On the Language of Nested Tuple Generating Dependencies

PHOKION G. KOLAITIS, UC Santa Cruz and IBM Research - Almaden
REINHARD PICHLER, TU Wien
EMANUEL SALLINGER, TU Wien and University of Oxford
VADIM SAVENKOV, Vienna University of Economics and Business

During the past 15 years, 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 article, 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?

CCS Concepts: • **Theory of computation → Data exchange**; • **Information systems → Mediators and data integration**;

Additional Key Words and Phrases: Schema mappings, data integration, data exchange, nested dependencies, second-order dependencies

The research of Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov was supported by the Austrian Science Fund, projects (FWF):P25207-N23 and (FWF):Y698, and the Vienna Science and Technology Fund, project ICT12-015. The research of Phokion Kolaitis on this article was partially supported by NSF Grant IIS-1217869. The full version was completed while Kolaitis was visiting the Simons Institute for the Theory of Computing during the fall of 2016. The research of Emanuel Sallinger was supported by the EPSRC programme grant EP/M025268/1, the Horizon 2020 grant 809965 and the Vienna Science and Technology Fund (WWTF) WWTF grant VRG18-013.

Authors' addresses: P. G. Kolaitis, University of California, Santa Cruz, Computer Science and Engineering Department, CA 95064, USA and IBM Research - Almaden, San Jose, CA 95120, USA; email: kolaitis@ucsc.edu; R. Pichler, TU Wien, Faculty of Informatics, Institute of Logic and Computation, Database and Artificial Intelligence Group, 1040 Vienna, Austria; email: pichler@dbai.tuwien.ac.at; E. Sallinger, TU Wien, Faculty of Informatics, Institute of Logic and Computation, Database and Artificial Intelligence Group, Knowledge Graph Lab, 1040 Vienna, Austria and University of Oxford, Department of Computer Science, OX1 3QD, Oxford, United Kingdom; email: sallinger@dbai.tuwien.ac.at; V. Savenkov, Vienna University of Economics and Business, Institute for Information Business, Welthandelsplatz 1, 1020 Vienna, Austria; email: vadim.savenkov@wu.ac.at.

0362-5915/2020/07-ART8 $15.00
https://doi.org/10.1145/3369554

ACM Reference format:
Phokion G. Kolaitis, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2020. On the Language of Nested Tuple Generating Dependencies. *ACM Trans. Database Syst.* 45, 2, Article 8 (July 2020), 59 pages.
https://doi.org/10.1145/3369554

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 15 years, 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 Reference [11], SO tgds are the *right* language for expressing compositions 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 comprehensive overviews of the literature, see References [1, 19, 21, 25, 27].

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

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

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 Reference [13] and demonstrated in Reference [18] 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 later became part of IBM's InfoSphere BigInsights suite. Richer nested mappings, including nested XML mappings and nested second-order tgds, were subsequently used in the Clip system [26]. In the peer data management context, in which data are exchanged between several different peer nodes (instead of between a single source schema and a single target schema), the Piazza system [16] was capable of performing query reformulation over XML data sources. As pointed out in Reference [6], the underlying mapping language Piazza XML (syntactically a fragment of XQuery) is essentially the language of nested tgds. Nested schema transformations in the peer data integration context were also used in the HePToX system [5].

The main argument in favor of nested GLAV mappings over GLAV mappings given in References [13, 18] 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. Compared to SO tgds, nested GLAV mappings "offer a more natural programming paradigm for mapping tasks" [13]. This state of affairs suggests that there are situations in which it is important to determine if an SO tgd can be replaced by an equivalent nested GLAV mapping. Sufficient conditions under which such a rewriting is possible have been identified in Reference [4] and, very recently, in Reference [15].

Plain SO tgds were introduced and studied in depth in Reference [2] as a language particularly well suited for handling both composition and inversion of GLAV mappings. The results in Reference [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 Reference [24]).

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 nested GLAV mapping. Nested GLAV mappings and plain SO tgds share several desirable structural properties, such as admitting universal solutions and being closed under target homomorphisms [2, 28]. 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. As regards reasoning tasks, the decidability of the implication problem for plain SO tgds remains open, whereas the logical equivalence problem for SO tgds (hence, also the implication problem for SO tgds) 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 References [3, 12]). Coping with this complexity motivates the research on simplifying SO tgds and, in particular, plain SO tgds with first-order nested tgds [4, 15].

In spite of the uses and applications of nested tgds, no systematic investigation of nested tgds in their own right has been carried out to date. Our goal in this article is to embark on such an investigation by focusing on the basic reasoning tasks, algorithmic problems, and structural properties of nested GLAV mappings.

The implication problem is a fundamental problem in mathematical logic and also one of the central problems in database dependency theory. Our first main result is that the implication problem (and, hence, the equivalence problem) for nested tgds is decidable, in contrast to SO tgds. 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 discovering new properties of the structure of the cores of universal solutions with respect to nested GLAV mappings. 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 Reference [12], 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)?

One of the chief goals in data exchange and data integration is to perform query answering. Hence, in addition to logical equivalence between mappings, one is also interested in *CQ-equivalence*. By definition, two mappings are CQ-equivalent if for every target conjunctive query q, the certain answers of q are the same for the two mappings. In general, logical equivalence implies CQ-equivalence, but not vice versa. However, logical equivalence and CQ-equivalence coincide for plain SO tgds and, in particular, for nested GLAV mappings. This can be seen by combining results from References [2] and [9]. More precisely, in Proposition 3.14 of Reference [9] the following property is shown: *Let $\mathcal{M}$ and $\mathcal{M}'$ be schema mappings such that both are preserved under target homomorphisms and both have the following property: For every instance I, if there is a solution for I, then there is a universal solution for I. Then, the fact that $\mathcal{M}$ and $\mathcal{M}'$ are CQ-equivalent implies that $\mathcal{M}$ and $\mathcal{M}'$ are logically equivalent.* Moreover, in Theorem 14 of Reference [2] it is shown that plain SO tgds possess these properties. Thus, even though we state and prove our results for logical equivalence only, they also hold for CQ-equivalence as well.

The remainder of the article 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 as well as 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. The article concludes with a discussion of open problems and directions for future research.

Relationship to previous work. This article is the full version of the conference paper [20]. This full version represents a significant expansion over the conference proceedings version. In particular, the body of the article includes detailed proofs of some of the main results and additional explanations. In addition, the present article contains an Appendix featuring some of the more technical proofs and a number of figures that facilitate the understanding of these proofs. In total, this submission contains roughly double the amount of material compared to the conference paper.

2 PRELIMINARIES

Schemas, Instances, and Homomorphisms. A *schema* $\mathbf{R}$ is a finite set $\{R_1, \ldots, R_k\}$ of relation symbols, where each R_i has a fixed arity ≥ 1. An *instance* I over $\mathbf{R}$, or an $\mathbf{R}$-*instance*, is a set $R_1^I \cup \ldots \cup R_k^I$, 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, when instance I is clear from the context.

The *active domain* $dom(I)$ of an instance I is the set of values occurring in the relations of I. We assume these values to be of two sorts, namely, *constants* and *(labeled) nulls*. By CONST and NULLS, we denote the disjoint sets of all constants and all nulls, respectively. Intuitively, nulls are used as placeholders for unknown constant values. We write $const(I)$ and $nulls(I)$ to denote, respectively, the set of constants and the set of nulls in $dom(I)$. A *fact* of an instance I over a schema $\mathbf{R}$ is a tuple occurring in one of the relations R_i of I. Such a fact is denoted $R_i(v_1, \ldots, v_m)$ or, for short, $R(\vec{a})$. Here, $\vec{a} = (v_1, \ldots, v_m)$ is a tuple of values from $dom(I)$, every v_i being either a constant or a null. The *empty tuple* is denoted by ().

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 that the active domains of source instances consist of constants; the active domains of target instances may consist of constants and nulls. Note that variables (in logical formulas) should not be confused with nulls (in target instances).

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 any two distinct 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 cardinality of the largest f-block of J.

Let J_1 and J_2 be two target instances. A function h that maps constants and nulls to constants and nulls is a *homomorphism* from J_1 to J_2 if the following holds: (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 $J_1 \rightarrow J_2$ 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 $J_1 \rightarrow J_2$ and $J_2 \rightarrow J_1$. 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 [17]. A homomorphism from J to J is called an *endomorphism*. An endomorphism is called *proper* if $h(v) \in$ CONST holds for at least one null v or $h(v_1) = h(v_2)$ holds for two distinct nulls v_1, v_2.

Example 2.1. Consider the following instance $I_1 = C_1 \cup C_2$ such that $C_1 = \{A(u_1, u_2), A(u_2, u_3)\}$ with nulls u_i and $C_2 = \{A(v_1, v_2), A(v_2, v_3), A(v_3, v_1)\}$ with nulls v_j. Intuitively, I_1 constitutes a directed graph where A is the edge relation and the nulls u_i and v_j are the vertices. We see that the smaller subinstance (i.e., subgraph) C_1 can be homomorphically mapped to C_2. This is witnessed by the homomorphism that maps each u_i to the respective v_i. Thus, we have that $core(I_1) = C_2$, as C_2 is homomorphically equivalent to the entire instance I_1.

The core looks quite different if we add a single "self-loop" (i.e., a fact $A(w, w)$ for some null w) to the instance. More precisely, let $I_2 = I_1 \cup S$ with $S = \{A(w, w)\}$ where w is a null. Then both C_1 and C_2 can be homomorphically mapped to S, that is, $core(I_2) = S$. In fact, it is a general observation that arbitrary graphs containing a self-loop "collapse" to the self-loop in the core. ◁

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*, which are logical formulas describing the relationship between $\mathbf{S}$ and $\mathbf{T}$. We say that $\mathcal{M}$ is *specified by* Σ; often, we will identify the set Σ of constraints with the mapping $\mathcal{M}$.

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 $J \rightarrow J'$. If C is a class of schema mappings, we say that C *admits universal solutions* if for every schema mapping $\mathcal{M}$ in 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 $\forall \vec{x}(\varphi(\vec{x}) \rightarrow \exists \vec{y}\psi(\vec{x}, \vec{y}))$, where $\varphi(\vec{x})$ is a conjunction of atoms over **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 **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. We refer to $\varphi(\vec{x})$ as the *left-hand side*, or *antecedent*, and $\exists \vec{y}\psi(\vec{x}, \vec{y})$ as the *right-hand side*, or *conclusion*.

Another name for s-t tgds is *global-and-local-as-view* (GLAV) constraints (see Reference [22]). We refer to a schema mapping specified entirely by a finite set of GLAV constraints as a *GLAV mapping*.

As shown in Reference [8], the class of GLAV mappings admits universal solutions. Moreover, if $\mathcal{M}$ is a GLAV mapping, then given a source instance I, the so-called "canonical" universal solution $chase(I, \mathcal{M})$ can be produced via the *oblivious chase procedure*, defined next.

Let $\forall \vec{x}(\varphi(\vec{x}) \rightarrow \exists \vec{y}\psi(\vec{x}, \vec{y}))$ be an s-t tgd in $\mathcal{M}$. For a conjunction of atoms $\varphi(\vec{x})$, we use $\vec{a}$ to define an *assignment* for $\vec{x}$ consisting of constants, such that the ith element of $\vec{a}$ replaces the ith variable of $\vec{x}$ in $\varphi(\vec{x})$, resulting in the formula $\varphi(\vec{a})$. We write $I \models \varphi(\vec{x})$ if there exists an assignment $\vec{a}$ for $\vec{x}$ such that for each atom $R(a_1, \ldots, a_k)$ of $\varphi(\vec{a})$, the fact $(a_1, \ldots, a_k)$ is in R^I. Such an assignment $\vec{a}$ is called a *satisfying assignment* for $\varphi(\vec{x})$ w.r.t. I (we omit the phrase "w.r.t. I" if I is clear from the context): this is denoted as $I \models \varphi(\vec{a})$. For every satisfying assignment $\vec{a}$ for $\varphi(\vec{x})$ w.r.t. I, the oblivious chase adds to the instance $chase(I, \mathcal{M})$ all atoms of $\psi(\vec{a}, \vec{B})$ as facts, where $\vec{B}$ is an assignment replacing all variables y in $\vec{y}$ with fresh, pairwise-distinct nulls. Such an extension of $chase(I, \mathcal{M})$ is performed for each satisfying assignment of each s-t tgd in $\mathcal{M}$.

egds. An *equality generating dependency* (egd) has the form $\forall \vec{x}(\varphi(\vec{x}) \rightarrow x_i = x_j)$, where the antecedent $\varphi(\vec{x})$ is a conjunction of atoms and x_i, x_j are the ith and the jth elements of $\vec{x}$, respectively. Sets of egds generalize functional dependencies. The chase of an instance I with the above egd proceeds as follows: Whenever $\vec{a}$ is a satisfying assignment for $\varphi(\vec{x})$ with $a_i \neq a_j$, then the chase replaces in I each occurrence of a_i with a_j. Typically in data exchange, egds on the target schema are defined, and only those unifications are allowed where at least one value of a_i, a_j is a null: If a_i and a_j are distinct constants, then the chase *halts with failure*. This convention does not apply here, since we use egds to specify *admissible source instances*.

SO tgds and Plain SO tgds. *Second-Order tgds*, or *SO tgds*, were introduced in Reference [11], 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 **S** be a source schema and **T** a target schema. A *second-order tuple-generating dependency (SO tgd)* is a formula of the form:

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

(1) $\vec{f}$ is a sequence of function symbols.

(2) Each φ_i is a conjunction of (i) relational atoms $S(y_1, \ldots, y_k)$, 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 $T(t_1, \ldots, t_l)$, 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 relational atom of φ_i.

As an example, the formula

$$\exists f(\forall e(Emp(e) \rightarrow Mgr(e, f(e))) \ \wedge \\ \forall e(Emp(e) \wedge (e = f(e)) \rightarrow SelfMgr(e)))$$

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 that 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:

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

The properties of plain SO tgds were recently investigated in Reference [2]. It is easy to see that every GLAV schema mapping is logically equivalent to a plain SO tgd. Moreover, as shown in Reference [11], the class of SO tgds admits universal solutions, hence, the same holds true for the class of plain SO tgds. In fact, the oblivious chase procedure can be extended to SO tgds, so that if σ is an SO tgd and I is a source instance, then $chase(I, \sigma)$ is the *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. Let X and Y be disjoint sets of variables. Consider the following recursive syntax rule:

$$\chi := \forall \vec{x} \, (\beta_1 \wedge \ldots \wedge \beta_k \rightarrow \exists \vec{y} \, (\alpha_1 \wedge \ldots \wedge \alpha_n \wedge \chi_1 \wedge \ldots \wedge \chi_\ell)), \tag{1}$$

where (i) n, ℓ are nonnegative integers, (ii) $\beta_1, \ldots, \beta_k$ with $k \geq 1$ are constant-free atoms whose relation names belong to the source schema, (iii) $\alpha_1, \ldots, \alpha_n$ are constant-free atoms whose relation names belong to the target schema, (iv) every variable that occurs in $\vec{x}$ also occurs in some atom among $\beta_1, \ldots, \beta_k$, (v) every variable occurring in some atom in $\beta_1, \ldots, \beta_k$ also belongs to X; and (vi) every variable that occurs in $\vec{y}$ belongs to Y.

An empty conjunction is understood to be equivalent to **true**. In particular, if $\ell = 0$, then $\chi_1 \wedge \ldots \wedge \chi_\ell \equiv$ **true**, which is the base case of the recursion.

A *nested tgd* is a first-order sentence that can be generated by the recursive syntax rule (1). It can be assumed w.l.o.g. that every variable that occurs in a nested tgd is quantified exactly once by either an existential or universal quantifier. By *subformula* τ of a nested tgd, we will always denote an implication $\forall \vec{x}(\varphi(\vec{x}_0, \vec{x}) \rightarrow \exists \vec{y} \ \psi(\vec{x}_0, \vec{x}, \vec{y}_0, \vec{y}))$ generated by one of the symbols $\chi_1 \ldots \chi_\ell$ in the syntax rule (1). Here, $\vec{x}_0$ and $\vec{y}_0$ are tuples of free variables, respectively, universally and existentially quantified outside the subformula in the nested tgd. For instance, the conclusion $\psi(\vec{x}_0, \vec{x}, \vec{y}_0, \vec{y})$ of τ may contain a conjunct $\tau' = \forall \vec{x}'(\varphi'(\vec{x}'_0, \vec{x}') \rightarrow \exists \vec{y}' \, \psi'(\vec{x}'_0, \vec{x}', \vec{y}'_0, \vec{y}'))$ where $\vec{x}'_0$ is a tuple of free variables containing both the free variables $\vec{x}_0$ of τ and the universally quantified variables $\vec{x}$ of τ. Our convention is that variables in such a tuple $\vec{x}'_0$ follow the order in which they are quantified in the nested tgd, that is, $\vec{x}'_0$ is the concatenation $\vec{x}_0 \cdot \vec{x}$.

$$\forall x_1\big(S_1(x_1) \rightarrow \exists y_1(\mathbf{true})\big)$$
$$\downarrow$$
$$\forall x_2\big(S_2(x_2) \rightarrow R_2(y_1, x_2)\big)$$
$$\downarrow$$
$$\forall x_3\big(S_3(x_1, x_3) \rightarrow R_3(y_1, x_3)\big)$$
$$\downarrow$$
$$\forall x_4\big(S_4(x_3, x_4) \rightarrow \exists y_2(R_4(y_2, x_4))\big)$$

Fig. 1. Tree representation of σ.

Generating a nested tgd σ by the recursive syntax rule (1) yields a tree representation of σ, such that the root node of this tree is labeled by the subformula stemming from instantiating $\forall \vec{x}\,(\beta_1 \wedge \ldots \wedge \beta_k \rightarrow \exists \vec{y}\,(\alpha_1 \wedge \ldots \wedge \alpha_n))$. Moreover, the root has ℓ subtrees, which correspond to the subformulas generated by $\chi_1, \ldots \chi_\ell$. We refer to the subformulas labeling the nodes in this tree representation as *parts* of σ. The child, parent, ancestor, and descendant relations of nodes in the tree naturally carry over to the parts of σ. For a part σ_i of σ, we thus write $parent(\sigma_i)$ to denote the parent σ_j of σ_i (i.e., σ_j is the label of the parent node of the node labeled by σ_i) and we write $child(\sigma_i)$, $anc(\sigma_i)$, and $desc(\sigma_i)$ for the set of children, ancestors, and descendants, respectively, of σ_i. The part labeling the root node will be referred to as the *root part* of σ. Every part of σ will be assumed to be uniquely identified by a positive integer index. The notation $\sigma_i(\vec{x}_i)$ will be used to explicitly specify the free variables $\vec{x}_i$ of the part σ_i. These are the variables quantified in $anc(\sigma_i)$.

Example 2.2. Assume that the nested tgd σ has the following form:

$$\forall x_1\Big(S_1(x_1) \rightarrow \exists y_1\Big(\forall x_2(S_2(x_2) \rightarrow R_2(y_1, x_2)) \wedge \forall x_3\Big(S_3(x_1, x_3) \rightarrow (R_3(y_1, x_3) \wedge \forall x_4(S_4(x_3, x_4) \rightarrow \exists y_2 R_4(y_2, x_4)))\Big)\Big)\Big).$$

There are four parts in σ (see Figure 1 for a tree representation of σ):

- σ_0: $\forall x_1\big(S_1(x_1) \rightarrow \exists y_1(\mathbf{true})\big)$
- $\sigma_1(x_1, y_1)$: $\forall x_2\big(S_2(x_2) \rightarrow R_2(y_1, x_2)\big)$
- $\sigma_2(x_1, y_1)$: $\forall x_3\big(S_3(x_1, x_3) \rightarrow R_3(y_1, x_3)\big)$
- $\sigma_3(x_1, x_3, y_1)$: $\forall x_4\big(S_4(x_3, x_4) \rightarrow \exists y_2(R_4(y_2, x_4))\big)$

Skolemization. In the above examples, parts of a nested tgd contain free variables that correspond to the existentially quantified variables of σ. More convenient is the Skolemized form of the nested tgd, in which free variables of the parts correspond to the universally quantified variables of σ. We use the standard first-order Skolemization procedure. Let SK denote a set of Skolem functional symbols disjoint from the set of variables, CONST and NULLS. These function symbols are considered as existentially quantified. The Skolemization replaces each existentially quantified variable y in σ with a unique term $f(\vec{x})$, where $f \in \mathsf{SK}$ and $\vec{x}$ is the tuple of those universally quantified variables that occur in the same part as $\exists y$ or in some ancestor thereof. For a part σ_i, the set $vars(\sigma_i)$ contains all Skolem terms replacing variables that were existentially quantified in σ_i before Skolemization.

Example 2.2 (continued). The Skolemization of the nested tgd σ is:

$$\forall x_1\Big(S_1(x_1) \rightarrow \Big(\forall x_2(S_2(x_2) \rightarrow R_2(f_1(x_1), x_2)) \wedge \forall x_3\Big(S_3(x_1, x_3) \rightarrow (R_3(f_1(x_1), x_3) \wedge \forall x_4(S_4(x_3, x_4) \rightarrow R_4(f_2(x_1, x_3, x_4), x_4)))\Big)\Big)\Big).$$

The parts of σ after Skolemization are given below.

- $\sigma_0 = \forall x_1(S_1(x_1) \rightarrow (\mathbf{true}))$; $vars(\sigma_0) = \{f_1(x_1)\}$,
- $\sigma_1(x_1) = \forall x_2(S_2(x_2) \rightarrow R_2(f_1(x_1), x_2))$; $vars(\sigma_1) = \emptyset$,
- $\sigma_2(x_1) = \forall x_3(S_3(x_1, x_3) \rightarrow R_3(f_1(x_1), x_3))$; $vars(\sigma_2) = \emptyset$,
- $\sigma_3(x_1, x_3) = \forall x_4(S_4(x_3, x_4) \rightarrow R_4(f_2(x_1, x_3, x_4), x_4))$; $vars(\sigma_3) = \{f_2(x_1, x_3, x_4)\}$.

From now on, we only consider Skolemized tgds unless explicitly specified otherwise. As has become apparent in the previous example, after Skolemization, the parts no longer contain as free variables the existentially quantified variables of ancestor parts. We conclude this section by drawing a connection between nested tgds and SO tgds. Although syntactically, nested tgds differ from SO tgds, which do not have nested implications, a nested tgd can be converted into an SO tgd.

