whether a given plain SO tgd is equivalent to some GLAV mapping becomes undecidable as soon as a single key dependency is allowed in the source schema.

For future work, the decidability of the equivalence problem for plain SO tgds and of the problem of determining whether a given plain SO tgd is equivalent to some GLAV mapping (resp. to some nested GLAV mapping) remains open. Moreover, the aforementioned decidability results for nested tgds call for further study: all of our decidability results depend on the notion of k-patterns introduced in Section 3. As pointed out in that section, the number of k-patterns and the maximum size of k-patterns for a given nested tgd are non-elementary in the depth of the nested tgd, and so are all algorithms utilizing patterns. It is worth investigating whether this high complexity is inherent in these problems or more efficient algorithms can be designed.

Another important direction for future research is concerned with structural characterizations of schema mappings along the lines of [17]. In this paper, we discovered necessary conditions (via the notions of unbounded f-degree and bounded path length) of schema mappings that are logically equivalent to some nested GLAV mapping. These properties sometimes provide an easy argument for telling apart plain SO tgds from nested GLAV mappings. It remains open whether these properties can be extended to a sufficient condition for the expressibility by a finite set of nested tgds. For instance, are all plain SO tgds with unbounded f-degree and/or bounded path length equivalent to a nested GLAV mapping? A structural characterization of plain SO tgds (raised in [2]) also remains an interesting open problem.

Acknowledgements. Kolaitis' research on this paper was partially supported by NSF Grant IIS-1217869. The research of Pichler, Sallinger, and Savenkov was supported by the Austrian Science Fund (FWF):P25207-N23.

7. References

[1] M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange.* 2010.

[2] M. Arenas, J. Pérez, J. Reutter, and C. Riveros. The language of plain SO-tgds: Composition, inversion and structural properties. *JCSS*, 79(6):763 – 784, 2013.

[3] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: Bringing exchanged data back. *ACM TODS*, 34(4), 2009.

[4] R. Fagin and P. G. Kolaitis. Local transformations and conjunctive-query equivalence. In *PODS*, pages 179–190, 2012.

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

[6] R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a theory of schema-mapping optimization. In *PODS*, pages 33–42, 2008.

[7] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. *ACM TODS*, 30(1):174–210, 2005.

[8] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM TODS*, 30(4):994–1055, 2005.

[9] I. Feinerer, R. Pichler, E. Sallinger, and V. Savenkov. On the undecidability of the equivalence of second-order tuple generating dependencies. In *AMW*, 2011.

[10] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In *VLDB*, pages 67–78, 2006.

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

[12] M. A. Hernández, H. Ho, L. Popa, A. Fuxman, R. J. Miller, T. Fukuda, and P. Papotti. Creating nested mappings with Clio. In *ICDE*, pages 1487–1488, 2007.

[13] P. G. Kolaitis, M. Lenzerini, and N. Schweikardt, editors. *Data Exchange, Integration, and Streams*, volume 5 of *Dagstuhl Follow-Ups*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

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

[15] L. Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[16] J. Madhavan and A. Y. Halevy. Composing mappings among data sources. In *VLDB*, pages 572–583, 2003.

[17] B. ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009.