E.g., a conversion procedure in Reference [14] first performs a Skolemization and then extends the antecedent in each subformula σ_i generated by the symbol χ_i in rule (1) by adding to it the implication antecedents of all ancestor subformulas of σ_i as conjuncts. A nested tgd can be shown equivalent to the resulting conjunction of its Skolemized parts, which is syntactically an SO tgd.

A *nested GLAV mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of nested tgds. Unless mentioned otherwise, in this article, we will always assume dependencies (tgds, nested tgds, and SO tgds) to be source-to-target dependencies.

3 THE IMPLICATION PROBLEM

Let $\mathbf{S}$, $\mathbf{T}$ be two relational schemas and $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$ be schema mappings, where Σ and Σ' are expressed in some logical formalism (e.g., SO tgds or nested tgds). We say that Σ implies Σ', denoted by $\Sigma \models \Sigma'$, if for every instance I of $\mathbf{S}$ and every instance J of $\mathbf{T}$ 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 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. In Section 5, we will study the implication and equivalence problems of schemas *in the presence of a set Σ_s of source constraints*. In this case, only those source instances I are considered that actually satisfy Σ_s. For instance, we will then say that Σ implies Σ' if for every instance I of $\mathbf{S}$ *with* $I \models \Sigma_s$ and every instance J of $\mathbf{T}$ such that $(I, J) \models \Sigma$, we have that $(I, J) \models \Sigma'$.

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.*

We start by giving an informal, high-level explanation of the implication algorithm. All concepts used in this explanation (such as the chase with nested tgds) will be made precise below. Let σ be a nested tgd and Σ be a set of nested tgds. The proof of Theorem 3.1 will establish that if $\Sigma \not\models \sigma$, then there exists a pair of "canonical" source and target instances (I_p, J_p) that satisfies σ such that there is no homomorphism from J_p to $chase(I_p, \Sigma)$. The pair of instances $(I_p, chase(I_p, \Sigma))$ then satisfies Σ and falsifies σ, witnessing $\Sigma \not\models \sigma$. The canonical instances are constructed as minimal and in a sense most general instances enabling a chase of I_p with σ, such that the number of *triggerings*

of each part of σ is bounded and the *pattern* of the chase has a certain form. This bound and the number of relevant chase patterns will appear quite high (in fact, non-elementary in the size of σ and Σ). In Section 4, we will show that this limit exhibits a crucial and so far unexplored difference between nested tgds and SO tgds.

The decision procedure behind Theorem 3.1 requires the introduction of several notions and technical tools that are presented in what follows:

Chase Forest. The oblivious chase of an instance with a nested tgd can be described as a sequence of recursive *triggerings*. To specify how an instance of the target schema $\mathbf{T}$ is produced by the chase, we define an injective function μ taking a term $f(\vec{b})$ with $f \in \mathsf{SK}$, $\vec{b} \in \mathsf{CONST}$, and returning a null $v \in \mathsf{NULLS}$. To save parentheses, we denote the null $\mu(f(\vec{b}))$ corresponding to the term $f(\vec{b})$, by $\hat{f}(\vec{b})$. Note that so encoded function μ applies to the whole term $f(\vec{b})$, even though, for the sake of readability, the hat symbol only appears over the function symbol f and not over the arguments.

Definition 3.2 (Triggering). Let I be an instance and σ a nested tgd. A *triggering* t of a part σ_i of σ (in its Skolemized form) with $\sigma_i = \forall\vec{x}(\varphi(\vec{x}_0, \vec{x}) \rightarrow \psi(\vec{x}_0, \vec{x}))$, is a triple $(\sigma_i, \vec{a}_0, \vec{a})$, where $\vec{a}$ is a tuple of constants assigned to the variables $\vec{x}$ and $\vec{a}_0$ is a tuple of constants assigned to the free variables $\vec{x}_0$ of σ_i such that $I \models \varphi(\vec{a}_0, \vec{a})$ holds. The set $facts(t)$ consists of all facts in $\psi(\vec{a}_0, \vec{a})$ obtained by substituting each instantiation of a Skolem term $f(\vec{a}')$ in $\psi(\vec{a}_0, \vec{a})$ with $\hat{f}(\vec{a}') = \mu(f(\vec{a}'))$, where $\vec{a}'$ is a prefix of the tuple $\vec{a}_0{\cdot}\vec{a}$. We define the set $nulls(t)$ to be $\emptyset$ if the tuple $\vec{x}$ is empty, otherwise, we set $nulls(t) = \{\hat{f}(\vec{a}_0{\cdot}\vec{a}) \mid f(\vec{a}_0{\cdot}\vec{a})$ occurs in $\psi(\vec{a}_0, \vec{a})\}$. That is, $nulls(t)$ is a set of nulls instantiating Skolem terms that occur in the part σ_i but not in its ancestor parts in σ, that is, all Skolem terms in the set $vars(\sigma_i)$. ◁

Definition 3.3 (Chase with a nested tgd, Chase forest). Let I be an instance and σ a nested tgd. Analogously to the tree structure of the parts of σ, we define the following tree structure of triggerings: Let $\sigma_i = \forall\vec{x}(\varphi_i(\vec{x}_0, \vec{x}) \rightarrow \psi_i(\vec{x}_0, \vec{x}))$ be a part of σ and let $t = (\sigma_i, \vec{a}_0, \vec{a})$ be a triggering:

- if σ_i is the root part of σ (in which case $\vec{a}_0$ is empty), then t is called a *root triggering*;
- if $parent(\sigma_i) = \sigma_j$ and there is a triggering $(\sigma_j, \vec{b}_0, \vec{b})$ with $\vec{a}_0 = \vec{b}_0{\cdot}\vec{b}$, then $(\sigma_j, \vec{b}_0, \vec{b})$ is called the *parent triggering* of t.

The *chase forest* $\mathcal{F}_{\sigma,I} = (V_{\sigma,I}, T_{\sigma,I})$ of I with σ is a rooted forest (i.e., a forest with a distinguished root for each connected component) whose set of vertices $V_{\sigma,I}$ is the set of all triggerings and where $T_{\sigma,I}$ contains an edge (t', t) if t' is the parent triggering of t. For a set Σ of nested tgds, the chase forest $\mathcal{F}_{\Sigma,I}$ of I with Σ is a pair $(V_{\Sigma,I}, T_{\Sigma,I})$ with $V_{\Sigma,I} = \bigcup_{\sigma\in\Sigma} V_{\sigma,I}$ and $T_{\Sigma,I} = \bigcup_{\sigma\in\Sigma} T_{\sigma,I}$.

The *chase of I with a nested tgd σ*, denoted as $chase(I, \sigma)$, is defined as the union of all facts of all triggerings in $\mathcal{F}_{\sigma,I}$, i.e., $chase(I, \sigma) = \bigcup_{t\in V_{\sigma,I}} facts(t)$. Likewise, the *chase of I with a set Σ of nested tgds*, denoted as $chase(I, \Sigma)$, is defined as the union of all facts of all triggerings in $\mathcal{F}_{\Sigma,I}$, i.e., $chase(I, \Sigma) = \bigcup_{t\in V_{\Sigma,I}} facts(t)$. ◁

Consider a triggering t in $\mathcal{F}_{\Sigma,I}$. We refer to the subtree of $\mathcal{F}_{\Sigma,I}$ rooted at t as the *chase tree* of t. The set of vertices in this subtree will be denoted as $desc(t)$. Ancestor triggerings of t are all vertices belonging to the path from the root to the parent of t in $\mathcal{F}_{\Sigma,I}$. As follows from Definition 3.3, free variables $\vec{x}_0$ of the part associated with t are bound in the ancestor triggerings of t to a tuple of constants $\vec{a}_0$. We shall refer to $\vec{a}_0$ as the *input assignment* of t. If t is a parent triggering of t_1, then we call t_1 a *child triggering* of t. If t_1 and t_2 are child triggerings of t, then we call t_1 and t_2 *sibling triggerings*.

Procedure IMPLIES(Σ,σ)

Data: Set Σ of nested tgds, nested tgd σ
Result: *true* if $\Sigma \models \sigma$, *false* otherwise
Skolemize σ and Σ in a standard way;
Let v_σ be the number of distinct Skolem functions in σ;
Let w_Σ be the maximum number of universally quantifiedvariables in a nested tgd in Σ;
Let $k = v_\sigma \cdot w_\Sigma + 1$;
Let $\mathcal{P}_k(\sigma)$ be the set of k-patterns of σ;
for *each k-pattern* $p^k \in \mathcal{P}_k(\sigma)$ **do**
    Let I_{p^k}, J_{p^k} be the canonical source and, respectively, the canonical target instances of p^k;
    if *no homomorphism from* J_{p^k} *to chase*(I_{p^k}, Σ) *exists* **then**
        return *false*;
    end
end
return *true*;

PROPOSITION 3.4. *Let I be a source instance and let Σ be a set of nested tgds. Then, the instance chase(I, Σ) is a* universal solution *for I under Σ. Namely, (i) $I \cup chase(I, \Sigma) \models \Sigma$ and (ii) for each J such that $(I, J) \models \Sigma$, we have $chase(I, \Sigma) \rightarrow J$.*

PROOF. We refer to Reference [14] for a procedure converting a nested tgd into an equivalent SO tgd, essentially by Skolemizing it, and taking a conjunction of all parts of a nested tgd, where the antecedent of each nested part is extended with the antecedents of all its ancestors. From this, we can conclude that Σ is logically equivalent to an SO tgd σ. It is easy to verify that the chase of such an SO tgd [11] creates exactly the same instance (up to renaming of nulls) as the chase in Definition 3.3. Since the chase of I with σ results in a universal solution for I under σ [11], and σ is logically equivalent to Σ, we have that $chase(I, \Sigma)$ is a universal solution for I under Σ. □

The second property of the chase, underpinning the decision procedure, is that facts produced by triggerings in distinct chase trees share no nulls. A consequence of this observation is that reasoning about nested tgds may be restricted to source instances that give rise to a single chase tree. Finally, we will show that we only need to consider chase trees whose fan-out (i.e., the number of child nodes of any vertex) has an upper bound that depends on the mapping but not on any particular source instance.

Patterns, Canonical Instances of Patterns. The algorithm behind Theorem 3.1 is described in the decision procedure IMPLIES. We now introduce the notions used in this procedure.

Definition 3.5 (Pattern). Let σ be a nested tgd. A *pattern* of σ is a vertex-labeled rooted tree whose labels are the parts $\sigma_1, \sigma_2, \ldots$ of σ, such that if v is a child of v' in the pattern, then $label(v) \in child(label(v'))$.

Let Σ be a set of nested tgds, $\sigma \in \Sigma$, let I be a source instance, and let T be a tree in the chase forest $\mathcal{F}_{\Sigma,I}$ of I with Σ. The *pattern of T* is the vertex-labeled tree obtained from T by relabeling each vertex with label $(\sigma_i, \vec{a}_0, \vec{a})$ by σ_i. ◁

Note that the pattern of any tree T in a chase forest $\mathcal{F}_{\Sigma,I}$ is a pattern of some $\sigma \in \Sigma$. Conversely, not every pattern of a nested tgd σ is necessarily attainable as the pattern of some tree T in the chase forest for some source instance I. For instance, consider the nested tgd $\sigma = \forall x_1 (S_1(x_1) \rightarrow ((S_2(x_1) \rightarrow T(x_1))))$ with a single nested part. This tgd can only generate chase trees with patterns having at most two nodes. This is because the assignment of the only variable x_1

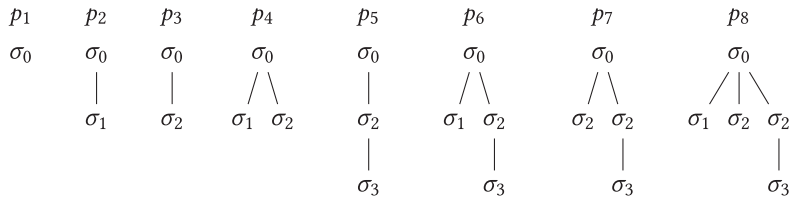

Fig. 2. 1-patterns of the tgd σ with the parts $\sigma_0, \ldots, \sigma_3$ from Example 2.2.

is determined by the root triggering, and thus only a single triggering of the nested part is possible. However, suppose that we modify σ to $\sigma' = \forall x_1\,(S_1(x_1) \rightarrow (\forall x_2(S_2(x_2) \rightarrow T(x_1, x_2)))$. Then, for appropriately chosen source instance I, the chase forest $\mathcal{F}_{\sigma',I}$ can give rise to patterns with an arbitrarily big number of nodes. For instance, for the source instance $I_n = \{S_1(a), S_2(b_1), \ldots, S_2(b_n)\}$, we get a pattern whose root node has n child nodes.

We also use the notion of "subtree clones" in a pattern. Manipulating patterns by adding or removing clones will be a basic technique in many proofs of this article.

Definition 3.6 (Pattern subtree, clone, k-pattern). Let σ be a nested tgd and let p be a pattern of σ. By a *subtree* of a pattern p, we mean a complete subtree with all descendant nodes. We call t' a *clone* of t if t, t' are isomorphic subtrees and their roots have the same parent node. Let C_t denote the set containing t and all its clones in p. If for each subtree t, $|C_t| \leq k$, we call p a *k-pattern*. $\mathcal{P}_k(\sigma)$ is the set of all k-patterns of σ. ◁

Example 3.7. Recall the nested tgd σ with four parts from Example 2.2. The set $\mathcal{P}_1(\sigma) = \{p_1, \ldots, p_8\}$ containing all 1-patterns of σ is shown in Figure 2. ◁

Proposition 3.8. *For a positive integer k and a nested tgd σ, the set of k-patterns of σ is finite.*

Proof. Note that each part σ_i of σ can be obtained from some subformula $\forall\vec{x}(\varphi \rightarrow \psi)$ of σ by omitting conjuncts with implications from ψ. Let $\lceil\cdot\rceil$ denote the function taking the part index i and retrieving the subformula of σ corresponding to the part σ_i: $\lceil i\rceil = \sigma_i(\vec{x}_i)$ where $\vec{x}_i$ denotes the free variables of the subformula. Syntactically, the sentence $\forall\vec{x}_i\,\sigma_i(\vec{x}_i)$ is a nested tgd. By slight abuse of notation, we use $\mathcal{P}_k(\lceil i\rceil)$ to denote the set of k-patterns of this nested tgd. Clearly, $\mathcal{P}_k(\lceil i\rceil)$ coincides with the set of all distinct subtrees in $\mathcal{P}_k(\sigma)$ with the root labeled by the part σ_i. We now proceed by structural induction on σ to prove the proposition.

Base case: $\lceil i\rceil$ contains no nested implications. The set of k-patterns $\mathcal{P}_k(\lceil i\rceil)$ then consists of a single tree whose single node is labeled σ_i.

Now let $\lceil i\rceil = \forall\vec{x}(\varphi_i \rightarrow \psi_i)$ be such that ψ_i is $\psi_\alpha \wedge \lceil i_1\rceil \wedge \ldots \wedge \lceil i_n\rceil$ where ψ_α is a conjunction of atoms and $\lceil i_1\rceil, \ldots, \lceil i_n\rceil$ are subformulas $\forall\vec{x}_{i_j}(\varphi_{i_j} \rightarrow \psi_{i_j})$, for $j \in \{1 \ldots n\}$. The inductive hypothesis is that the sets $\mathcal{P}_k(\lceil i_1\rceil), \ldots, \mathcal{P}_k(\lceil i_n\rceil)$ of k-patterns are finite. The set $\mathcal{P}_k(\lceil i\rceil)$ contains all patterns that have (i) the root node labeled with σ_i, and (ii) up to k copies of each pattern from the set $\bigcup_{j\in\{1\ldots n\}} \mathcal{P}_k(\lceil i_j\rceil)$ as subtrees rooted at child nodes of the root. Since each set $\mathcal{P}_k(\lceil i_j\rceil)$ is finite, $\mathcal{P}_k(\lceil i\rceil)$ is finite as well. It follows that also the set $\mathcal{P}_k(\sigma)$ is finite (albeit non-elementary in k). □

Definition 3.9 (Canonical chase tree, canonical instances of a pattern). Let p be a pattern of a nested tgd σ. The *canonical chase tree* T_p of p is a labeled tree isomorphic to p: for every node labeled σ_i in p, there is a unique triggering $t = (\sigma_i, \vec{a}_0, \vec{a})$ in T_p, such that $\vec{a}$ assigns fresh, pairwise-distinct constants to the universally quantified variables of σ_i, and $\vec{a}_0$ is either empty if t is a root triggering, or equal to $\vec{b}_0{\cdot}\vec{b}$ if $(\sigma_j, \vec{b}_0, \vec{b})$ is the parent triggering of t. The *canonical source instance* I_p, respectively, *canonical target instance* J_p, of the chase tree T_p are defined as minimal instances

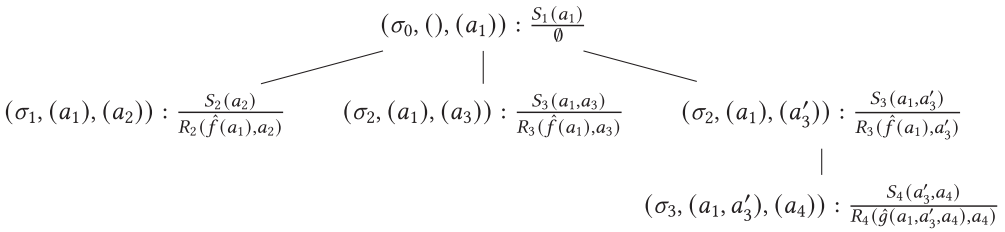

Fig. 3. Canonical chase tree of the pattern p_8 and the facts of its canonical source instance I_{p_8} (above the bars) and the canonical target instance J_{p_8} (below the bars).

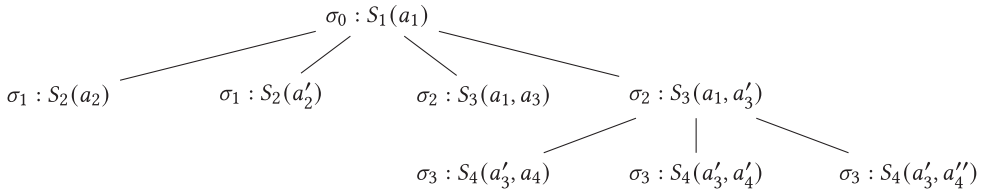

Fig. 4. A 3-pattern and the facts constituting its canonical source instance.

such that for each node $t = (\sigma_i, \vec{a}_0, \vec{a})$ of T_p with $\sigma_i = \forall \vec{x}(\varphi(\vec{x}_0, \vec{x}) \rightarrow \psi(\vec{x}_0, \vec{x}))$, the atoms of $\varphi(\vec{a}_0, \vec{a})$ are contained in I_p and the atoms of $\psi(\vec{a}_0, \vec{a})$—substituting every instantiation $f(\vec{a}')$ of a Skolem term in $\psi(\vec{a}_0, \vec{a})$ by a fresh null $\hat{f}(\vec{a}')$—are in J_p, i.e., $facts(t) \subseteq J_p$ holds. ◁

Note that the canonical target instance J_p is, in general, not a solution of the canonical source instance I_p, since it may not be fully chased. Moreover, note that we often speak of *the* canonical source and target instances of a pattern p instead of T_p, even though the constants used to create T_p from p can be arbitrary. The justification is that such instances are unique up to renaming of constants, and the concrete names of the constants are irrelevant for our further considerations.

Example 3.10. Figure 3 shows the canonical chase tree T_{p_8}, canonical source instance I_{p_8}, and the canonical target instance J_{p_8} of the 1-pattern p_8 from Example 3.7. ◁

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

Example 3.11. One possible 3-pattern based on the 1-pattern p_8 from Example 3.10, in which one clone of the node σ_1 and two clones of the node σ_3 are added, is shown in Figure 4, along with the facts of its canonical source instance. ◁

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

Example 3.12. Consider the nested tgd τ and the s-t tgds τ' and τ'':

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

We now use the procedure Implies to test if any of τ', τ'' implies τ. The first step is the Skolemization of τ:

$$\tau_1 : \quad \forall x_1\Big(S_1(x_1) \rightarrow$$
$$\tau_2 : \qquad \forall x_2(S_2(x_2) \rightarrow R(x_2, f(x_1)))\Big).$$

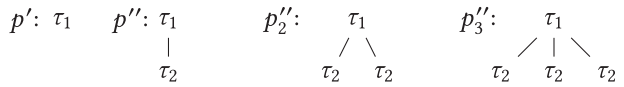

Fig. 5. Patterns used to test $\tau' \models \tau$ and $\tau'' \models \tau$.

According to line 4 of the procedure, the bound k on the number of clones should be 3 for testing $\tau' \models \tau$ and $\tau'' \models \tau$, since we have $v_\tau = 1$ and $w_{\{\tau'\}} = w_{\{\tau''\}} = 2$. 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 $\{p', p'', p''_2, p''_3\}$ in Figure 5 is actually the complete set of 3-patterns of τ.

Let I_p and J_p denote the canonical source, respectively, 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 $\Sigma \models \tau$ holds, the procedure Implies checks the existence of a homomorphism from J_p to $chase(I_p, \Sigma)$ for all patterns $p \in \{p', p'', p''_2, p''_3\}$ (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:

$$I_{p''_2} = \{S_1(a_1), S_2(a_2), S_2(a'_2)\},\ J_{p''_2} = \{R_2(a_2, \hat{f}(a_1)), R_2(a'_2, \hat{f}(a_1))\}.$$

The Skolemization of τ' yields $\forall x_1 \forall x_2\,(S_1(x_1) \wedge S_2(x_2) \rightarrow R(x_2, g(x_1, x_2)))$ and the chase of $I_{p''_2}$ with τ' results in $J_{\tau'} = \{R(a_2, g(a_1, a_2)),\ R(a'_2, g(a_1, a'_2))\}$. There is no homomorphism from $J_{p''_2}$ to $J_{\tau'}$ and thus Implies$(\{\tau'\}, \tau)$ outputs *false*, indicating $\tau' \not\models \tau$. The failure of implication is because the existentially quantified variable in τ' depends on both x_1 and x_2, but only on x_1 in τ.

We now check if $\tau'' \models \tau$ 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 $[\hat{f}(a_1) \mapsto a_1]$ is a homomorphism from $J_{p''_2}$ to $J_{\tau''}$, and thus the test at line 8 of the procedure Implies passes successfully for the pattern p''_2. One can verify that so do the checks for the patterns p'' and p''_3. Therefore, Implies$(\{\tau''\}, \tau)$ outputs *true*, indicating $\tau'' \models \tau$. Indeed, where τ'' yields the value of x_1, τ only requires *some* value (that is, a null) determined by the assignment of x_1, which is a strictly weaker condition. ◁

3.1 Proof of Theorem 3.1

Two ideas underlie the correctness of the procedure Implies. The first is a well-known property of schema mappings that are closed under target homomorphisms and allow for universal solutions for every source instance. Namely, $\Sigma \models \sigma$ if and only if for every source instance I, there is a homomorphism from a universal solution J_I^σ for I under σ into all solutions for I under Σ. By Proposition 3.4 it suffices to homomorphically embed $chase(I, \sigma)$ into $chase(I, \Sigma)$, which is equivalent to embedding every f-block of $chase(I, \sigma)$ in $chase(I, \Sigma)$ independently [10]. This is not a feasible test, however, since enumerating all source instances is required. Our main finding in this section is how to address this issue for nested tgds: We show that for arbitrary I, $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds if for every k-pattern $p^k \in \mathcal{P}_k(\sigma)$, a homomorphism from J_{p^k} to $chase(I_{p^k}, \Sigma)$ exists, where k is a constant depending on σ and Σ, as defined at line 4 of the procedure Implies, and I_{p^k} and J_{p^k} are the canonical source instance and, respectively, the canonical target instance of p^k. That is, instead of an infinite number of source instances, one can enumerate a finite set of patterns.

The proof of this latter idea makes use of special notation and basic facts about chase trees, which we present next.

Active domain of the chase result. The chase with nested tgds identifies nulls with instantiations of Skolem terms. By Definition 3.3, for every fact $R(e_1, \ldots, e_m)$ in $chase(I, \Sigma)$ and for every $i \in \{1, \ldots, m\}$, e_i is either a constant from $dom(I)$ or a null $\hat{f}(\vec{a}) = \mu(f(\vec{a}))$, where $f(\vec{x})$ is a Skolem

function in Σ, $\vec{a}$ is a tuple of constants from $dom(I)$ instantiating the variables $\vec{x}$ during the chase, and μ is an injective function mapping the Skolem term $f(\vec{a})$ onto the null $\hat{f}(\vec{a}) \in \mathsf{NULLS}$. By the injectivity of μ, we can define an inverse function μ^{-1} from NULLS to the set of Skolem terms $\mathsf{SK} \times \mathsf{CONST}^*$. This inverse function allows us to formally define an operation of extracting all nulls used to build a chase instance.

Definition 3.13. Let J be a target instance produced by the chase with a nested tgd. We define the set $cst(J)$ as the set containing $dom(J) \cap \mathsf{CONST}$ plus all constants occurring within terms $\mu^{-1}(e)$ for every $e \in dom(J) \cap \mathsf{NULLS}$. ◁

The relationship between the sets $dom(J)$ and $cst(J)$ is thus the following: If J consists solely of constants (e.g., J is a source instance), then we have $dom(J) = cst(J)$. Otherwise, $dom(J) \setminus cst(J)$ is the set of nulls of J and $cst(J) \setminus dom(J)$ is the set of constants that only occur in the tuple $\vec{a}$ of arguments of a term $f(\vec{a})$ associated by μ with some null in $dom(J)$. As specified above, we use $\hat{f}(\vec{a})$ as a shorthand for $\mu(f(\vec{a}))$.

Independence of chase trees. Note that by the definition of the chase, nulls created by the chase with nested tgds correspond to Skolem terms of a specific structure: If a null $\hat{f}(\vec{a})$ is introduced by a non-root triggering t for the first time in the chase, then all the constants used in any of the ancestor triggerings of t occur in a prefix of $\vec{a}$. This observation provides a means of restricting the set of nulls that connect chase fragments generated by different triggerings.

Lemma 3.14. *Let Σ be a set of nested tgds, let I be a source instance for Σ, and let $J = chase(I, \Sigma)$. Consider a triggering $(\sigma_i, \vec{a}_0, \vec{a})$ in the chase forest of I with Σ and a subinstance J' of J defined as follows: $J' = \bigcup\{facts(t) \mid t \in desc((\sigma_i, \vec{a}_0, \vec{a}))\}$. If a null $\hat{f}(\vec{b})$ occurs both in J' and in $J \setminus J'$, then (i) the variables $\vec{x}$ in the Skolem term $f(\vec{x})$, which $f(\vec{b})$ instantiates, appear free in σ_i and (ii) $\vec{b}$ is a prefix of $\vec{a}_0$.*

Proof. We first prove by contradiction that $\vec{b}$ is a prefix of $\vec{a}_0$. Suppose that $\vec{b}$ is not a prefix of $\vec{a}_0$. By the definition of triggering and the convention that variables in Skolem terms appear in the order they are quantified in the nested tgd, each null in J' has the form $g(\vec{c})$ such that either $\vec{c} = \vec{a}_0{\cdot}\vec{a}$ or the shorter tuple in the pair $(\vec{c}, \vec{a}_0{\cdot}\vec{a})$ is a proper prefix of the longer one. Therefore, it must be the case that $\vec{a}_0$ is a proper prefix of $\vec{b}$. Since J' consists of facts generated by triggerings in $desc((\sigma_i, \vec{a}_0, \vec{a}))$, we have that $\vec{b}$ must be prefixed by $\vec{a}_0{\cdot}\vec{a}$. From this and the assumption $\hat{f}(\vec{b}) \in dom(J')$, we have that the Skolem function f replaces a variable existentially quantified in $desc(\sigma_i)$. Let $t' = (\sigma_j, \vec{a}'_0, \vec{a}')$ be an arbitrary triggering such that $\hat{f}(\vec{b})$ occurs in $facts(t')$ and $t' \notin desc(t)$. The existence of t' follows from the assumption of the lemma that $\hat{f}(\vec{b})$ occurs in the subinstance $chase(I, \Sigma) \setminus \bigcup\{facts(t) \mid t \in desc((\sigma_i, \vec{a}_0, \vec{a}))\}$. Since $\sigma_j \in desc(\sigma_i)$ and $\vec{b}$ is prefixed by $\vec{a}_0{\cdot}\vec{a}$, we have that t' must be contained in $desc(t)$, which is a contradiction. We have shown that $\vec{b}$ must be a prefix of $\vec{a}_0$, and since $\vec{a}_0$ is the input assignment of $(\sigma_i, \vec{a}_0, \vec{a})$, $\vec{b}$ instantiates exclusively the variables that occur free in σ_i. □

The following corollary of Lemma 3.14 is crucial for the correctness of the procedure Implies:

Corollary 3.15. *Consider a finite set Σ of nested tgds, a source instance I, and subinstances J_1, J_2 of $chase(I, \Sigma)$ produced by distinct chase trees in the chase forest of I with Σ. That is, $J_1 = \bigcup\{facts(t) \mid t \in desc(t_1)\}$, $J_2 = \bigcup\{facts(t) \mid t \in desc(t_2)\}$ for distinct root triggerings t_1, t_2. Then, J_1 and J_2 have no nulls in common.*

Proof. By Lemma 3.14, a null $\hat{f}(\vec{b})$ occurring both in J_1 and in J_2 must be such that $\vec{b}$ is a prefix of the input assignment of both t_1 and t_2, and thus (since t_1 and t_2 are root triggerings) $\vec{b}$ must be the empty tuple. However, the arguments of a Skolem term have to be non-empty. □

Transforming patterns by increments. We now turn to k-patterns as the main tool of the procedure Implies. The following definition facilitates the reasoning about patterns obtained by inserting new clones of subtrees.

Definition 3.16 (k^+-increment of a pattern). Let p be a pattern of a nested tgd σ. An *increment* of p is obtained by (i) choosing a node v of p, (ii) choosing a subtree π rooted at some child node of v, and (iii) adding to p a clone π' of π.

An increment is called a k^+-increment of p if, in the above definition of an increment, the newly inserted subtree π' has at least k clones in p (and thus becomes one of at least $k+1$ isomorphic subtrees rooted at v).

We write $p \leq p'$ if pattern p' can be obtained from a pattern p by a sequence of zero or more increments, and we write $p \leq_k p'$ if p' can be obtained from p by a sequence of zero or more k^+-increments. ◁

Equipped with this notation, we formulate a property of k-patterns that underpins the proof of Theorem 3.1. It will also be used in the proof of Theorem 4.14 in Section 4 on the structure of the core of nested tgds.

Lemma 3.17. *For every positive integer k and for every pattern p of σ, there is a k-pattern $p^k \in \mathcal{P}_k(\sigma)$ such that $p^k \leq_k p$ holds. Moreover, p can be obtained from p^k by applying k^+-increment steps in such a way that whenever a subtree t is added, then for every subtree t' of t, the set $C_{t'}$ of all clones of t' (according to Definition 3.6) satisfies the condition $|C_{t'}| \leq k$.*

Intuitively, the above condition $|C_{t'}| \leq k$ for subtrees t' of cloned subtrees t means that the k^+-increment steps are applied in a *top-down* fashion, i.e., it cannot happen that some subtree t' is cloned and later a subtree t containing t' is also cloned. This fact will be important in Theorem 4.14 to get an appropriate upper bound on the size of any cloned subtree t.

Proof. We consider the inverse operation of k^+-increments, namely, k^+-elimination defined as follows: Let v be a node in a pattern p such that there are $k' > k$ isomorphic subtrees rooted at the child nodes of v. Then, we may delete $k' - k$ of these subtrees.

Below, we will show that there exists a pattern $p^k \in \mathcal{P}_k(\sigma)$ that can be obtained from p by a sequence of k^+-elimination steps in such a way that every subtree t' of an eliminated subtree t satisfies the condition $|C_{t'}| \leq k$. The desired transformation of p^k into p is then obtained by applying the sequence of corresponding k^+-increment steps in inverse order.

The proof is by induction on the number n of "k-violations" in pattern p, i.e., the number of pairs (v, C_t) where v is a node in p and C_t is a set of clones rooted at child nodes of v such that $|C_t| > k$.

For the induction begin, if $n = 0$, then $p \in \mathcal{P}_k(\sigma)$ and we are done. For the induction step, suppose that p has $n > 0$ k-violations. We choose a node v at maximal depth in p, such that there are $k' > k$ isomorphic subtrees rooted at the child nodes of v. By our assumption of maximal depth, we conclude that every subtree t' of any of these isomorphic subtrees satisfies the condition $|C_{t'}| \leq k$. We can transform p into p' by a k^+-elimination step that deletes $k' - k$ of these isomorphic subtrees. The resulting pattern p' has only $n-1$ k-violations. Hence, by the induction hypothesis, p' can be transformed into a pattern $p^k \in \mathcal{P}_k(\sigma)$ via a sequence of k^+-elimination steps such that every subtree t' of an eliminated subtree t satisfies the condition $|C_{t'}| \leq k$. Hence, by first transforming p into p' and then p' into p^k, we get the desired transformation from p into p^k. □

Abstracting chase trees via canonical target instances. It turns out that, to test the condition $chase(I, \sigma) \rightarrow chase(I, \Sigma)$, it suffices to use canonical instances of patterns realized in the chase tree of I with σ. This way one can, e.g., abstract away from undesired multiple occurrences of constants in the facts of I.

Lemma 3.18. *Let σ be a nested tgd, let Σ be a set of nested tgds, and let I be an arbitrary source instance. Moreover, let $\mathcal{P}$ be the set of patterns of the chase forest of I with σ. Then, if for each pattern p in $\mathcal{P}$, $J_p \rightarrow chase(I_p, \Sigma)$ holds, where I_p, J_p are the canonical sources, respectively, target instances, of p, then $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds as well.*

Note that the opposite direction of the above lemma does, in general, not hold, i.e.: It is not guaranteed that $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ implies $J_p \rightarrow chase(I_p, \Sigma)$. However, this is irrelevant, since the only purpose of Lemma 3.18 is to restrict the set of source instances that have to be considered in the implication test of nested tgds: In principle, we would have to test that $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds *for every possible source instance*. Lemma 3.18 tells us that we may restrict ourselves to *canonical source instances*. That is, we get rid of undesired double occurrences of constants. Below, in Lemma 3.22, we will prove a further restriction of the source instances that have to be considered. More precisely, we will show in Lemma 3.22 that we may restrict ourselves to canonical source instances *up to a certain size*. The decidability of the implication problem of nested tgds (see the proof of Theorem 3.1 at the end of this section) will then be an easy consequence.

The proof of Lemma 3.18 uses a notion of homomorphism that affects the *constants of an instance* and preserves function symbols in nulls, in contrast to the usual constant-preserving homomorphism acting on nulls, as defined in Section 2.

Definition 3.19 (C-mapping, c-homomorphism). A *c-mapping* is a function $\theta : \mathsf{CONST} \rightarrow \mathsf{CONST}$ that is naturally extended to Skolem terms with arguments from CONST: For every $f(a_1, \ldots, a_k)$, we define $\theta(f(a_1, \ldots, a_k)) = f(\theta(a_1), \ldots, \theta(a_k))$. For $e \in \mathsf{NULLS}$, we then let $\theta(e) = \mu(\theta(\mu^{-1}(e)))$ or, using the shorthand notation, $\theta(\hat{f}(\vec{a})) = \hat{f}(\theta(\vec{a}))$. The latter extension leads to a function $\theta^{[\mathsf{null}]} : \mathsf{NULLS} \rightarrow \mathsf{NULLS}$ derived from the c-mapping θ. For tuples of constants and nulls, we define $\theta((s_1, \ldots, s_k)) = (\theta(s_1), \ldots, \theta(s_k))$. Finally, a c-mapping θ is called a *c-homomorphism* from J to J' if for every fact $R(\vec{s}) \in J$, there is a fact $R(\theta(\vec{s})) \in J'$. ◁

Example 3.20. There is a c-homomorphism $\theta = [a \mapsto a', b \mapsto b']$ transforming the instance $J = \{R(a, b, \hat{f}(a, b))\}$ into $J' = \{R(a', b', \hat{f}(a', b'))\}$. ◁

Proof of Lemma 3.18. By Corollary 3.15, subinstances of $chase(I, \sigma)$ generated by distinct chase trees have no nulls in common. Therefore, a homomorphism from $chase(I, \sigma)$ to $chase(I, \Sigma)$ exists if and only if for every chase tree T in $\mathcal{F}_{\sigma,I}$, the condition $J \rightarrow chase(I, \Sigma)$ holds, where $J = \bigcup\{ facts(t) \mid t \text{ is a node of } T\}$.

Fix an arbitrary chase tree T in $\mathcal{F}_{\sigma,I}$ with t_0 denoting its root triggering, and let $J = \bigcup\{ facts(t) \mid t$ be a node of $T\}$. We use p to denote the pattern of T, and we denote by T_p the canonical chase tree of p, i.e., T_p is a chase tree where the triggerings assign fresh, pairwise-distinct constants to variables. Furthermore, let I_p be the canonical source instance and let J_p be the canonical target instance. Since the chase tree T is arbitrary, it is sufficient to show that $J_p \rightarrow chase(I_p, \Sigma)$ implies $J \rightarrow chase(I, \Sigma)$. Under the assumption that a homomorphism h from J_p to $chase(I_p, \Sigma)$ exists, we will construct a homomorphism e from J to $chase(I, \Sigma)$.

Since p is a pattern of T, there is a one-to-one correspondence $\theta^{[\mathsf{tr}]}$ between the triggerings in T and those in the canonical chase tree T_p. Since triggerings in T_p assign fresh, pairwise-distinct

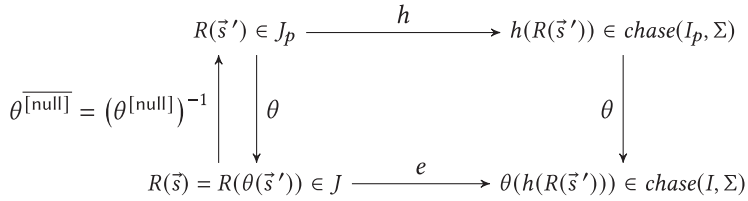

Fig. 6. Mappings used in the proof of Lemma 3.18.

constants to variables, there is a c-homomorphism θ from I_p to I, which, by extending its domain to nulls according to Definition 3.19 is also a c-homomorphism from J_p to J. Specifically, for every triggering $(\sigma_i, \vec{a}_0, \vec{a})$ in T and triggering $(\sigma_i, \vec{a}'_0, \vec{a}')$ in T_p such that $\theta^{[\text{tr}]}((\sigma_i, \vec{a}'_0, \vec{a}')) = (\sigma_i, \vec{a}_0, \vec{a})$, θ maps the elements of $\vec{a}'$ to the respective elements of $\vec{a}$. Consider the mapping $\theta^{[\text{null}]}$ on nulls of J_p, derived from θ as in Definition 3.19. In Definition 3.2, the function *nulls* is defined, which, for a triggering $(\sigma_j, \vec{b}_0, \vec{b})$, retrieves all nulls $\hat{f}_i(\vec{b}_0 \cdot \vec{b})$ instantiating Skolem terms from $vars(\sigma_j)$. Note that $\bigcup\{\, nulls(t) \mid t \text{ a node of } T_p\}$ is exactly the set N_p of nulls of J_p: $N_p = dom(J_p) \cap \text{NULLS}$. Similarly, the set N of nulls of J is $\bigcup\{\, nulls(t) \mid t \text{ a node of } T\}$.

We claim that $\theta^{[\text{null}]}$ is a bijection between the nulls N_p in J_p and the nulls N in J. This can be seen as follows: By Corollary 3.15, we know that any two different triggerings introduce disjoint sets of nulls. Moreover, $\theta^{[\text{tr}]}$ is a bijection between triggerings in T_p and in T. Hence, it suffices to show that, for every single triggering t in T_p, the mapping $\theta^{[\text{null}]}$ is a bijection from $nulls(t)$ to $nulls(\theta^{[\text{tr}]}(t))$. Let $t = (\sigma_i, \vec{a}'_0, \vec{a}')$ and $\theta^{[\text{tr}]}(t) = (\sigma_i, \vec{a}_0, \vec{a})$. Moreover, let $\vec{x}_0$ denote the universally quantified variables of all ancestors of part σ_i and let $\vec{x}$ denote the universally quantified variables of part σ_i. Then the Skolem terms introducing nulls by a triggering of σ_i are all of the form $f_1(\vec{x}_0, \vec{x}), \ldots, f_m(\vec{x}_0, \vec{x})$ for some $m \geq 0$. We thus have $nulls(t) = \{f_1(\vec{a}'_0, \vec{a}'), \ldots, f_m(\vec{a}'_0, \vec{a}')\}$ and $nulls(\theta^{[\text{tr}]}(t)) = \{f_1(\vec{a}_0, \vec{a}), \ldots, f_m(\vec{a}_0, \vec{a})\}$. In other words, the triggering of such a part generates m distinct nulls (corresponding to the m existentially quantified variables in the non-Skolemized form) no matter if in this triggering some of the variables in $(\vec{x}_0, \vec{x})$ are assigned the same constant. Hence, $\theta^{[\text{null}]}$ is indeed a bijection from $nulls(t)$ to $nulls(\theta^{[\text{tr}]}(t))$.

This observation is crucial for our construction, which will be using $\theta^{\overline{[\text{null}]}}$, the inverse of $\theta^{[\text{null}]}$, that is $\theta^{\overline{[\text{null}]}} = (\theta^{[\text{null}]})^{-1} : N \to N_p$. Specifically, our desired homomorphism e from J to $chase(I, \Sigma)$ will be defined as the composition of $\theta^{\overline{[\text{null}]}}$ with h and $\theta^{[\text{null}]}$, where h is the homomorphism from J_p to $chase(I_p, \Sigma)$ from the assumption of the lemma:

$$e(s) = \begin{cases} s, & \text{if } s \text{ is a constant} \\ \theta^{[\text{null}]}(h(\theta^{\overline{[\text{null}]}}(s))), & \text{if } s \text{ is a null.} \end{cases}$$

We need to show that this mapping e satisfies the definition of homomorphism: For an arbitrary fact $R(\vec{s}) \in J$, we prove that $R(e(\vec{s})) \in chase(I, \Sigma)$ holds.

By the definition of canonical chase tree and bijectivity of $\theta^{[\text{tr}]}$, θ is a surjective mapping. Thus, for each fact $R(\vec{s}) \in J$, there exists a fact $R(\vec{s}\,')$ in J_p such that $\theta(\vec{s}\,') = \vec{s}$. The existence of the homomorphism h from J_p to $chase(I_p, \Sigma)$ implies that for each $R(\vec{s}\,')$ there is a fact $h(R(\vec{s}\,')) \in chase(I_p, \Sigma)$. Consider the chase forest $\mathcal{F}_{\Sigma, I_p}$ of I_p with Σ and let $(\tau_j, \vec{b}_0, \vec{b})$ be the triggering of some part τ_j in a nested tgd of Σ that has produced the atom $h(R(\vec{s}\,'))$. Since θ is a c-homomorphism from I_p to I, there also exists a triggering $(\tau_j, \theta(\vec{b}_0), \theta(\vec{b}))$ in the chase forest $\mathcal{F}_{\Sigma, I}$ of I producing the fact $\theta(h(R(\vec{s}\,'))) \in chase(I, \Sigma)$. These facts and mappings are presented in Figure 6.

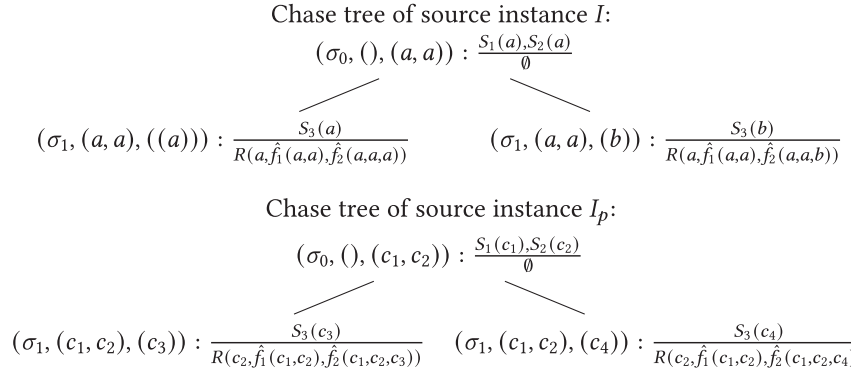

Fig. 7. Chase tree of I in Example 3.21 (upper part) and its canonical chase tree of I_p (lower part).

We have shown that for an arbitrary fact $R(\vec{s}\,') \in J_p$ such that $\theta(\vec{s}\,') = \vec{s}$, there exists a fact $\theta(h(R(\vec{s}\,')))$ in $chase(I, \Sigma)$. It remains to show that this fact coincides with $R(e(\vec{s}))$. Recall that e preserves constants and maps each null s_i to $\theta^{[\mathsf{null}]}(h(\theta^{\overline{[\mathsf{null}]}}(s_i)))$. For each term position $i \leq |\vec{s}\,'|$ and for the ith element s'_i of $\vec{s}\,'$, consider two cases:

Case 1: s'_i is a null. Then, also the corresponding element s_i of $\vec{s} \in dom(J)$ is a null. Then, $\theta(h(s'_i)) = \theta(h(\theta^{\overline{[\mathsf{null}]}}(s_i))) = \theta^{[\mathsf{null}]}(h(\theta^{\overline{[\mathsf{null}]}}(s_i)))$ holds, and thus $\theta(h(s'_i)) = e(s_i)$.

Case 2: s'_i is a constant. By the choice of $\vec{s}\,'$, the equality $\theta(s'_i) = s_i$ holds. Note also that h preserves constants, so $\theta(h(s'_i)) = \theta(s'_i) = s_i = e(s_i)$.

We have demonstrated that, for an arbitrary fact $R(\vec{s}) \in J$, there exists a fact $R(e(\vec{s}))$ in $chase(I, \Sigma)$ and thus e is a homomorphism from J to $chase(I, \Sigma)$. □

Example 3.21. Consider the nested tgds σ and σ':

$$\sigma : \forall x_1 \forall x_2 (S_1(x_1) \wedge S_2(x_2) \rightarrow \exists y (\forall x_3 (S_3(x_3) \rightarrow \exists z (R(x_2, y, z))))),$$
$$\sigma' : \forall x_1 (S_1(x_1) \rightarrow \exists y (\forall x_2 \forall x_3 (S_2(x_2) \wedge S_3(x_3) \rightarrow \exists z (R(x_2, y, z))))),$$

with the Skolemized forms given below:

$$\sigma : \forall x_1 \forall x_2 (S_1(x_1) \wedge S_2(x_2) \rightarrow \forall x_3 (S_3(x_3) \rightarrow R(x_2, f_1(x_1, x_2), f_2(x_1, x_2, x_3)))),$$
$$\sigma' : \forall x_1 (S_1(x_1) \rightarrow (\forall x_2 \forall x_3 (S_2(x_2) \wedge S_3(x_3) \rightarrow R(x_2, g_1(x_1), g_2(x_1, x_2, x_3))))).$$

Suppose that we want to test $\Sigma \models \sigma$ with $\Sigma = \{\sigma'\}$ and consider the source instance $I = \{S_1(a), S_1(b), S_2(a), S_2(b), S_3(a), S_3(b)\}$. Figure 7 shows a chase tree T of source instance I with nested tgds σ and the corresponding canonical chase tree T_p. In these trees, apart from the triggerings, we also display the relevant source atoms enabling each triggering (above the bar) and the target atoms produced by each triggering (below the bar). In case of T_p, these are the canonical source and target instances, respectively. In particular, we have

$I_p = \{S_1(c_1), S_2(c_2), S_3(c_3), S_3(c_4)\}$,
$J_p = \{R(c_2, \hat{f}_1(c_1, c_2), \hat{f}_2(c_1, c_2, c_3)), R(c_2, \hat{f}_1(c_1, c_2), \hat{f}_2(c_1, c_2, c_4))\}$, and
$J = \{R(a, \hat{f}_1(a, a), \hat{f}_2(a, a, a)), R(a, \hat{f}_1(a, a), \hat{f}_2(a, a, b))\}$.

Moreover, we have $chase(I_p, \Sigma) = \{R(c_2, \hat{g}_1(c_1), \hat{g}_2(c_1, c_2, c_3)), R(c_2, \hat{g}_1(c_1), \hat{g}_2(c_1, c_2, c_4))\}$. Hence, there exists a homomorphism $h : J_p \rightarrow chase(I_p, \Sigma)$ with $h = [\hat{f}_1(c_1, c_2) \mapsto \hat{g}_1(c_1), \hat{f}_2(c_1, c_2, c_3) \mapsto \hat{g}_2(c_1, c_2, c_3), \hat{f}_2(c_1, c_2, c_4) \mapsto \hat{g}_2(c_1, c_2, c_4)]$.

The bijection $\theta^{[\text{tr}]}$ between the three triggerings in T_p and the three triggerings in T is clear. The c-homomorphism θ according to the proof of Lemma 3.18 is $\theta = [c_1 \mapsto a, c_2 \mapsto a, c_3 \mapsto a, c_4 \mapsto b]$ and thus $\theta^{[\text{null}]} = [\hat{f}_1(c_1, c_2) \mapsto \hat{f}_1(a, a), \hat{f}_2(c_1, c_2, c_3) \mapsto \hat{f}_2(a, a, a), \hat{f}_2(c_1, c_2, c_4) \mapsto \hat{f}_2(a, a, b)]$. From this, we define the mapping $e : dom(J) \to dom(chase(I, \Sigma))$ as in the proof of Lemma 3.18:

$$e(s) = \begin{cases} s, & \text{if } s \text{ is a constant} \\ \theta^{[\text{null}]}(h(\theta^{\overline{[\text{null}]}}(\hat{f}_1(a, a)))) = \theta^{[\text{null}]}(h(\hat{f}_1(c_1, c_2)))) = \hat{g}_1(a), & \text{if } s = \hat{f}_1(a, a). \\ \theta^{[\text{null}]}(h(\theta^{\overline{[\text{null}]}}(\hat{f}_2(a, a, a)))) = \hat{g}_2(a, a, a), & \text{if } s = \hat{f}_2(a, a, a). \\ \theta^{[\text{null}]}(h(\theta^{\overline{[\text{null}]}}(\hat{f}_2(a, a, b)))) = \hat{g}_2(a, a, b), & \text{if } s = \hat{f}_2(a, a, b). \end{cases}$$

Moreover, $chase(I, \Sigma)$ with $\Sigma = \{\sigma'\}$ contains eight atoms obtained by instantiating any of the variables x_1, x_2, x_3 in σ' to either a or b. In particular, $\{R(a, \hat{g}_1(a), \hat{g}_2(a, a, a)), R(a, \hat{g}_1(a), \hat{g}_2(a, a, b))\} \subseteq chase(I, \Sigma)$ holds. Hence, e is indeed a homomorphism from J to $chase(I, \Sigma)$. ◁

Lemma 3.18 reduced checking $chase(I, \sigma) \to chase(I, \Sigma)$ to a series of tests of $J_p \to chase(I_p, \Sigma)$, for the canonical source and target instances I_p, respectively, J_p, of each pattern p, realized in the chase forest of I with σ. The next Lemma 3.22 strengthens this result by showing that only *patterns up to a certain size* (which is independent of a concrete source instance I) need to be considered.

Lemma 3.22. *Consider the following setting:*

- *A set Σ of nested tgds and a nested tgd σ.*
- *Integer $k > v_\sigma w_\Sigma$, where v_σ is the number of existentially quantified variables (i.e., distinct Skolem functions) in σ and w_Σ is the maximum number of universally quantified variables in any nested tgd in Σ.*
- *Patterns p^k, p of σ where p^k is a k-pattern and $p^k \leq_k p$ holds.*
- *Pairs (I_{p^k}, J_{p^k}) and (I_p, J_p) of the canonical source and target instances of the patterns p^k and p, respectively. We assume $I_{p^k} \subseteq I_p$ and $J_{p^k} \subseteq J_p$, that is, the same constants as in (I_{p^k}, J_{p^k}) are used to instantiate the subtree p^k of p in (I_p, J_p).*

Then, every homomorphism h^k from J_{p^k} to $chase(I_{p^k}, \Sigma)$ can be extended to a homomorphism h from J_p to $chase(I_p, \Sigma)$.

The proof of Lemma 3.22 will require reasoning about parts of canonical instances corresponding to clone subtrees in patterns. To this end, we first formulate an easy lemma that establishes the relationship between such subinstances.

Definition 3.23. Let J be an instance resulting from a chase with nested tgds. For a subinstance K of J, we define the set $lcst(K)$ of *local constants of K* as $cst(K) \setminus cst(J \setminus K)$. That is, local to K are constants occurring in K and nowhere else in J, as non-functional terms (i.e., constants) or as arguments of nulls. ◁

Lemma 3.24. *Let p be a pattern of a chase tree T with a nested tgd σ and let J_p be its canonical target instance; let v_1, v_2 be child nodes of some node v of p such that the subtrees t_1, t_2 of p rooted at nodes v_1, v_2 are isomorphic; finally, let K_1, K_2 denote the subinstances of J_p such that K_i consists of the facts generated by any of the triggerings in the subtree t_i with $i \in \{1, 2\}$. Then K_1 and K_2 are equal up to a c-isomorphism (i.e., a bijection that is a c-homomorphism in both directions) that maps constants from $lcst(K_1)$ to $lcst(K_2)$ and leaves all other constants unchanged.*

Proof. The proof is straightforward. □

With Lemma 3.24, we can now prove Lemma 3.22 about extensions of homomorphisms to canonical instances resulting from k^+-increments.

Proof of Lemma 3.22. The proof is by induction on the length i of the sequence of k^+-increments of p^k transforming it to p. For the base case $i = 0$, the existence of h is an assumption of the lemma. Take as induction hypothesis that for some $i \geq 0$, a homomorphism h_i from J_{p_i} to $chase(I_{p_i}, \Sigma)$ exists, s.t. h_i extends h^k. We show that h_i can be further extended to a homomorphism h_{i+1} from $J_{p_{i+1}}$ to $chase(I_{p_{i+1}}, \Sigma)$, where p_{i+1} is obtained from p_i by a single k^+-increment.

By the definition of k^+-increment, p_{i+1} is obtained by inserting a copy of some node v in p_i—including the full subtree π_v rooted at v—to the parent node of v. Moreover, v is one of ℓ nodes $v_1, \ldots, v_\ell$ with $\ell \geq k$ having the same parent in p_i and being roots to subtrees each of which is isomorphic to π_v. In the canonical chase tree T_{p_i} of p_i this corresponds to a parent triggering t_ρ of the part $label(parent(v))$, with $\ell \geq k$ isomorphic subtrees rooted at triggerings $t_1 = (\sigma_i, \vec{a}_0, \vec{a}_1), \ldots, t_\ell = (\sigma_i, \vec{a}_0, \vec{a}_\ell)$ of the same part σ_i with the same input assignment $\vec{a}_0$.

The canonical chase tree $T_{p_{i+1}}$ extends T_{p_i} with yet another subtree rooted at a triggering $t_{\ell+1} = (\sigma_i, \vec{a}_0, \vec{a}_{\ell+1})$ placed directly under t_ρ. For $j \in \{1, \ldots, \ell+1\}$, we define $K_j \subseteq J_{p_{i+1}}$ as $K_j = \bigcup\{ facts(t) \mid t \in desc(t_j)\}$, i.e., $K_j \subseteq J_{p_i}$ for each $j \in \{1, \ldots, \ell\}$ and $K_{\ell+1} = J_{p_{i+1}} \setminus J_{p_i}$.

Let F denote the set of nulls shared between any two distinct instances in the set $\{K_1, \ldots, K_{\ell+1}\}$. By Lemma 3.14, each such shared null has the form $\hat{f}(\vec{b})$ where $\vec{b}$ is a prefix of the common input assignment $\vec{a}_0$ of the triggerings $t_1, \ldots, t_{\ell+1}$. Thus, the number of nulls in F is bounded: $|F| \leq v_\sigma$. This observation pinpoints the key idea of the proof, formulated as the following claim:

Claim A. *There exists at least one $\tilde{K} \in \{K_1, \ldots, K_\ell\}$ such that for every null $\hat{f}(\vec{a}) \in F$, with $h_i(\hat{f}(\vec{a})) = \hat{g}(\vec{b})$, $\vec{b}$ does not contain a local constant of $\tilde{K}$.* This claim is proved as follows: As explained above, there are at most v_σ nulls in F, and each term in an instance obtained by the chase with Σ is built using at most w_Σ constants, namely: either a single constant or up to w_Σ constants serving as arguments of a null. Therefore, at most w_Σ instances among $\{K_1, \ldots, K_\ell\}$ can contribute their local constants to the term $h_i(\hat{f}(\vec{a}))$, and thus local constants from at most $v_\sigma w_\Sigma$ instances can occur in $h_i(F)$. However, by the assumption of the lemma, $k > v_\sigma w_\Sigma$. Hence, by $\ell \geq k$, a subinstance $\tilde{K} \in \{K_1, \ldots, K_\ell\}$ with the desired property exists. ◁

We now consider a subinstance $\tilde{K} \in \{K_1, \ldots, K_\ell\}$ from the claim above and the subinstance $K' = K_{\ell+1}$, distinguishing $J_{p_{i+1}}$ from J_{p_i}. Our task is to extend the homomorphism h_i to the nulls of K', which are not present in the domain of J_{p_i}, to obtain a homomorphism h_{i+1} from $J_{p_{i+1}}$ to $chase(I_{p_{i+1}}, \Sigma)$. $\tilde{K}$ and K' represent two isomorphic sibling subtrees of the pattern p_{i+1} in its canonical target instance. By Lemma 3.24 a renaming of local constants of $\tilde{K}$ into the local constants of K' exists, transforming $\tilde{K}$ into K'. We let θ be such a renaming extended with the identity mapping on all other constants in J_{p_i}. That is, θ is the identity on $cst(J_{p_i}) \setminus lcst(\tilde{K})$. By definition θ is a bijection between $cst(\tilde{K})$ and $cst(K')$. Consider the function h'_{i+1} defined on $dom(K')$ as $h'_{i+1}(x) = \theta(h_i(\theta^{-1}(x)))$. Mapping h'_{i+1} can be used to extend h_i in the following sense: For every $x \in dom(K') \cap dom(J_{p_i})$, we have $h'_{i+1}(x) = h_i(x)$. Indeed, θ^{-1} preserves the non-local constants of K' and by Claim A, h_i maps each null containing no local constants of $\tilde{K}$ to a term also using no local constants of $\tilde{K}$: Thus, by construction of θ for every null $\hat{f}(\vec{a})$ in the domain of h_i, we have $\theta(h_i(\theta^{-1}(\hat{f}(\vec{a})))) = h_i(\hat{f}(\vec{a}))$. Moreover, for a constant a, we have $\theta(h_i(\theta^{-1}(a))) = \theta(\theta^{-1}(a)) = a$.

We now take $h_{i+1} = h_i \cup h'_{i+1}$ and show that h_{i+1} is a homomorphism. First note that θ is a c-homomorphism from I_{p_i} to $I_{p_{i+1}}$. Extended to nulls as in Definition 3.19, θ is also a c-homomorphism from J_{p_i} to $J_{p_{i+1}}$ and from $chase(I_{p_i}, \Sigma)$ to $chase(I_{p_{i+1}}, \Sigma)$, analogously to the proof of Lemma 3.18. In particular, $\theta(h_i(\tilde{K})) \subseteq chase(I_{p_{i+1}}, \Sigma)$ and, since $\tilde{K} = \theta^{-1}(K')$, we have $\theta(h_i(\theta^{-1}(K'))) \subseteq chase(I_{p_{i+1}}, \Sigma)$. Note that h_i preserves constants, and therefore $h'_{i+1} = \theta^{-1} \circ h_i \circ \theta$ is a constant-preserving homomorphism from K' to $chase(I_{p_{i+1}}, \Sigma)$. Since $h_{i+1} = h_i \cup h'_{i+1}$, we have

just shown that h_{i+1} is a homomorphism from K' to $chase(I_{p_{i+1}}, \Sigma)$ and with that, a homomorphism from $J_{p_{i+1}}$ to $chase(I_{p_{i+1}}, \Sigma)$, as desired. □

We now have all properties required to prove Theorem 3.1:

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

Proof. The decision procedure Implies relies on the usual characterization of the implication in the finite based on the chase, namely, $\Sigma \models \sigma$ if and only if $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ holds for every source instance I. This characterization is applicable to mappings that are closed under target homomorphisms, which is the case for SO tgds [11], and hence, in particular for nested tgds. Indeed, assume $\Sigma \models \sigma$. That is, for each pair $(I, J) \models \Sigma$, also $(I, J) \models \sigma$ is the case. In other terms, each solution J for I under Σ is also a solution for I under σ and $chase(I, \sigma) \rightarrow J$ must hold. In particular, this holds for $J = chase(I, \Sigma)$, and therefore $chase(I, \sigma) \rightarrow chase(I, \Sigma)$ is the case. In the opposite direction, assume that for all source instances I, a homomorphism from $chase(I, \sigma)$ to $chase(I, \Sigma)$ exists. Since $chase(I, \Sigma)$ is a universal solution under Σ, for arbitrary J such that $(I, J) \models \Sigma$, $chase(I, \Sigma) \rightarrow J$ and thus $chase(I, \sigma) \rightarrow J$ holds. Since σ is closed under target homomorphisms, an arbitrary solution J for I under Σ is also a solution for I under σ, for arbitrary I. This concludes the proof of the characterization of implication in the finite via the chase.

The completeness of the decision procedure is immediate: $\Sigma \not\models \sigma$ is only returned if for some k-pattern p^k with the canonical source, respectively, target instances I_{p^k}, J_{p^k}, no homomorphism from J_{p^k} to $chase(I_{p^k}, \Sigma)$ exists. Note that J_{p^k} is a subinstance of $chase(I_{p^k}, \sigma)$, and thus $chase(I_{p^k}, \sigma) \not\rightarrow chase(I_{p^k}, \Sigma)$ holds as well. This makes I_{p^k} a witness for the failure of the implication.

It remains to show the soundness. Assume that for some nested tgd σ and a set Σ of nested tgds the procedure returns *true*. Consider an arbitrary source instance I corresponding to the target instance $J_\sigma = chase(I, \sigma)$. We show that a homomorphism from J_σ to J_Σ exists, where $J_\Sigma = chase(I, \Sigma)$. By Lemma 3.18, it suffices to show that for each pattern p in the chase forest of I with σ, a homomorphism e from J_p to $chase(I_p, \Sigma)$ exists, where I_p, J_p are the canonical source and target instances of p, respectively.

Let p^k be a k-pattern of σ such that $p^k \leq_k p$ holds. By Lemma 3.17, such k-pattern p^k exists. Since the procedure Implies returned *true*, it follows that a homomorphism h^k from J_{p^k} to $chase(I_{p^k}, \Sigma)$ exists, where I_{p^k} and J_{p^k} are, respectively, the canonical source and target instances of p^k.

We now apply Lemma 3.22 to prove that h^k can be extended to a homomorphism h from J_p to $chase(I_p, \Sigma)$, where I_p, J_p are, respectively, the canonical source and target instances of p. Since p is arbitrary, by Lemma 3.18 this implies that a homomorphism from $chase(I, \sigma)$ to $chase(I, \Sigma)$ exists for every source instance I and, hence, $\Sigma \models \sigma$. □

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

Corollary 3.25. *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 Reference [12], which builds on Reference [3]. As a matter of fact, an examination of the proof of Theorem 1 in Reference [12] 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 3.25 contributes 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 GLAV mappings contains the class of GLAV mappings and is, in turn, contained in the class of (mappings specified by) 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 GLAV mappings from GLAV mappings, and also plain SO tgds from nested GLAV mappings. The main result of this section is an algorithm for telling whether or not a given nested GLAV mapping is logically equivalent to a GLAV mapping. In addition, we give useful sufficient conditions for showing that a plain SO tgd is not logically equivalent to any nested GLAV mapping. Moreover, these conditions are easy to apply in concrete instances, in the sense that they give the user tools based on simple and natural concepts.

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 $chase(I, \mathcal{M})$ 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 $core(chase(I, \mathcal{M}))$ as *the* core of the universal solutions for I w.r.t. $\mathcal{M}$ [10]. Note that, in general, $core(chase(I, \mathcal{M}))$ need not be a universal solution for I w.r.t. $\mathcal{M}$ [9]. However, if $\mathcal{M}$ is specified by a plain SO tgd, then $core(chase(I, \mathcal{M}))$ is a universal solution for I w.r.t. $\mathcal{M}$. The reason for this is that, as shown in Reference [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, $core(chase(I, \mathcal{M}))$ 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 Reference [9].

Definition 4.1. A schema mapping $\mathcal{M}$ specified by an SO tgd has *bounded f-block size* if there is an integer b such that for every source instance I, the f-block size of $core(chase(I, \mathcal{M}))$ is at most b, the smallest such b is then called *the f-block size of* $\mathcal{M}$; otherwise, we say that $\mathcal{M}$ has *unbounded f-block size*. ◁

The next result follows immediately from Proposition 3.14 and Theorem 4.10 in Reference [9]. Recall that every nested GLAV mapping is equivalent to some plain SO tgd.

Theorem 4.2 ([9]). *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.*

Proof. Theorem 4.10 [9] states: *A schema mapping $\mathcal{M}$ specified by an SO tgd is CQ-equivalent to a GLAV schema mapping if and only if $\mathcal{M}$ has bounded f-block size.* This is almost the statement we need, except that it talks about CQ-equivalence instead of logical equivalence. However, as has been detailed on page four in Section 1, by combining results from References [2] and [9], we may conclude that CQ-equivalence and logical equivalence coincide for plain SO tgds. □

The preceding Theorem 4.2 will be used to prove the main result in this section, which we now state formally.

Theorem 4.3. *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 4.2, 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 4.4. Let C be a class of schema mappings. We say that C has *effective threshold for f-block size* if there exists a recursive function $f : C \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, s.t. every mapping $\mathcal{M} \in C$ either has f-block size at most $f(\mathcal{M})$ or has unbounded f-block size. ◁

Before showing that nested GLAV mappings actually have effective bounded threshold for f-block size, let us give the following lemma and introduce some useful notation:

Lemma 4.5. *For every nested GLAV mapping specified by a set Σ of nested tgds, there exists a single nested tgd σ, such that Σ and σ have the same f-block size. Moreover, for every source instance I for Σ, there is a source instance I' for σ with $|I'| = |I| + 1$, such that $chase(I, \Sigma) = chase(I', \sigma)$.*

Proof. Let $\Sigma = \{\sigma_i \mid 1 \leq i \leq k\}$ be a set of nested tgds. Let R denote a relation symbol not occurring in Σ and let x be a variable not occurring in Σ. Then σ is given by

$$R(x) \to (\sigma_1 \wedge \ldots \wedge \sigma_k).$$

We now claim that Σ and σ have the same f-block size. Indeed, since R is a fresh relation symbol, the source instance I for Σ witnessing some f-block size can be extended to $I' = I \cup \{R(a)\}$ for σ where a is an arbitrary constant. Thus, we know that σ has at least the f-block size of Σ. Conversely, σ has at most the f-block size of Σ, since the variable x does not occur anywhere in Σ, thus not allowing f-blocks to be connected through a shared variable. □

In Section 3, we introduced several crucial notions for the analysis of nested tgds, such as chase forests, chase trees, patterns, isomorphic subtrees in a pattern, k-patterns, and so on. To prove that the class of nested GLAV mappings has effective threshold for f-block size, we have to relativize these notions to a subinstance of the chase result.

Definition 4.6 (Chase tree with restricted conclusions). Let σ be a nested tgd and let I be a source instance. Let $\mathcal{F}_{\sigma,I}$ be the chase forest of I with σ and let $T_{\sigma,I} = (V, T)$ be a chase tree in this chase forest. Moreover, let $J \subseteq chase(I, \sigma)$. Recall from Definition 3.3 that each vertex in $T_{\sigma,I}$ corresponds to a triggering $(\sigma_i, \vec{a}_0, \vec{a})$ and that σ_i is a Skolemized formula of the form $\forall\vec{x}(\varphi(\vec{x}_0, \vec{x}) \to \alpha_1(\vec{x}_0, \vec{x}) \wedge \cdots \wedge \alpha_n(\vec{x}_0, \vec{x}))$.

The *chase tree with conclusions restricted to J* (or, simply, the *RC-chase tree w.r.t. J*), which we denote as $T_{\sigma,I}(J)$, is obtained from $T_{\sigma,I}$ by leaving the tree structure unchanged and extending the triples $(\sigma_i, \vec{a}_0, \vec{a})$ in the vertices with the fourth element C, defined as the following set of conclusion atoms of the part σ_i: $C = \{\alpha_i(\vec{x}_0, \vec{x}) \mid \mu(\alpha_i(\vec{a}_0, \vec{a})) \in J\}$, where $\mu(\alpha_i(\vec{a}_0, \vec{a}))$ denotes the fact obtained from the atom $\alpha_i(\vec{a}_0, \vec{a})$ by replacing every Skolem term $f(\vec{a}_0, \vec{a})$ in it by the null $\mu(f(\vec{a}_0, \vec{a})) = \hat{f}(\vec{a}_0, \vec{a})$. ◁

In other words, the additional label C of each vertex in an RC-chase tree adds to the chase tree the information as to which target atoms are ultimately retained in the subset J of $chase(I, \sigma)$. We now also extend the notion of patterns from Definition 3.5 to allow for restrictions of the conclusions.

Definition 4.7 (Patterns with restricted conclusions). Let σ be a nested tgd and let p be a pattern of σ. Recall that a pattern is a rooted tree where each vertex is labeled by some part σ_i of σ. A *pattern with restricted conclusions* (or, simply, an *RC-pattern*) obtained from p has the same tree structure as p, and the label σ_i of each vertex is extended to the label (σ_i, C), where C is an arbitrary subset of the conclusion atoms $\{\alpha_1(\vec{x}_0, \vec{x}), \ldots, \alpha_n(\vec{x}_0, \vec{x})\}$ of σ_i.

Let I be an arbitrary source instance and let $J \subseteq chase(I, \sigma)$ such that all atoms in J are generated by a single chase tree $T_{\sigma,I}$. Then the *RC-pattern w.r.t. J* is the RC-pattern obtained from $T_{\sigma,I}(J)$ by leaving the tree structure unchanged and restricting every label $(\sigma_i, \vec{a}_0, \vec{a}, C)$ to (σ_i, C). ◁

Analogously to Definition 3.6, we can consider the notions of (isomorphic) subtrees of an RC-pattern and clones of subtrees, which lead to the definition of RC-k-patterns.

Definition 4.8 (RC-k-patterns). Let σ be a nested tgd and let p be an RC-pattern of σ. Let (V, T) denote the tree structure of p, and let λ denote the vertex labeling of p. As in Definition 3.6, we mean by a "subtree" a complete subtree with all its descendant nodes. Now let v be some vertex in V and let $t_1 = (V_1, T_1)$ and $t_2 = (V_2, T_2)$ denote two subtrees of (V, T) rooted at two different child nodes of v. We call t_1 and t_2 *isomorphic* if they have the same tree structure and the same labeling, i.e.: there exists an isomorphism h from (V_1, T_1) to (V_2, T_2) such that $\lambda(v) = \lambda(h(v))$ holds for every $v \in V_1$. If this is the case, we also say that t_2 is an *RC-clone* of t_1.

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

Recall from Proposition 3.8 that for a positive integer k and a nested tgd σ, the set of k-patterns of σ is finite. The proof was by structural induction on σ making use of the observation, that the sets of k-patterns of immediate subparts $\sigma'_1, \ldots, \sigma'_n$ of some part σ_i in σ can only be combined in finitely (but of course exponentially) many ways into k-patterns of σ_i. For RC-patterns, we get an additional exponentiality, since, in addition to the parts of σ, we also have to take subsets of the conclusions of each part into account. However, this does not affect the finiteness of RC-$\mathcal{P}_k$. We thus get the following result:

Corollary 4.9. *For a positive integer k and a nested tgd σ, the set RC-$\mathcal{P}_k$ of RC-k-patterns of σ is finite.*

The definition of the canonical source instance I_p of a pattern (see Definition 3.9) is not affected by the restriction of the conclusions in an RC-pattern p. The same applies to the canonical chase tree (canonical RC-chase tree), with the provision that the triggerings at its vertices are now specified by quadruples $t = (\sigma_i, \vec{a}_0, \vec{a}, C)$ rather than triples $(\sigma_i, \vec{a}_0, \vec{a})$. The function $facts(t)$ applied to such a quadruple t returns the instantiation of atoms in C by $\vec{a}_0$ and $\vec{a}$. The canonical target instance J_p of the RC-pattern p with the canonical chase tree (V, T) is the union $\bigcup_{(\sigma_i, \vec{a}_0, \vec{a}, C) \in V} facts((\sigma_i, \vec{a}_0, \vec{a}, C))$.

Let T be a subtree of an RC-pattern and $\tau = (V', T')$ be the respective subtree of the canonical chase tree. By the *set of facts generated by T (respectively, by τ)*, we mean the set $K = \bigcup\{facts(t) \mid t \in V'\}$, that is, the set of facts generated by triggerings in τ, restricted to the target atoms in the sets C at each vertex in τ. The set of *local constants* of T (respectively, *of τ*) is

$$lcst(T) = lcst(\tau) = \{a \mid \text{ there exists } \vec{a} \text{ such that } a \text{ occurs in } \vec{a} \text{ and } (\sigma_i, \vec{a}_0, \vec{a}, C) \in V'\},$$

that is, the set of constants occurring in the assignments of the triggerings in τ. Regarding K as a subset of the canonical target instance J_p of the RC-pattern p, observe that $lcst(T) \cap cst(K)$ (the local constants of T actually occurring in K) are exactly the local constants of K in the sense of Definition 3.23, namely, those constants occurring in K and nowhere else in J_p. We say that a fact A *contains a local constant from τ* if at least one constant of $lcst(\tau)$ occurs in A: $cst(\{A\}) \cap lcst(\tau) \neq \emptyset$. Finally, for every such RC-pattern subtree T, with the corresponding subtree τ of the canonical chase tree and the set K of facts generated by T, we fix an *order of local constants*, which allows us to speak of *the tuple $\vec{b}$ of local constants of τ* (respectively, *of K*, respectively, *of T*). Note that, for each triggering $(\sigma_i, \vec{a}_0, \vec{a})$, the tuple notation $\vec{a}$ assumes an order of the constants in $\vec{a}$ anyway. This

order of local constants of individual triggerings can be extended to an order of the local constants of T by fixing the order of the nodes in T (e.g., assume top-down, left-to-right order).

Now consider a pattern p and let K_1, K_2 be sets of facts generated by isomorphic subtrees of a pattern such that the roots of these subtrees are siblings in p. Moreover, let the respective tuples of local constants be $\vec{b}_{K_1}$ and $\vec{b}_{K_2}$. Then, the c-homomorphism θ mapping the ith constant in $\vec{b}_{K_1}$ to the ith constant in $\vec{b}_{K_2}$ for every i clearly defines a c-isomorphism between K_1 and K_2. It is convenient to denote this c-isomorphism by $\theta(\vec{b}_{K_1}) = \vec{b}_{K_2}$.

Theorem 4.10. *The class of nested GLAV mappings has effective threshold for f-block size.*

Proof. By Lemma 4.5, it suffices to consider mappings defined by a single nested tgd σ. We have to show that there exists a recursive function g that, for a given nested tgd σ, returns an integer $g(\sigma)$ witnessing bounded f-block size for σ. The proof proceeds in several steps and is split into three claims. We give the overall picture of the proof here. Detailed proofs of the claims are given in the Appendix.

Given an arbitrary nested tgd σ, let n denote the maximum size of all RC-1-patterns of σ. Here, we define the size of an RC-pattern p as the sum of the cardinalities of the sets C of conclusion atoms in the labels of the vertices in p. By the finiteness of RC-$\mathcal{P}_k$ according to Corollary 4.9, this maximum exists. We set the threshold $g(\sigma)$ to $g(\sigma) = n + 1$. Now suppose that there exists a source instance I, s.t. $core(chase(I, \sigma))$ has f-block size $m \geq g(\sigma)$. We will show that then there exists a source instance I', s.t. $core(chase(I', \sigma))$ has f-block size $\geq m + 1$. Since our construction of I' from I can be iterated arbitrarily often, it follows immediately that σ has unbounded f-block size.

Considering canonical source instances only. By Corollary 3.15, we know that the facts in some f-block B of $core(chase(I, \sigma))$ must be generated by a single chase tree in the chase forest $\mathcal{F}_{\sigma,I}$. Now suppose that f-block size m of $core(chase(I, \sigma))$ is realized by some f-block B of $core(chase(I, \sigma))$ and that the facts in B are generated by RC-chase tree T_p with RC-pattern p. Let I_p be the canonical source instance of p. Then the following property holds:

Claim 1. The f-block size of $core(chase(I_p, \sigma))$ is at least m.

The proof idea of this claim is as follows: Recall from the proof of Lemma 3.18 that there is a one-to-one correspondence between the triggerings in (and the facts generated by) the chase tree $T_{\sigma,I}$ and the corresponding pattern p equipped with the variable bindings according to the canonical source instance I_p. We can thus define the set B_p of facts in the canonical target instance corresponding to the set B in $chase(I, \sigma)$. It can be shown that B_p or a superset thereof must also be an f-block in the core of the canonical target instance. In the remainder of the proof, we may thus assume w.l.o.g. that I is the canonical source instance of p.

Isomorphic subtrees in T_p with local constants in facts of B. By our definition of the size of an RC-pattern p as the total number of conclusion atoms in the labels of p and by Corollary 3.15 recalled above, the size of p yields an upper bound on the cardinality of an f-block whose facts are generated by chase tree T_p with pattern p. We are assuming that f-block B has cardinality greater than the maximum size of all 1-patterns of σ. Hence, there exist (at least) two siblings in the pattern p that are the roots of isomorphic subtrees T_1 and T_2, such that both T_1 and T_2 generate some facts of B. We strengthen this observation to the following claim:

Claim 2. For each $i \in \{1, 2\}$, there exists an atom A_i generated by T_i such that $A_i \in B$ and A_i contains a local constant of T_i. (Recall that in a canonical target instance, local constants do not occur outside a subinstance generated by a subtree of a chase tree.)

By the considerations on the size of RC-1-patterns and of B, it is clear that p must be a k-pattern with $k \geq 2$, i.e., there must exist isomorphic subtrees T_1 and T_2 whose roots are siblings and which generate some facts of B. For the proof of Claim 2, it only remains to show that it cannot happen that the facts generated by T_1 and T_2 contain no local constants. Indeed, this can be excluded by showing that otherwise there exists an endomorphism on $core(chase(I, \sigma))$ that allows us to "shrink" B by mapping all facts generated by T_1 to the facts generated by T_2 or vice versa. But this contradicts the assumption that B is an f-block in $core(chase(I, \sigma))$.

Construct another isomorphic subtree. Let $\vec{b}$ and $\vec{c}$ denote the tuples of the local constants of T_1 and of T_2, respectively. By the above considerations, there exists a one-to-one correspondence between these vectors of local constants. We now introduce a vector of fresh constants $\vec{d}$ such that we again have a one-to-one correspondence between this vector and $\vec{b}$. Let θ realize this one-to-one correspondence, i.e.: $\theta(b_i) = d_i$ for every b_i in $\vec{b}$ and d_i in $\vec{d}$. We extend I to $I' = I \cup \theta(I)$. Hence, the chase tree of I' with σ is obtained from the chase tree of I by adding another isomorphic subtree T_3 to the RC-pattern p and taking $\vec{d}$ as the local constants in the variable assignments in T_3. For the f-block size of $core(chase(I', \sigma))$ the following property can be shown:

Claim 3. The instance $core(chase(I', \sigma))$ has f-block size $\geq m + 1$.

The proof of this claim essentially consists of two steps: First, we observe that the set of facts generated by T_1 and by T_2 have labeled nulls of the form $\hat{f}(\vec{a})$ in common, where $\vec{a}$ is a vector of constants used in the variable instantiations at the joint ancestor triggerings of T_1 and T_2. This is an immediate consequence of Lemma 3.14. It can then be shown that also the facts generated by T_3 contain these nulls. In other words, the facts generated by T_3 in $chase(I', \sigma)$ are connected to the facts generated by T_1 and by T_2. To complete the proof of Claim 3, it remains to show that B persists in the $chase(I', \sigma)$ and that at least one more fact is connected to B and also persists in the core. The latter property can be established for the fact A_3 generated by T_3, which corresponds to A_i in the facts generated by T_i for $i \in \{1, 2\}$. Since A_3 contains a null $\hat{f}(\vec{a} \cdot \vec{d}')$ with $\vec{d}' \subseteq \vec{d}$, no endomorphism on $chase(I', \sigma)$ can map it to one of the facts in $B \subseteq chase(I', \sigma)$. □

It is worth briefly reflecting on the different proof methods applied in Theorem 3.1 (in particular, Lemma 3.22) and in Theorem 4.10 above. Recall that the proofs in Section 3 made use of k-patterns, where k depends on the size of the nested tgds involved. In contrast, the proof of Theorem 4.10 above only uses 2-patterns—independently of the size of the nested tgds. In principle, the crucial step in both cases is the extension of a homomorphism: In case of Lemma 3.22, the very essence of the lemma is the existence of an extension of a given homomorphism. In case of Theorem 4.10, the extension of a homomorphism is hidden in the proof of step 3. The important difference between these two proofs is that in case of Lemma 3.22, the extension of the homomorphism has to be constructed yet. In case of Theorem 4.10, we just need to prove certain properties of an assumed extension of the original homomorphism. Obviously, actually constructing the extension of a given homomorphism is more difficult than establishing certain properties of the assumed extension of a given homomorphism. Hence, the former requires stronger tools, namely, k-patterns. It will turn out that this is also the case in the proof of Theorem 4.14 below.

But first let us recall the following claim from Reference [7]:

Claim 1. *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 4.3 would follow immediately by combining Theorem 4.2 and Theorem 4.10 with Claim 1. Though the algorithm for Claim 1 presented in Reference [7, Theorem 5.2] appears to be correct, the proof of correctness of the algorithm given there has a flaw, which

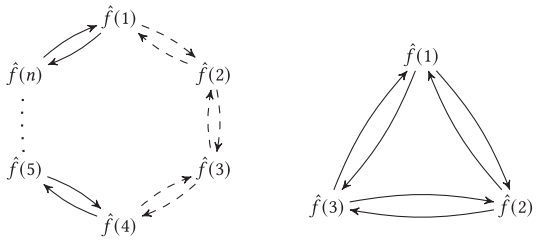

Fig. 8. The undirected cycle of length n on the left side and of length 3 on the right side.

will be pointed out in the sequel. It should be noted that the above claim would also follow from Theorem 3 in Reference [24] together with Theorem 4.10 in Reference [9]; however, Theorem 3 in Reference [24] is stated without proof. In view of this state of affairs, we prove that Claim 1 indeed holds for nested GLAV mappings. For this purpose, we introduce the following concept:

Definition 4.11. 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 $J \subseteq core(chase(I, \mathcal{M}))$, there are a source instance I' and a connected target instance J' such that

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

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

We extend this notion to classes of schema mappings.

Definition 4.12. 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 $a(\mathcal{M})$. ◁

For understanding the intuition behind the concept of bounded anchor, let us consider the following example:

Example 4.13. 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. Intuitively, what a bounded anchor gives is the following guarantee: If we find an f-block of a certain size in the target instance, then there is a "small" source instance (the "anchor") that also yields an f-block of at least that certain size in the target instance. Here, "small" means that the source instance is of the same size as the f-block up to a constant factor.

Showing that a schema mapping has bounded anchor is structurally not an easy task, as it involves showing that for *any* source and connected target instance, we can find such an anchor. All the more, it will be interesting to see that nested GLAV mappings indeed always have bounded anchor. Let us now illustrate, based on our example SO tgds σ, why it is not even simple to find such an anchor for a given source instance (or rather, class of source instances).

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 for some $n \geq 3$. 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 8 for $n > 5$ (each arc, whether it is solid or dashed, denotes an R-atom). 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 8 (i.e., six R-atoms denoted by the six 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, depicted on the right side of Figure 8. ◁

Note that Example 4.13 yields a counter-example to a step in the proof of correctness of the algorithm in Theorem 5.2 in Reference [7], where the search for I' and J' was confined to subsets of the given instances I and J.

We show next that the class of nested GLAV mappings indeed has effective bounded anchor.

Theorem 4.14. *The class of nested GLAV mappings has effective bounded anchor.*

Proof Idea. As before, by Lemma 4.5, it suffices to consider a single nested tgd σ. We have to show that there exists a recursive function a that, for a given schema mapping specified by a nested tgd σ, returns an integer $a(\sigma)$ witnessing bounded anchor for σ. Let I be a source instance and let $J \subseteq core(chase(I, \sigma))$ be connected. We have to show that there exists a source instance I' with $|I'| \leq a(\sigma)|J|$ and connected $J' \subseteq core(chase(I', \sigma))$ with $|J'| \geq |J|$.

The proof reuses the key ideas of the proof of Theorem 3.1: Analogously to Lemma 3.22, let $k = v_\sigma w_\sigma + 1$, and define $a(\sigma)$ as the maximum size of the canonical source instance of any k-pattern of σ. As in Lemma 3.18, we may restrict ourselves to canonical source and target instances, referred to as I and J. Moreover, by Lemma 3.17, any pattern p can be obtained from a k-pattern p_0 by a sequence of k^+-increments, i.e., we get a sequence of patterns $p_0, \ldots, p_n$ with $p_n = p$ and $p_{i-1} \leq_k p_i$ for every i. By Lemma 3.22, we know that if $p_i \leq_k p$ holds, then certain homomorphisms from J_{p_i} can be extended to homomorphisms from J_p. We use this fact to argue that any homomorphism h from J_{p_i} to $core(chase(I_{p_i}, \sigma))$ preserves the subinstance $J_{p_i} \cap J$ of $core(chase(I_{p_i}, \sigma))$, since, otherwise, the extension of h to a homomorphism from J to $core(chase(I, \sigma))$ would also reduce the subinstance J of $core(chase(I, \sigma))$. □

Having shown that the class of nested GLAV mappings has effective bounded anchor, we can now prove that Claim 1 indeed holds for the class of nested GLAV mappings.

Theorem 4.15. *Let 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 C and a positive integer b, is the f-block size of $\mathcal{M}$ at most b?*

Proof. Let a denote the witness of 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 larger than b ("no"). Otherwise, we return that the f-block size is at most b ("yes").

For the correctness, it is clear that in the "no" case, if we witness an f-block of size larger than b for some source instance, then the f-block size is indeed larger than b. We thus have to show that in the "yes" case, if for all source instances K with $|K| \leq a(b + 1)$ the f-block size of $core(chase(K, \Sigma))$ is at most b, then also for all other, larger source instances the f-block size is at most b. We will prove this by showing the following: If for source instance I with $|I| > a(b + 1)$ we have that $core(chase(I, \Sigma))$ has f-block size at least $b + 1$, then there is also some source instance I' with $|I'| \leq a(b + 1)$ s.t. $core(chase(I', \Sigma))$ has f-block size at least $b + 1$.

Assume that $core(chase(I, \mathcal{M}))$ contains an f-block B of size at least $b + 1$. We take an arbitrary connected $J \subseteq B$ with $|J| = b + 1$. By assumption, J is connected. Since Σ has bounded anchor

witnessed by a, we know that there is I' with $|I'| \leq a|J|$ and connected $J' \subseteq core(chase(I', \mathcal{M}))$ with $|J'| \geq |J|$. Now, since $|J| = b + 1$, we know that $|I'| \leq a|J| = a(b + 1)$ satisfies the requirement on the size of I'. Since J' is connected, and $|J'| \geq |J| = b + 1$, we also produce the required size of the f-block. □

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

Theorem 4.16. *Let 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 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 C. Consider the following algorithm: Given a mapping $\mathcal{M}$ in C, compute the bound $b = f(\mathcal{M})$ for the effective threshold for f-block size. Since C has effective bounded anchor, we can use the algorithm in Theorem 4.15 to test whether $\mathcal{M}$ has f-block size bounded by b. If it does, return that $\mathcal{M}$ has bounded f-block size; otherwise, 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 4.3. By Theorem 4.10 and Theorem 4.14, the class of nested GLAV mappings has both effective threshold for f-block size and effective bounded anchor. Therefore, by Theorem 4.16, the following problem is decidable: Given a nested GLAV mapping, does it have bounded f-block size? Thus, together with Theorem 4.2, we get the decidability result stated in Theorem 4.3. □

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 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 methods for doing this are Ehrenfeucht-Fraïssé games or locality methods (see Reference [23]). In fact, this method is behind the proof in Reference [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 4.2. 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 is contained in one of these categories, then we can conclude that indeed σ is not logically equivalent to any nested GLAV mapping.

F-Degree. 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 C is a class of source instances. We say that $\mathcal{M}$ has *bounded f-block size on* C if there is a positive integer b such that for every source instance I in 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* 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* C if there is a positive integer d such that for every source instance I in 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* C.

Before illustrating the close connection between f-block size and f-degree for nested GLAV mappings, we show that there is no such close connection between f-block size and f-degree for plain SO tgds. This simple proof for SO tgds allows the reader to get an intuition of the two concepts before the more complex construction for nested GLAV mappings.

Proposition 4.17. *There is a plain SO tgd τ and a class C of source instances such that τ has unbounded f-block size on C, but bounded f-degree on 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 C be the class of all source instances I such that S is a successor relation. If $I \in C$, 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 C is unbounded, but the f-degree of τ on C is 2. □

We now show the close connection between f-degree and f-block size for nested GLAV mappings. From this it will immediately follow that the plain SO tgd τ in Proposition 4.17 is not logically equivalent to any nested GLAV mapping.

Theorem 4.18. *Let $\mathcal{M}$ be a nested GLAV mapping and C a class of source instances. Then $\mathcal{M}$ has bounded f-block size on C if and only if $\mathcal{M}$ has bounded f-degree on C.*

The theorem follows immediately from the following lemma. Note that the lemma holds even if we consider the Gaifman graph of facts for arbitrary instancesnot just for the core of some chase result. But for our purposes it will suffice to prove this restricted version.

Lemma 4.19. *Let $\mathcal{M}$ be a nested GLAV mapping. The following claims hold:*

(1) For an arbitrary source instance I, it holds that $core(chase(I, \mathcal{M}))$ has f-block size greater than f-degree.
(2) For each integer k, there exists an integer m_k, such that for an arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_k$, then it has f-degree $g \geq k$.

Proof. (1) Since the f-degree g of $core(chase(I, \mathcal{M}))$ is the maximum number of facts sharing at least one null with a given fact $R(\vec{c})$, the f-block size of $core(chase(I, \mathcal{M}))$ is at least $g + 1$. ◁

(2) We start by showing the following claim:

Claim A. *For each integer n, there exists an integer m_n such that for an arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_n$, then at least one null occurs at least n times in $core(chase(I, \mathcal{M}))$.*

Proof of Claim A. Let n be an arbitrary integer. We have to show that there exists an integer m_n such that for an arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_n$, then there are at least n occurrences of some null in $core(chase(I, \mathcal{M}))$.

Let w denote the maximum conclusion size (i.e., number of atoms in the conclusion) of a part of a nested tgd of $\mathcal{M}$. Let p denote the maximum number of parts in a nested tgd of $\mathcal{M}$. Let d denote the maximum nesting depth of a nested tgd of $\mathcal{M}$. Let e denote the maximum number of existentially quantified variables in a part of a nested tgd of $\mathcal{M}$. Define $m_n = w(p\,\ell)^d$ where $\ell = n\,d\,e$. In the rest of the proof of this claim, we will show why this choice of m_n yields our desired result.

Let I be an arbitrary source instance and assume that $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_n$. Let J_B be an arbitrary f-block of $core(chase(I, \mathcal{M}))$ of size $b \geq m_n$. Let T be the RC-chase tree w.r.t. J_B. Since J_B has size at least $m_n = w(p\,\ell)^d$, and we know that a single triggering can produce at most w facts, we know that there must be at least $\lceil m_n/w \rceil = (p\,\ell)^d$ triggerings in T. Since the height of T is bounded by d, we know that some triggering in T must have at least $p\,\ell$ child triggerings. Note that by definition, the child triggerings of a triggering in a chase tree have the same input assignment $\vec{a}_0$. Of these $p\,\ell$ triggerings, since there are at most p parts of nested tgds in $\mathcal{M}$, one of these parts σ_j must have at least ℓ triggerings $t_1, \ldots, t_\ell$.

Let $J_i = \{facts(t) \mid t \in desc(t_i)\}$ denote the subinstance of J_B generated by the subtree $desc(t_i)$ rooted at t_i. Note that all of these J_i are non-empty by the definition of T. From Lemma 3.14, we can conclude that $J_1 \cup \ldots \cup J_\ell$ can belong to the same f-block only if each of the facts in $J_1 \cup \ldots \cup J_\ell$ contains a null $\hat{f}(\vec{c})$ with f from $anc(\sigma_j)$ and $\vec{c}$ a prefix of $\vec{a}_0$. Observe that there are at most $d\,e$ Skolem functions f (as e denotes the maximum number of existentially quantified variables in a part of a nested tgd in $\mathcal{M}$ and d bounds the depth)—and thus at most $d\,e$ nulls instantiating the Skolem terms in $J_1 \cup \ldots \cup J_\ell$ (since their arguments $\vec{c}$ are a prefix of $\vec{a}_0$). Observe that there are at least $n\,d\,e$ facts in $J_1 \cup \ldots \cup J_\ell$ (as $\ell = n\,d\,e$ and all J_i are non-empty). Thus, there must exist at least one null that occurs at least $n\,d\,e/d\,e = n$ times, which was to be shown. ◁

It is not difficult to see that Claim A implies the second claim of the lemma. Indeed, let k be an arbitrary integer. We have to show that there exists an integer m_k, such that for an arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_k$, then it has f-degree $g \geq k$. Let r denote the maximum arity of a relation in the target schema of $\mathcal{M}$. We now invoke Claim A with $n = r(k+1)$, yielding an integer m_n such that for an arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_n$, then at least one null occurs at least n times in $core(chase(I, \mathcal{M}))$. Define $m_k = m_n$. Let I be an arbitrary source instance such that $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_k$ (and, equivalently, $b \geq m_n$). We thus know that there are at least n occurrences of some null in $core(chase(I, \Sigma))$.

Since there are $n = r(k+1)$ occurrences of one null in a block of $core(chase(I, \mathcal{M}))$, there are at least $\lceil n/r \rceil = k+1$ facts in this block (as r is the maximum arity of a relation in the target schema). Moreover, these facts give rise to a clique of the same size $\lceil n/r \rceil$ in the Gaifman graph of facts of $core(chase(I, \mathcal{M}))$, and thus the f-degree g of $core(chase(I, \mathcal{M}))$ is at least $\lceil n/r \rceil - 1 = k$. That is, it holds that f-degree $g \geq k$ as desired in the second claim of the lemma, which was to be shown. □

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 Proposition 4.17.

Altogether, f-degree is an easy-to-use tool for showing that a schema mapping is not logically equivalent to a nested GLAV mapping (as we did in Proposition 4.17). However, it is not always sufficient for dealing with more complex schema mappings, as the next example shows.

Example 4.20. Consider the following plain SO tgd σ:

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

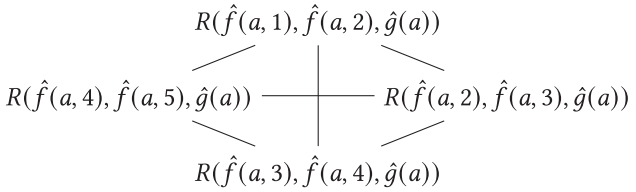

Fig. 9. Gaifman graph of facts of Example 4.20 for a successor relation of length 5.

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 C of source instances, σ has unbounded f-block size on C if and only if it has unbounded f-degree on C. For example, if 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 Figure 9. Thus, Theorem 4.18 cannot be used to show that σ is not logically equivalent to any nested GLAV mapping. ◁

The preceding example shows that, in addition to Theorem 4.18, 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 (up to isomorphism) on some classes of instances, yet the plain SO tgd is not logically equivalent to any nested GLAV mapping.

Example 4.21. 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 isomorphic to those of the plain SO tgd σ in Example 4.20, i.e., they are complete graphs (see Figures 9 and 11). Yet, as we are about to discover, σ is not logically equivalent to any nested GLAV mapping. Intuitively, the reason is that σ can "build arbitrarily long chains of nulls," while σ' can not, but we shall now introduce a concrete tool that allows one to make this distinction on a formal level. ◁

Path Length. To cope with this situation, we need to look beyond the Gaifman graph of facts. Recall that the *Gaifman graph of facts* 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 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 Gaifman graph of nulls can be used to show inexpressibility in situations where the structure of the Gaifman graph of facts 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 Gaifman graph of nulls of $core(chase(I, \mathcal{M}))$ is at most l; otherwise, we say that $\mathcal{M}$ has *unbounded path length*.

Theorem 4.22. *Every nested GLAV mapping has bounded path length.*

Proof. Let $\mathcal{M}$ be a nested GLAV mapping. Let I be an arbitrary source instance. We show that the length of the longest simple path in the Gaifman graph of nulls of $core(chase(I, \mathcal{M}))$ is bounded by a constant only depending on $\mathcal{M}$. Let $y_1, \ldots, y_n$ be an arbitrary simple path in the

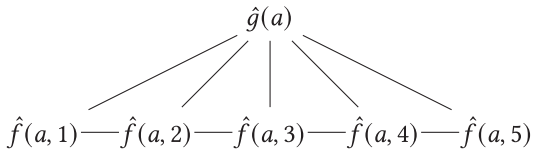

Fig. 10. Gaifman graph of nulls of Example 4.20 for a successor relation of length 5.

Gaifman graph of nulls of $core(chase(I, \mathcal{M}))$. We denote by e_i the edge (y_{i-1}, y_i) of this path and by A_i the atom in $core(chase(I, \mathcal{M}))$ where y_{i-1} and y_i occur together (if there is more than one such atom, we choose an arbitrary one).

Let $\mathcal{A} = A_1 \cup \ldots \cup A_n$ and let T be the RC-chase tree w.r.t. $\mathcal{A}$. Let d be the maximum depth of a nested tgd of $\mathcal{M}$. Let e be the maximum number of existentially quantified variables in a part of a nested tgd of $\mathcal{M}$.

Claim 1. *In T, every node has at most $m = d\,e + 1$ child nodes.*

Proof of Claim 1. Consider an arbitrary node t in T. Suppose that t has s child nodes $t_1, \ldots, t_s$. By construction of T, the subtree rooted at each t_α contains at least one atom $A_{i_\alpha} \in \mathcal{A}$. If there are several such atoms of $\mathcal{A}$ in the subtree rooted at node t_α, then we choose A_{i_α} arbitrarily. W.l.o.g., assume that the nodes $t_1, \ldots, t_s$ are arranged in such an order that $i_1 < i_2 < \cdots < i_s$.

Let $\alpha \in \{1, \ldots, s-1\}$ be arbitrary. Consider t_α and $t_{\alpha+1}$: There must exist atoms A_j and A_{j+1} with $i_\alpha \leq j < j+1 \leq i_{\alpha+1}$ such that A_j is in the subtree rooted at t_α and A_{j+1} is outside this subtree. By definition of the atoms A_i (i.e., the fact that they realize the edges of the path $y_1, \ldots, y_n$ in the Gaifman graph of nulls), A_j and A_{j+1} contain the null y_j. Since A_j is *in* the subtree below t_α and A_{j+1} is *outside* this subtree, y_j must be one of the nulls introduced in the branch from the root to t, i.e., y_j corresponds to a null $\hat{f}(\vec{c})$ where $\vec{c}$ is a prefix of the input assignment $\vec{a}_0$ of t_α (which is the same as the input assignment of $t_{\alpha+1}$, as sibling triggerings by definition have the same input assignments).

Recall that we are considering a simple path $y_1, \ldots, y_n$. Hence, the nulls y_j that exist for each α must be pairwise-distinct for different values of α. The number of nulls y_j (which, as discussed before, are of the form $\hat{f}(\vec{c})$ where $\vec{c}$ is a prefix of the input assignment $\vec{a}_0$ shared by all t_α) is bounded by $d\,e$. Hence, the number s of child nodes t_α of t must be bounded by $m = d\,e + 1$. ◁

By Claim 1, it holds that T has at most m^d nodes. Let w be the maximum conclusion size, i.e., the maximum number of atoms in the conclusion of a part of a nested tgd of $\mathcal{M}$. Then $\mathcal{A}$ contains at most $m^d\,w$ atoms. Let r be the maximum arity of the relation symbols in the target schema. Thus, since each atom can contain at most r nulls, and all nulls $y_1, \ldots, y_n$ are contained in the atoms $\mathcal{A} = \{A_1, \ldots, A_n\}$, we get the upper bound $m^d\,w\,r$ on the number n of nulls. □

Equipped with Theorem 4.22, we now have a tool to show that the plain SO tgd σ of Example 4.20 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 Figure 10, where the Gaifman graph of nulls contains a simple path of length 4, in contrast to Figure 12).

5 ADDING SOURCE CONSTRAINTS

In the previous section, we showed that it is decidable whether a schema mapping specified by 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, we will show that for nested tgds and

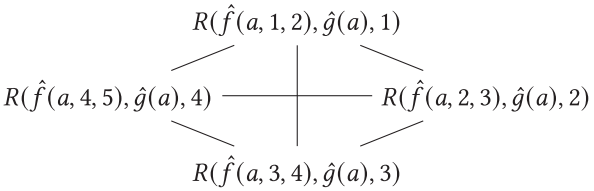

Fig. 11. Gaifman graph of facts of Example 4.21 for a successor relation of length 5.

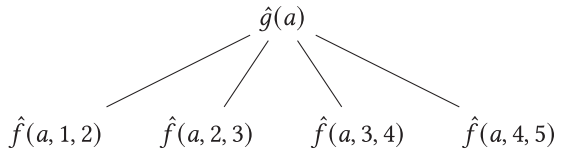

Fig. 12. Gaifman graph of nulls of Example 4.21 for a successor relation of length 5.

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 4.2 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 Reference [9], thus played an important role in Section 4. A close inspection of the proofs of Proposition 3.14 and Theorem 4.10 in Reference [9] shows that these results (and therefore Theorem 4.2) still hold in the presence of source egds. This fact is used 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. By the above comments it suffices to show the undecidability of the following problem: Given a schema mapping specified by a plain SO tgd and a source key dependency, does the mapping have 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. We first describe the basic structure of our proof and then give the construction. The basic structure 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.

Construction. We now show the reduction, and we give the correctness proof in the Appendix. Let TM be a Turing machine. We now define a schema mapping from a source schema to a target schema specified by an SO tgd σ. We assume w.l.o.g. that the Turing machine starts from the empty tape, and that from the halting state, there are no further transitions possible.

Source Schema. For every alphabet symbol a_i of the Turing machine, there is a unary predicate A_i in the source schema. The intended meaning is that if $A_i(x)$ holds, then x equals a_i. Furthermore, we assume that $a_0 = \triangleright$ (i.e., the special symbol always marking the left end of the tape) and $a_1 = \sqcup$ (i.e., the blank symbol). Similarly, for every state q_i there is a unary predicate Q_i in the source schema, where q_0 denotes the starting state and q_1 the halting state. Also, we will use the unary predicate C_1 to represent a cursor being present, and C_0 for the cursor being absent.

For describing time and space of the Turing machine computation, we add a unary predicate Z ("zero") as well as a binary predicate S ("successor") to the source schema. The intended meaning is for Z to be the root of a linear order, i.e., $Z(0)$, and for S to describe the order itself, i.e., $S(0, 1), S(1, 2)$, and so on. For describing the configurations of the Turing machine, we use predicates state and tape. The intended meaning is that $\mathsf{state}(x, q)$ denotes that at time x, the machine is in state q. Similarly, $\mathsf{tape}(x, y, a, c)$ denotes that at time x and tape position y, the symbol a is stored and the cursor is at this position ($c = 1$) or not ($c = 0$).

Target Schema. The target schema contains a unary predicate O ("origin") and a binary predicate N ("next"). The intended meaning is to represent a linear order, rooted at O and the successor relation stored in N. That is, O and N are comparable to Z and S of the source schema. However, the target schema is intended to represent a one-dimensional enumeration of the two-dimensional space-time structure of the Turing machine computation. We will illustrate this idea in detail later.

Dependency. We will now construct the SO tgd σ by describing all parts of the tgd (where all individual symbols are considered to be universally quantified). The final SO tgd is obtained by constructing a conjunction of those parts and adding existential function quantifiers for all function symbols. We first define the abbreviation *fail*, which, when put on the right-hand side of an SO tgd part and triggered, will lead to the core "collapsing."

- $\mathit{fail} := O(x) \wedge N(x, x)$

Chain Guards. We require that the start of our linear order described by Z has no predecessor and thus add the following part to our SO tgd:

- $Z(y) \wedge S(x, y) \rightarrow \mathit{fail}$.

We also require S to be irreflexive:

- $S(x, x) \rightarrow \mathit{fail}$.

Note that we do not directly enforce functionality on S.

Uniqueness Guards. We need to make sure that for every time instant x there is only one distinct state stored. Also, for every time instant x and tape position y, we require that only one distinct tape symbol and cursor information is stored. For each $i \neq j$, we thus add the following parts:

- $\mathsf{state}(x, q_i) \wedge \mathsf{state}(x, q_j) \wedge Q_i(q_i) \wedge Q_j(q_j) \rightarrow \mathit{fail}$,
- $\mathsf{tape}(x, y, a_i, \_) \wedge \mathsf{tape}(x, y, a_j, \_) \wedge A_i(a_i) \wedge A_j(a_j) \rightarrow \mathit{fail}$,
- $\mathsf{tape}(x, y, \_, c_i) \wedge \mathsf{tape}(x, y, \_, c_j) \wedge C_i(c_i) \wedge C_j(c_j) \rightarrow \mathit{fail}$.

Transition Guards. The symbol stored at a specific tape position y and time instant x, as well as the fact whether the cursor is located at this tape cell, depends only on the previous time instant x', and there only on tape position y, the tape position on the left y_l, and the one on the right y_r.

We now need to check whether the configuration at time instant x and tape position y is correct according to the definition of the Turing machine. We thus collect all required information for current state q_α, tape symbol a_β, and cursor information c_β (1). That is, the previous state $q_{\alpha'}$ as well as tape symbol and cursor information at the middle (2), left (3), and right (4) positions in the previous configuration (a_{β^p} and c_{β^p} for $p \in \{l, m, r\}$). For such a setting $\pi = (q_\alpha, q_{\alpha'}, a_\beta, a_{\beta^l}, a_{\beta^m}, a_{\beta^r}, c_\beta, c_{\beta^l}, c_{\beta^m}, c_{\beta^r})$, we can thus define the following abbreviation $check_\pi[x, y]$:

$$
\begin{aligned}
&check_\pi[x, y] := \\
&\mathrm{state}(x, q_\alpha) \wedge Q_\alpha(q_\alpha) \wedge \mathrm{tape}(x, y, a_\beta, c_\beta) \wedge A_\beta(a_\beta) \wedge C_\beta(c_\beta) && (1)\\
&\wedge S(x', x) \wedge \mathrm{state}(x', q_{\alpha'}) \wedge Q_{\alpha'}(q_{\alpha'}) \wedge \mathrm{tape}(x', y, a_{\beta^m}, c_{\beta^m}) \wedge A_{\beta^m}(a_{\beta^m}) \wedge C_{\beta^m}(c_{\beta^m}) && (2)\\
&\wedge S(y_l, y) \wedge \mathrm{tape}(x', y_l, a_{\beta^l}, c_{\beta^l}) \wedge A_{\beta_l}(a_{\beta^l}) \wedge C_\beta(c_{\beta^l}) && (3)\\
&\wedge S(y, y_r) \wedge \mathrm{tape}(x', y_r, a_{\beta^r}, c_{\beta^r}) \wedge A_{\beta_r}(a_{\beta^r}) \wedge C_\beta(c_{\beta^r}). && (4)
\end{aligned}
$$

For an arbitrary setting π, we can clearly identify whether or not it correctly follows the description of the Turing machine. We thus build the two sets Π_{good} as the set of all π that follow the definition of TM, and Π_{bad} for those that do not. For later use, we also define the special setting $\star = (\_, \_, \_, \_, \_, \_, \_, \_, \_, \_)$ containing anonymous variables instead of constants. The intuition behind this setting is that it only checks for the existence of the relevant tuples and not in addition checks for good or bad ones.

For instance, when in a setting $\pi \in \Pi_{\text{good}}$ all c_β cursor markers have the value 0, then this setting corresponds to an inertia rule (i.e., a rule that enforces that, unless changed, the configuration stays the same). When at least one of $c_{\beta^l}, c_{\beta^m}, c_{\beta^r}$ has the value 1, then this represents a transition rule. A clear example of a $\pi \in \Pi_{\text{bad}}$ is when $c_{\beta^l}, c_{\beta^m}, c_{\beta^r}$ all have the value 0 and c_β has the value 1. Note that these sets Π_{good} and Π_{bad} are clearly finite for a fixed Turing machine.

Now for all $\pi_{bad} \in \Pi_{\text{bad}}$, we add the part:

- $check_{\pi_{bad}}[x, y] \rightarrow fail$.

Note that for the border cases of the two-dimensional configuration structure (initial time instant, first tape position, the diagonal after which only blank symbols are present) adaptations need to be made to $check_\pi[x, y]$. They are technical but do not provide deep insight. In particular, (2) needs to be omitted in the initial time instant, (3) needs to be omitted at the first tape position, and (4) needs to be omitted at the diagonal.

Constructing the Target. Our intention is for the target instance to represent an enumeration of the configuration matrix of the Turing machine. The structure of this enumeration is illustrated in Figure 13. The vertical axis represents time (starting at the top) and the horizontal axis the tape (starting at the left). 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 4th tape cell. To realize this enumeration, we first need to materialize the origin O (corresponding to the black square in Figure 13):

- $Z(x) \rightarrow O(f(x, x))$.

We also need to realize the arcs of the enumeration as N atoms. There are two possible cases. The $\leftarrow$ step goes from the given tape position to the previous one at the same time instant. The $\searrow$

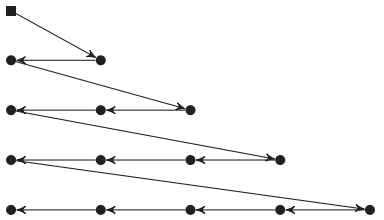

Fig. 13. Intended enumeration represented in the target instance. Arrows denote N atoms. The origin node represented by O is marked by a rectangle.

step to the diagonal in the configuration matrix and the next time instant is made when the initial tape position is reached. These are implemented as follows as SO tgd parts:

- $check_\star[x, y] \wedge S(y, y') \rightarrow N(f(x, y'), f(x, y))$ ($\leftarrow$ transition),
- $check_\star[x', x'] \wedge S(x, x') \wedge Z(y) \rightarrow N(f(x, y), f(x', x'))$ ($\searrow$ transition).

Note that the conditions in the antecedents of the above implications are mutually exclusive, since for any y for which $Z(y)$ holds, there cannot be a predecessor $S(\_, y)$.

The 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 will 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 13.

Unfoundedness Traps. The idea of the following parts is to get rid of "unfounded" paths, that is, paths not beginning at an origin O. For each combination of time instant x and tape position y, we add a self-loop in N using the function symbol h and then connect this self-loop to the corresponding function symbol f. The effect is that in the core, paths that do not start at an O atom can be "shrunk" into this self-loop, i.e., the core will only contain our self-loop and the connection, but not an unfounded path of arbitrary length. We add:

- $check_\star[x, y] \rightarrow N(h(x, y), h(x, y))$,
- $check_\star[x, y] \rightarrow N(h(x, y), f(x, y))$.

Key Dependency. We now make sure that in the S relation there is no element with two predecessors by adding the following key dependency:

- $S(x, y) \wedge S(x', y) \rightarrow x = x'$.

Note that the S relation can still have a number of deviations from representing a linear order, in particular an element can have more than one distinct successor. This concludes our construction. Correctness is shown in the Appendix. □

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 Reference [12] 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 4.18, 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 GLAV mappings. We now show that in contrast, the problem of deciding whether a nested GLAV mapping is equivalent to a GLAV mapping is decidable even in the presence of source egds.

The proof strategy of Theorem 4.3 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 4.10 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 $I = \{Q(a), P_1(a, b), P_2(a, b), P_2(a, c)\}$. The proof of Theorem 4.10 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. ◁

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. To this end, it is convenient to allow source egds to *unify two constants*. Formally, this simply means that we replace one of the constants by the other. We will say that instance K' is obtained from instance K by an application of egd ϵ if a firing of ϵ unifies two domain elements b_i and b_j, which means that K' is obtained from K by replacing every occurrence of b_j in K by b_i.

Definition 5.4 (Legal canonical source/target instance). 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 σ, we define the *legal canonical source, respectively, target instances* I_p^s and J_p^s, as the instances obtained from the canonical source instance I_p and target instance J_p as follows: Instance I_p^s is obtained from I_p by chasing it with Σ_s. Each unification of constants performed by this chase is also applied to the constants and instantiations of Skolem terms occurring in J_p resulting in the instance J_p^s. ◁

The following lemma is folklore, e.g., a similar property is shown in Lemma 3.4 of Reference [8]. Intuitively, it states that egd applications carry over from an instance to its homomorphic image. For the sake of completeness, we give a proof of this lemma in the Appendix.

Lemma 5.5. *Let K_1 and K_2 be instances and let θ be a c-homomorphism from K_1 to K_2. Let Σ be a set of egds and let K_1' denote an instance that is obtained from K_1 by a sequence S_1 of applications of egds from Σ. Then there exists a sequence S_2 of applications of egds from Σ to instance K_2, transforming K_2 into instance K_2', such that there exists a c-homomorphism θ' from K_1' to K_2'. Moreover, whenever S_1 leads to the unification of two domain elements $s_1, s_2 \in dom(K_1)$, then also S_2 leads to the unification of $\theta(s_1), \theta(s_2) \in dom(K_2)$.*

In Section 3, the cloning of subtrees in a pattern p of some nested tgd σ played an important role. Adding an isomorphic copy of some (complete) subtree in p and taking the canonical source/target instance of the extended pattern was straightforward. As mentioned above, in the presence of source egds, we have to consider *legal* canonical source/target instances. Hence, when adding an isomorphic copy of some subtree in p, we have to make sure that we again get a *legal* canonical source instance of the extended pattern. The following two lemmas show how this can be ensured without explicitly carrying out the chase with the source egds.

Definition 5.6. Let σ be a nested tgd. Let K be a set of facts generated by a subtree T of some (RC-)pattern p of σ, and $\vec{a}$ be the tuple of local constants of T (as defined in Section 4.1). The chase with Σ_s unifies constants in I_p, transforming I_p, K, and $\vec{a}$, respectively, into the legal canonical source instance I_p^s, the set of facts K^s in the canonical target instance of p, and the tuple of constants $\vec{a}^s$, which we call *pseudolocal constants* of T (respectively, of K^s).

Note that due to the source egds Σ_s, the elements of $\vec{a}^s$ need not be pairwise-distinct.

Lemma 5.7. *Let σ be a nested tgd and let Σ_s be a set of source egds. Let p be a k-pattern of σ with $k \geq 2$ and consider two isomorphic (complete) subtrees T_1 and T_2 in p such that the roots of T_1 and T_2 are siblings. Let I_p denote the canonical source instance of p and let $\vec{b}$ (respectively, $\vec{c}$) denote the local constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p. Let I_p^s denote the legal, canonical source instance of p and let $\vec{b}'$ (respectively, $\vec{c}\,'$) denote the pseudolocal constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p^s. We write b'_i (respectively, c'_i) to refer to the ith component of $\vec{b}'$ (respectively, $\vec{c}\,'$). Then $\vec{b}'$ and $\vec{c}\,'$ fulfill the following properties:*

(1) for all i, j, if $b'_i = b'_j$ in I_p^s, then also $c'_i = c'_j$ holds;
(2) for all i, j, if $b'_i = c'_j$ in I_p^s, then also $c'_i = c'_j$ holds.

Note that the properties in Lemma 5.7 are symmetric in T_1 and T_2 and, hence, also in $\vec{b}'$ and $\vec{c}\,'$. Property (1) can thus be strengthened to the equivalence that $b'_i = b'_j$ holds in I_p^s if and only if $c'_i = c'_j$ holds. Moreover, by applying Property (1) to Property (2), we obtain that $b'_i = c'_j$ in I_p^s implies all of the equalities $b'_i = c'_i = c'_j = b'_j$. We apply these properties to construct the legal canonical source instance after adding a clone without explicitly carrying out the chase with the source egds.

Lemma 5.8. *Let σ be a nested tgd and let Σ_s be a set of source egds. Let p be a k-pattern of σ with $k \geq 2$ and consider two isomorphic (complete) subtrees T_1 and T_2 in p such that the roots of T_1 and T_2 are siblings. Let I_p^s denote the legal, canonical source instance of p and let $\vec{b}'$ (respectively, $\vec{c}\,'$) denote the pseudolocal constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p^s. As in Lemma 5.7, we write b'_i (respectively, c'_i) to refer to the ith component of $\vec{b}'$ (respectively, $\vec{c}\,'$).*

Let r be obtained from p by adding another isomorphic subtree T_3 as sibling of T_1 and T_2. Let $\vec{d}$ denote the vector of fresh, pairwise-distinct constants used in the variable instantiations of the parts in T_3 when constructing the canonical source instance I_r. Moreover, let I_r^s denote the legal canonical source instance of r. We claim that I_r^s can be obtained from I_p^s by leaving the variable instantiations at all parts outside T_3 unchanged and by constructing the vector $\vec{d}'$ of pseudolocal constants in T_3 from $\vec{d}$ as follows:

(1) for all i, if $b'_i = c'_i$, then set $d'_i = b'_i$;
(2) for all i, j, if $b'_i = b'_j$, then set $d'_i = d'_j$.

Proof. We divide the proof in two steps, showing that the unifications required according to (1) and (2) above are necessary and sufficient. Both steps rely on the one-to-one correspondence

between the triggerings in T_1, T_2, and T_3 and, likewise, between the variable instantiations $\vec{b}$, $\vec{c}$, and $\vec{d}$ for the canonical source instance I_r.

Necessary unifications. Let p' be equal to pattern p but use $\vec{b}$ and $\vec{d}$ as local constants of T_1 and T_2 to produce the canonical source instance $I_{p'}$. Consider the c-homomorphism θ from I_p to $I_{p'}$ with $\theta(c_i) = d_i$ for every i and $\theta(s) = s$ otherwise. Now assume that the chase of I_p with Σ unifies the elements b_i and c_i. Then, by Lemma 5.5, the chase of $I_{p'}$ with Σ unifies the elements $\theta(b_i)$ and $\theta(c_i)$ with $\theta(b_i) = b_i$ and $\theta(c_i) = d_i$. By $I_{p'} \subseteq I_r$, we thus also have in I_r^s the equality $d_i' = b_i'$ as requested by condition (1).

To show that the unifications according to condition (2) are also necessary, we again consider pattern p' with canonical source instance $I_{p'}$. By the isomorphism of I_p and $I_{p'}$, if the chase of I_p with Σ unifies b_i and b_j, then so does the chase of $I_{p'}$ with Σ. But then, by applying Lemma 5.7 to $I_{p'}$, we conclude that $d_i' = d_j'$ is enforced by the chase of $I_{p'}$. By $I_{p'} \subseteq I_r$, we thus also have in I_r^s the equality $d_i' = d_j'$ as requested by condition (2).

Sufficient unifications. The proof is indirect. Suppose that the chase of I_r with Σ enforces further unifications. We distinguish several cases and derive a contradiction in each of them.

Case 1. Suppose that some d_i is unified with some d_j although $b_i' \neq b_j'$ holds in I_p. Consider the c-homomorphism θ from I_r to I_p with $\theta(d_i) = b_i$ for every i and $\theta(s) = s$ otherwise. Since we assume that the chase of I_r with Σ unifies d_i with d_j, the chase I_p with Σ unifies b_i with b_j by Lemma 5.5. This contradicts our assumption that $b_i' \neq b_j'$ holds in I_p^s.

Case 2. Suppose that some d_i is unified with b_i (or, symmetrically with some c_i) although this equality is not enforced by conditions (1) and (2). In particular, this means that $b_i' \neq c_i'$ in I_p. Consider the c-homomorphism θ from I_r to I_p with $\theta(d_i) = c_i$ for every i and $\theta(s) = s$ otherwise. By Lemma 5.5 and by condition $d_i' = b_i'$ in I_r^s, we conclude that $c_i' = b_i'$ holds in I_p^s—a contradiction.

Case 3. Suppose that some d_i is unified with some b_j (or, symmetrically with some c_j) with $i \neq j$ although this equality is not enforced by conditions (1) and (2). Again, we consider the c-homomorphism θ from I_r to I_p with $\theta(d_i) = c_i$ for every i and $\theta(s) = s$ otherwise. By Lemma 5.5 and by condition $d_i' = b_j'$ in I_r^s, we conclude that $c_i' = b_j'$ holds in I_p^s. By Lemma 5.7, all the equalities $c_i' = b_i' = b_j' = c_j'$ hold in I_p^s. But then condition (1) is applicable and enforces the equalities $d_i' = b_i'$ and $d_j' = b_j'$. Together with $b_i' = b_j'$, also the equality $d_i' = b_j'$ is thus enforced by condition (1)—a contradiction.

Case 4. Suppose that some d_i is unified with some t in $dom(I_r)$ outside $\vec{b}$, $\vec{c}$, and $\vec{d}$, although this equality is not enforced by conditions (1) and (2). We consider two c-homomorphisms θ_1 and θ_2 from I_r to I_p with $\theta_1(d_i) = b_i$ and $\theta_2(d_i) = c_i$ for every i and $\theta_1(s) = s$ and $\theta_2(s) = s$ otherwise. We are assuming that d_i is unified with some t outside $\vec{b}$, $\vec{c}$, and $\vec{d}$ by the chase of I_r with Σ. By applying Lemma 5.5 to θ_1, we conclude that c_i is unified with t by the chase of I_p with Σ. By applying Lemma 5.5 to θ_2, we conclude that also b_i is unified with t by the chase of I_p with Σ. But then, in total, b_i is unified with c_i by the chase of I_p with Σ. Hence, by condition (1), we set $d_i' = b_i'$. Together with $b_i' = s$, we conclude that the unification of d_i with s is enforced by conditions (1) and (2). □

Before we can lift Theorem 4.10 to the setting with source egds, we observe that the above definitions and properties of *legal* canonical source instances can be literally carried over from patterns to RC-patterns introduced in Definition 4.7. We are now able to show our desired result.

THEOREM 5.9. *The class of nested tgds with source egds has effective threshold for f-block size.*

Proof. The proof proceeds analogously to the proof of Theorem 4.10—replacing canonical instances by *legal* canonical instances. Below, we describe how the proof of Theorem 4.10 has to be adapted to take the set Σ_s of source egds into account.

Again, by Lemma 4.5, it suffices to consider a single nested tgd σ. Moreover, let σ be an arbitrary nested tgd σ and let n denote the maximum size of all RC-1-patterns of σ. Then, we again set the threshold $g(\sigma)$ to $g(\sigma) = n + 1$. Now suppose that there exists a source instance I with $I \models \Sigma$, s.t. $core(chase(I, \sigma))$ has f-block size $m \geq g(\sigma)$. As in the proof of Theorem 4.10, it suffices to show that, from I, we can construct a source instance I' with $I' \models \Sigma$, s.t. $core(chase(I', \sigma))$ has f-block size $\geq m + 1$.

Considering legal canonical source instances only. By Corollary 3.15, we know that the facts in some f-block B of $core(chase(I, \sigma))$ must be generated by a single chase tree in the chase forest $\mathcal{F}_{\sigma,I}$. Now suppose that f-block size m of $core(chase(I, \sigma))$ is realized by some f-block B of $core(chase(I, \sigma))$ and that the facts in B are generated by RC-chase tree T_p with RC-pattern p. Let I^s_p be the legal canonical source instance of p. Then the following property holds:

Claim 1. The f-block size of $core(chase(I^s_p, \sigma))$ is at least m.

The proof idea of this claim is the same as in the proof of Theorem 4.10. We just need now in add Lemma 5.5 to conclude that there exists a c-homomorphism from I^s_p to I and that there again is a one-to-one correspondence between the triggerings in (and the atoms generated by) the chase tree $T_{\sigma,I}$ and the corresponding pattern p equipped with the variable bindings according to the legal canonical source instance I^s_p. We can thus again define the set B_p of atoms in the legal canonical target instance corresponding to the set B in $chase(I, \sigma)$. As in the proof of Theorem 4.10, it can be shown that B_p or a superset thereof must also be an f-block in the core of the canonical target instance. In the remainder of the proof, we may thus assume w.l.o.g. that I is the legal canonical source instance of p.

Isomorphic subtrees in T_p with local constants in atoms of B. As in the proof of Theorem 4.10, the considerations on the threshold $g(\sigma)$ allow us to conclude that there exist (at least) two siblings in the pattern p, which are the roots of isomorphic subtrees T_1 and T_2, such that both T_1 and T_2 generate some facts of B. Note that now that we are considering *legal* canonical source instances, it may happen that local constants of subtrees T_1 and T_2 are possibly unified by the source chase. We therefore strengthen Claim 2 from the proof of Theorem 4.10 to rule out such unifications.

Claim 2. For each $i \in \{1, 2\}$, there exists an atom A_i generated by T_i such that $A_i \in B$, atom A_i contains a pseudolocal constant b_i of T_i, and b_1 and b_2 are not unified by the chase with the source egds.

Again, the proof idea is the same as in the proof of Theorem 4.10, i.e., if no such atoms A_1 and A_2 existed, then we would derive a contradiction by exhibiting an endomorphism on $chase(\sigma, I)$, which further shrinks the core.

Construct another isomorphic subtree. Let $\vec{b}$ denote the pseudolocal constants of T_1 and let $\vec{c}$ denote the pseudolocal constants of T_2. By Lemma 5.8, we know which constant unifications have to be carried out when adding another clone T_3 with pseudolocal constants $\vec{d}$. Let θ be the mapping that realizes the one-to-one correspondence between $\vec{b}$ and $\vec{d}$, i.e., $\theta(b_i) = d_i$ for every b_i in $\vec{b}$ and d_i in $\vec{d}$. We extend I to $I' = I \cup \theta(I)$. For the f-block size of $core(chase(I', \sigma))$, the following property can be shown:

Claim 3. The instance $core(chase(I', \sigma))$ has f-block size $\geq m + 1$.

Again, the proof of this claim follows the same pattern as the proof of Claim 3 in the proof of Theorem 4.10, □

Lemma 5.8 shows how legal canonical instances of a pattern p can be obtained via *extensions* of respective legal canonical instances of a smaller pattern $p' \leq_2 p$, exactly as in the case when no source egds are present. With this provision, we can lift Lemma 3.22 to the case of source egds.

Lemma 5.10. *Consider the following setting:*

- *A set Σ of nested tgds and a nested tgd σ.*
- *Integer $k > m_\sigma w_\Sigma$, where m_σ is the maximum number of nulls in the canonical target instances of 1-patterns of σ and w_Σ is the maximum number of universally quantified variables in any nested tgd in Σ.*
- *Patterns p^k, p of σ where p^k is a k-pattern and $p^k \leq_k p$ holds.*
- *Pairs $(I^s_{p^k}, J^s_{p^k})$ and (I^s_p, J^s_p) of the legal canonical source and the legal canonical target instances of the patterns p^k and p, respectively. We assume that I^s_p and J^s_p have been constructed by extending $I^s_{p^k}$ and $J^s_{p^k}$, as specified in Lemma 5.8.*

Then, every homomorphism h^k from J_{p^k} to chase(I_{p^k}, Σ) can be extended to a homomorphism h from J_p to chase(I_p, Σ).

Proof. As in the case of the original Lemma 3.22, the proof is by induction on the length i of the sequence of k^+-increments of p^k transforming it to p, with the trivial base case $i = 0$. Assume that for some sequence length i, a homomorphism h_i from $J^s_{p_i}$ to chase$(I^s_{p_i}, \Sigma)$ exists, s.t. h_i extends h^k. We show that h_i can be transformed to a homomorphism h_{i+1} from $J^s_{p_{i+1}}$ to chase$(I^s_{p_{i+1}}, \Sigma)$, where p_{i+1} is obtained from p_i by a single k^+-increment.

We thus have trees $T_1, \ldots, T_k$ in p_i that are instantiated, in the legal canonical target instance $J^s_{p_i}$, by the subinstances $K_1, \ldots, K_k$. In particular, if $T_1, \ldots, T_k$ are rooted at nodes with the label σ_i, the trees $T_1, \ldots, T_k$ represent chase trees resulting from the triggerings of the part σ_i with the input assignment $\vec{a}_0$. By $\vec{a}_1, \ldots, \vec{a}_k$, we denote tuples of pseudolocal constants of $T_1, \ldots, T_k$.

The pattern p_{i+1} adds another clone T_{k+1} to the subtrees $T_1, \ldots, T_k$. In the legal canonical target instance, this corresponds to an addition of a further subinstance K_{k+1} with a tuple of pseudolocal constants $\vec{a}_{k+1}$.

Let F denote the set of nulls shared between any two distinct instances in the set $\{K_1, \ldots, K_{k+1}\}$. In the absence of egds, in the proof of Lemma 3.22, we argued that each such shared null has the form $\hat{f}(\vec{b})$ where $\vec{b}$ is a prefix of the common input assignment $\vec{a}_0$ of the triggerings corresponding to the roots of $T_1, \ldots, T_{k+1}$. The effect of the source egds, however, may be a unification of constants a_r in $\vec{a}_j$ with a'_ℓ in $\vec{a}_{j'}$. Source egds may thus also lead to the unification of a null $\hat{f}(\vec{a}_0 \cdot \vec{a}_j)$ in K_j with a null $\hat{f}(\vec{a}_0 \cdot \vec{a}_{j'})$ in $K_{j'}$. Hence, in the presence of source egds, the set F may be significantly bigger than the number of distinct Skolem functions v_σ. However, Lemma 5.8 allows us to derive an upper bound on $|F|$ also in this case. Indeed, Lemma 5.8 establishes precisely the form of the tuple of pseudolocal constants $\vec{a}_{k+1}$ of K_{k+1}. More specifically, if constants are positions r and ℓ are unified in $\vec{a}_j$ and $\vec{a}_{j''}$, then the positions r and ℓ contain the same constant in all tuples $\vec{a}_1, \ldots, \vec{a}_{k+1}$. By the same token, we may conclude that the unification of nulls $\hat{f}(\vec{a}_0 \cdot \vec{a}_{k+1})$ in K_{k+1} and $\hat{f}(\vec{a}_0 \cdot \vec{a}_{j'})$ in $K_{j'}$ is only possible if these nulls are also unified with the corresponding null $\hat{f}(\vec{a}_0 \cdot \vec{a}_j)$ in K_j. In other words, the number of nulls created by ancestor triggerings of $T_1, \ldots, T_n$ plus the number of nulls in K_j give us an upper bound on the number of nulls shared between any two distinct subinstances K_j and $K_{j'}$. That is, additional nulls contained in clones can be neglected for this upper bound. But then we have $|F| \leq m_\sigma$. By Proposition 3.8, this maximum m_σ exists and can be computed.

Having obtained a new upper bound on $|F|$, the remainder of the proof of Lemma 3.22 can be easily carried over to the setting with source egds. □

Applying Lemma 5.10 instead of Lemma 3.22, the argumentation in the proof of Theorem 4.14 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 4.3:

Theorem 5.11. *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 is decidable in the presence of source egds.

Theorem 5.12. *The implication problem for nested tgds is decidable in the presence of source egds, i.e., the following problem is decidable: Given source and target schemas* $\mathbf{S}$, $\mathbf{T}$, *a set* Σ_s *of source egds, a set* Σ *of nested tgds, and a nested tgd* σ, *does* $\Sigma_s \cup \Sigma \models \sigma$ *hold?*

Proof. The proof uses exactly the same ideas as the proof of Theorem 3.1 (i.e, decidability of implication without source constraints) with two crucial modifications:

(1) we have to use *legal* canonical (source and target) instances rather than arbitrary canonical instances;
(2) the upper bound k on the k-patterns to be inspected in the procedure Implies in Section 3 has to be increased to match the precondition of Lemma 5.10.

The proof is analogous to the proof of Theorem 3.1:

Characterization of implication via the existence of homomorphisms. The implication $\Sigma_s \cup \Sigma \models \sigma$ holds if every instance I over source schema $\mathbf{S}$ satisfying Σ_s and Σ also satisfies σ. Analogously to the case without source dependencies, the latter condition is equivalent to the following: For every instance I over source schema $\mathbf{S}$ satisfying Σ_s, there exists a homomorphism from $chase(I, \sigma)$ to $chase(I, \Sigma)$. In other words, the source egds only play a role in restricting the set of source instances I to be considered by this implication criterion. The criterion itself (namely, characterizing implication via the existence of homomorphisms between the chase results of σ and the chase results of Σ) is not affected.

Considering legal canonical source instances only. Recall that, in Lemma 3.18, we have shown that it suffices to inspect *canonical* source instances I rather than all source instances over a given source schema. Analogously, we can show that the following equivalence holds: Every instance I over source schema $\mathbf{S}$ satisfying Σ_s and Σ also satisfies σ if and only if every *legal canonical* source instance over source schema $\mathbf{S}$ satisfying Σ also satisfies σ. Clearly, we can drop the condition that Σ_s must be satisfied, because every legal canonical source instance is guaranteed to do so. The extension of the proof of Lemma 3.18 to the setting with source egds is straightforward.

Considering only k-patterns up to some fixed k. This is by virtue of Lemma 5.10.

Definition of a decision procedure. To sum up, we can define a decision procedure Implies^s for the implication problem of nested tgds in the presence of source egds by modifying the algorithm Implies from Section 3 in two ways: (1) rather than defining $k = v_\sigma \cdot w_\Sigma + 1$, we set $k = m_\sigma \cdot w_\Sigma + 1$, where m_σ denotes the maximum number of nulls in the target canonical instance of any 1-pattern of σ; and (2) we use *legal* canonical instances of k-patterns in place of canonical instances. □

Theorem 5.12 separates the complexity of reasoning for nested and plain SO tgds, since as shown in Reference [12], 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 article, 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 (respectively, 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 an important open question for future work to find out whether this high complexity is inherent in these problems or more efficient algorithms can be designed. Lemma 3.22 seems to be the key to investigating this question: There, we managed to prove that any homomorphism can be extended from a k-pattern p^k to a pattern p obtained by an arbitrary sequence of k^+-increments provided that k is chosen *sufficiently big*. It remains to be analyzed if such an extension of homomorphisms can also be guaranteed for smaller patterns.

Another direction for future research is concerned with structural characterizations of schema mappings along the lines of Reference [28]. In this article, 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 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 Reference [2]) also remains an interesting open problem.

APPENDICES

A PROOFS FOR SECTION 4

We first complete the proof of Theorem 4.10:

THEOREM 4.10 *The class of nested GLAV mappings has effective threshold for f-block size.*

PROOF. In the main body of the text, we have already presented the overall proof, which was based on the following three claims:

CLAIM 1. *The f-block size of* $core(chase(I_p, \sigma))$ *is at least* m.

CLAIM 2. *For each* $i \in \{1, 2\}$, *there exists an atom* A_i *generated by* T_i *such that* $A_i \in B$ *and* A_i *contains a local constant of* T_i. *(Recall that in a canonical target instance, local constants do not occur outside a subinstance generated by a subtree of a chase tree.)*

CLAIM 3. *The instance* $core(chase(I', \sigma))$ *has f-block size* $\geq m + 1$.

It only remains to prove these claims.

Proof of Claim 1. Let B be an f-block of size m of $core(chase(I, \sigma))$ and suppose that the facts in B are generated by RC-chase tree T_p with RC-pattern p. Moreover, let I_p be the canonical source instance of p. Recall from the proof of Lemma 3.18 that there is a one-to-one correspondence $\theta^{[\text{tr}]}$ between the triggerings in T_p and those in the canonical chase tree (i.e., the chase tree obtained from p assuming that variables are instantiated to fresh, pairwise-distinct constants). Clearly, this correspondence also carries over to chase trees with restricted conclusions. Moreover, there is a c-homomorphism θ from I_p to I, which is also a c-homomorphism from J_p (the canonical target instance of the RC-pattern p) to B. Specifically, for every triggering $(\sigma_i, \vec{a}_0, \vec{a})$ in T_p and triggering $(\sigma_i, \vec{a}'_0, \vec{a}')$ in the canonical chase tree such that $\theta^{[\text{tr}]}((\sigma_i, \vec{a}'_0, \vec{a}')) = (\sigma_i, \vec{a}_0, \vec{a})$, mapping θ maps the elements of $\vec{a}'$ and $\vec{a}'_0$ to the respective elements of $\vec{a}$ and $\vec{a}_0$.

Recall also the mapping $\theta^{[\text{null}]}$ on nulls of J_p, which sends each null $\hat{f}(\vec{a}'_0 \cdot \vec{a}') \in nulls((\sigma_i, \vec{a}'_0, \vec{a}'))$ to a distinct null $\hat{f}(\vec{a}_0 \cdot \vec{a}) \in nulls((\sigma_i, \vec{a}_0, \vec{a}))$. In other words, $\theta^{[\text{null}]} : N_p \to N$ is a bijection between N_p and N, where we write N_p and N to denote the sets of nulls in J_p and in B, respectively. Mapping $\theta^{[\text{null}]}$ is bijective even in the case when θ is not injective. As in the proof of Lemma 3.18, we write $\theta^{\overline{[\text{null}]}}$ to denote the inverse of $\theta^{[\text{null}]}$.

We will now show that J_p is connected in (the Gaifman graph of facts of) $chase(I_p, \sigma)$ and that J_p persists (up to variable renaming) in $core(chase(I_p, \sigma))$. By $J_p = \theta^{-1}(B)$, we have $|J_p| \geq m$. In total, we may thus conclude that $core(chase(I_p, \sigma))$ has f-block size at least m. To see that J_p is indeed connected, we observe that the c-homomorphism θ from J_p to B coincides with $\theta^{[\text{null}]}$ when restricted to the nulls in J_p and B, respectively. By the bijectivity of $\theta^{[\text{null}]}$, $\theta^{[\text{null}]}$ (and, hence, also $\theta^{\overline{[\text{null}]}}$) is a variable renaming. But then the connectedness of B implies the connectedness of $J_p = \theta^{-1}(B)$.

It remains to show that J_p persists (up to variable renaming) in $core(chase(I_p, \sigma))$. To this end, it suffices to show that every homomorphism h from J_p to $chase(I_p, \sigma)$ is a variable renaming. Suppose to the contrary that there exists a homomorphism h from J_p to $chase(I_p, \sigma)$ that either maps some null to a constant or maps two distinct nulls to the same null. From h, we construct the homomorphism e from B to $chase(I, \sigma)$ as in the proof of Lemma 3.18, namely:

$$e(s) = \begin{cases} s, & \text{if } s \text{ is a constant} \\ \theta^{[\text{null}]}(h(\theta^{\overline{[\text{null}]}}(s))), & \text{if } s \text{ is a null.} \end{cases}$$

As in the proof of Lemma 3.18, it can be shown that e is indeed a homomorphism from B to $chase(I, \sigma)$. But then, since we are assuming that h is *not* a variable renaming, we conclude that e is not a variable renaming either. This means that B can be further reduced in $core(chase(I, \sigma))$, which contradicts our original assumption that B is an f-block in $core(chase(I, \sigma))$. ◁

Proof of Claim 2. In the remainder of the proof, we assume w.l.o.g. that I is the canonical source instance of p. Let B be an f-block in $core(chase(I, \sigma))$ of size m and let T_1 and T_2 be two isomorphic subtrees in the RC-pattern p, such that the roots of T_1 and T_2 are siblings. By the isomorphism between T_1 and T_2, there exists a one-to-one correspondence between the facts generated by T_1 and the facts generated by T_2. Moreover, there exists a c-homomorphism ρ between the set of facts generated by T_1 and the set of facts generated by T_2. Since we are dealing with the canonical source instance, ρ is in fact bijective.

We have to show that, for each $i \in \{1, 2\}$, there exists an atom A_i generated by T_i such that $A_i \in B$ and A_i contains a local constant of T_i. Suppose to the contrary that in the facts generated by one of the T_i's, the local constants only occur (if at all) as arguments of Skolem terms in these facts. By the existence of the c-homomorphism ρ, which is in fact a c-isomorphism, it follows that

then none of the facts generated by either T_1 or T_2 contains a local constant as argument. But then ρ is in fact a homomorphism (leaving constants unchanged) and not just a c-homomorphism.

We now define the endomorphism ρ' on B such that $\rho'(s) = \rho(s)$ if s is a null $\hat{f}(\vec{a})$ such that a local constant generated by T_1 occurs in $\vec{a}$, and $\rho'(s) = s$ otherwise. Clearly, ρ' is a homomorphism (and, hence, an endomorphism), because it maps every atom generated by T_1 to an atom generated by T_2 and maps all other atoms to themselves. But the existence of such a proper endomorphism ρ' contradicts the assumption that B is an f-block in $core(chase(I, \sigma))$. ◁

Proof of Claim 3. We partition the atoms in B into three subsets $\mathcal{A}, \mathcal{B}, \mathcal{C}$, such that $\mathcal{B}$ contains the atoms generated by T_1, $\mathcal{C}$ contains the atoms generated by T_2, and $\mathcal{A}$ contains the remaining atoms of B. Let $\vec{b}$ denote the local constants of T_1 and let $\vec{c}$ denote the local constants of T_2. Then ρ from the proof of Claim 2 maps each constant b_i in $\vec{b}$ to the corresponding constant c_i in $\vec{c}$. Let $\vec{a}$ denote the local constants of the atoms generated by any ancestor triggering of the roots of T_1 and of T_2. Then the atoms generated by T_1 (respectively, T_2) are of the form $R[\vec{a}, \vec{b}]$ (respectively, $R[\vec{a}, \vec{c}]$), i.e.: the arguments in these facts are either constants in $\vec{a}$ and $\vec{b}$ (respectively, in $\vec{a}$ and $\vec{c}$) or nulls resulting from instantiating the arguments of Skolem terms by constants in $\vec{a}$ and $\vec{b}$ (respectively, in $\vec{a}$ and $\vec{c}$). In particular, by Claim 2, there indeed exists an atom A_1 (respectively, A_2) of the form $R[\vec{a}, \vec{b}]$ (respectively, $R[\vec{a}, \vec{c}]$), such that at least one constant b_i in $\vec{b}$ (respectively, c_i in $\vec{c}$) occurs as argument in this atom.

We now introduce a new vector $\vec{d}$ of fresh, pairwise-distinct constants, such that $\vec{d}$ has the same size as $\vec{b}$ and $\vec{c}$. Moreover, we define a mapping θ that maps every b_i in $\vec{b}$ to the corresponding constant d_i in $\vec{d}$. Let $\theta(I)$ be the instance obtained from I by applying θ to the constants occurring in the facts of I. Moreover, we set $I' = I \cup \theta(I)$. We claim that I' has the desired property according to Claim 3, namely: $core(chase(I', \sigma))$ has f-block size $\geq m + 1$.

Before we prove this claim, we make some observations and introduce some more terminology. Let $J = chase(I, \sigma)$ and $J' = chase(I', \sigma)$. We observe that θ can be extended to a c-homomorphism from J to J' by applying θ also to the constants used to instantiate the arguments of Skolem terms. By slight abuse of notation, we also use θ to refer to this c-homomorphism. Indeed, let F be a fact of J. We have to show that $\theta(F) \in J'$. Let σ_i be the nested tgd part whose firing in the chase of I generates F and let I_F be the facts from I that trigger the firing of σ_i and all its ancestor parts. Let $I'_F = \theta(I_F)$. It is clear that $I'_F \subseteq I'$ holds by construction of I'. But then σ_i also fires on I'_F generating $\theta(F)$. Thus, $\theta(F) \in J'$, which was to be shown.

Moreover, we can define a c-homomorphism $\bar{\theta}$ also in the opposite direction, i.e., from J' to J. To this end, we define $\bar{\theta}$ as follows:

$$\bar{\theta}(s) = \begin{cases} s, & \text{if } s \in const(J) \\ \theta^{-1}(s), & \text{if } s \in const(J') \setminus const(J) \\ \hat{f}(\theta^{-1}(\vec{e})), & \text{if } s \text{ is a null of the form } \hat{f}(\vec{e}). \end{cases}$$

In the above definition, we write $\theta^{-1}(\vec{e})$ to denote the application of θ^{-1} to every component of vector $\vec{e}$. To show that $\bar{\theta}$ is a c-homomorphism, let F' be a fact in J'. We have to show that $\bar{\theta}(F') \in J$. Let σ_i be the nested tgd part whose firing generates F' in the chase of I' with σ. Let $I'_{F'}$ be the facts from I' that trigger the firing of σ_i and of all its ancestor parts. Let $I_F = \theta^{-1}(I'_{F'})$. It is clear that $I_F \subseteq I$ by construction of I' as $I' = I \cup I\theta$. But then σ_i also triggers on I_F in the chase of I with σ, and this triggering generates $\bar{\theta}(F')$. Thus, $\bar{\theta}(F') \in J$, which was to be shown.

Now let us define $\mathcal{D} := \theta(\mathcal{B})$. By the above observations, we clearly have $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D} \subseteq J'$. We now prove Claim 3 by showing several properties of $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$.

Property 1. $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is connected in J'.

Proof of Property 1. By assumption, $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ is an f-block and, hence, connected. It follows from Lemma 3.14 that the connection of $\mathcal{B}$ and $\mathcal{C}$ is realized via nulls of the form $\hat{f}(\vec{a})$ occurring both in the atoms of $\mathcal{B}$ and of $\mathcal{C}$, where $\vec{a}$ is a vector of constants used in the variable instantiations of the triggerings at the joint ancestors of the roots of T_1 and T_2. But then all these nulls of the form $\hat{f}(\vec{a})$ are also generated in $\mathcal{D}$ in the chase of I'. Hence, the set $\mathcal{D}$ of atoms is connected to $\mathcal{B}$ and $\mathcal{C}$.

Property 2. Every endomorphism h' on J' is a variable renaming on $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$, i.e., h' maps no null in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ to a constant and h' maps no two nulls in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ to the same null.

Proof of Property 2. For the proof of both Properties 2 and 3, we make use of the following observation: Consider an arbitrary endomorphism h on J. Then h is a variable renaming on the f-block B. Indeed, let g be the endomorphism on J that reduces J to the core, i.e., $C = core(chase(I, \sigma)) = g(J)$. Now suppose that h is not a variable renaming on B, i.e., it either maps some null in B to a constant or it maps two nulls in B to the same null. We can consider the composition $g(h(\cdot))$ as a homomorphism from B to J. Clearly, $g(h(\cdot))$ is not a variable renaming on B either. By definition of f-blocks, the nulls in B form a connected component. Hence, we can define an endomorphism e on C by setting $e(s) = g(h(s))$ for all nulls s in B and $e(s) = s$ otherwise. Since h either maps some null in B to a constant or it maps two nulls in B to the same null, so does e. But then e further reduces B, which contradicts the assumption that B is an f-block in $core(chase(I, \sigma))$.

We now prove Property 2 indirectly: Suppose that there exists an endomorphism h' on J' that is not a variable renaming on $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. By restricting h', we can consider h' as a homomorphism from $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ to J'. Moreover, we can compose h' with $\bar{\theta}$ from above to get a c-homomorphism $\bar{\theta}(h'(\cdot))$ from $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ to J. To complete the proof, it suffices to show that $\bar{\theta}(h'(\cdot))$ is in fact a homomorphism, i.e., it is the identity on constants. It is easy to verify that this is the case: Let s be a constant occurring in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. By $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \subseteq J$, we have $s \in const(J)$. By the definition of homomorphisms, we have $h'(s) = s$. Hence, by the definition of $\bar{\theta}$, we also have $\bar{\theta}(h'(s)) = s$.

Since h' is not a variable renaming on $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$, the homomorphism $\bar{\theta}(h'(\cdot))$ is not a variable renaming either. Since $B = \mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ is an f-block, $\bar{\theta}(h'(\cdot))$ can be extended to an endomorphism h on J, which again is not a variable renaming on $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. This contradicts our above observation that such an endomorphism cannot exist.

Property 3. There is no endomorphism on J' that maps a labeled null in $\mathcal{D}$ to a constant.

Proof of Property 3. Suppose to the contrary that there exists an endomorphism h' on J' that maps a labeled null in $\mathcal{D}$ to a constant. We define a mapping $h = \bar{\theta}(h'(\theta(\cdot)))$. Clearly, h is a c-homomorphism from J to J. By restricting h, we can consider h also as a c-homomorphism from $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ to J. We claim that h restricted to $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ is in fact a homomorphism. To show this, consider an arbitrary constant s occurring in some fact of $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. We have to show that $h(s) = s$ holds. We distinguish two cases as to whether s is equal to some b_i from the local constants of $\mathcal{B}$ or not. First consider the case $s = b_i$ for some i. Then $\theta(b_i) = d_i$. Since h' is a homomorphism, we have $h'(d_i) = d_i$. Finally, by the definition of $\bar{\theta}$, we have $\bar{\theta}(d_i) = b_i$. In total, we thus have $\bar{\theta}(h'(\theta(b_i))) = b_i$ as desired. Now consider the case $s \in const(J) \setminus lcst(\mathcal{B})$. In this case, we have $\theta(s) = s$. Since h' is a homomorphism, we also have $h'(s) = s$. Finally, since $s \in const(J)$, we also have $\bar{\theta}(s) = s$. In total, we thus again have $\bar{\theta}(h'(\theta(s))) = s$ as desired.

By assumption, h' maps some null in $\mathcal{D}$ to a constant. By definition, θ maps each null in $\mathcal{B}$ to a null in $\mathcal{D}$. Hence, $h'(\theta(\cdot))$ maps some null in $\mathcal{B}$ to a constant, and so does $h = \bar{\theta}(h'(\theta(\cdot)))$. However, by the above considerations, this contradicts the assumption that $B = \mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$ is an f-block in $core(chase(I, \sigma))$. ◁

We are now ready to complete the proof of Claim 3: More specifically, we show that $core(chase(I', \sigma))$ has f-block size at least $m + 1$. Let h' be the endomorphism on J' that reduces J' to $core(chase(I', \sigma))$. By Claim 1, we know that $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is connected. By Claim 2 and 3, no null in $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$ is mapped to a constant by h'. Hence, also $h'(\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D})$ is connected. By Claim 2, h' is a variable renaming on $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$. Hence, $h'(\mathcal{A} \cup \mathcal{B} \cup \mathcal{C})$ has the same size as $\mathcal{A} \cup \mathcal{B} \cup \mathcal{C}$, namely, m. Finally, we also know that $h'(\mathcal{D})$ contains at least one fact $h'(R[\vec{a}, \vec{d}]) \notin h'(\mathcal{A} \cup \mathcal{B} \cup \mathcal{C})$ such that $h'(R[\vec{a}, \vec{d}])$ is connected to $h'(\mathcal{A} \cup \mathcal{B} \cup \mathcal{C})$. Thus, altogether, $h'(\mathcal{A} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D})$ is connected and has size at least $m + 1$ in $core(chase(I', \sigma))$. This concludes the proof of Claim 3 and, hence, also of Theorem 4.10. □

We now also provide a detailed proof of Theorem 4.14:

Theorem 4.14. *The class of nested GLAV mappings has effective bounded anchor.*

Proof. As before, by Lemma 4.5, it suffices to consider a single nested tgd σ. We have to show that there exists a recursive function a that, for a given schema mapping specified by a nested tgd σ, returns an integer $a(\sigma)$ witnessing bounded anchor for σ. Let I be a source instance and let $J \subseteq core(chase(I, \sigma))$ be connected. We have to show that there exists a source instance I' with $|I'| \leq a(\sigma)|J|$ and connected $J' \subseteq core(chase(I', \sigma))$ with $|J'| \geq |J|$.

By Corollary 3.15, we know that the facts in the connected subinstance J of $core(chase(I, \sigma))$ must be generated by a single chase tree in the chase forest $\mathcal{F}_{\sigma,I}$, i.e., J is generated by some chase tree T_p with pattern p. Let I_p be the canonical source instance of p. As in the proof of Theorem 4.10, it can be shown that also $core(chase(I_p, \sigma))$ contains a connected subinstance J^* of size at least $|J|$. W.l.o.g., we may assume that J^* is a *maximal* connected subinstance of $core(chase(I_p, \sigma))$. Again by Corollary 3.15, J^* is actually a subinstance of the canonical target instance J_p of pattern p.

Analogously to Lemma 3.22, let $k = v_\sigma w_\sigma + 1$, where v_σ is the number of existentially quantified variables (i.e., distinct Skolem functions) and w_σ is the number of universally quantified variables in σ. Then, we define $a(\sigma)$ as the maximum size of the canonical source instance of any k-pattern of σ.

If $|I_p| \leq a(\sigma) \leq a(\sigma)|J|$, then we set $I' = I_p$ and $J' = J^*$ and we are done. Otherwise, p cannot be a k-pattern. By Lemma 3.17, there exists a k-pattern p_0 with $p_0 \leq_k p$, i.e., pattern p is obtained from p_0 by a sequence of k^+-increments. More precisely, there exists a sequence of patterns $p_0, p_1, p_2, \ldots, p_n$ with $p_n = p$, such that each p_i is obtained from p_{i-1} by a adding a further clone of some subtree t of p_{i-1}, such that p_{i-1} already contains at least k clones of t and for every subtree t' of t, p_{i-1} contains at most k clones.

For $i \in \{0, \ldots, n-1\}$, let J_{p_i} denote the canonical target instance of p_i and define $J^*_{p_i} = J^* \cap J_{p_i}$. By the connectedness of J^*, also $J^*_{p_{n-1}}$ must be a connected subinstance of $J_{p_{n-1}}$, since adding another clone cannot make two disconnected subinstances connected. By an easy induction argument, we conclude that $J^*_{p_i}$ must be a connected subinstance of J_{p_i} for every i.

Moreover, for every i, $J^*_{p_i}$ is (up to variable renaming) indeed a subinstance of $core(chase(I_{p_i}, \sigma))$, i.e., every homomorphism h from J_{p_i} to $core(chase(I_{p_i}, \sigma))$ is a null renaming on the nulls of $J^*_{p_i}$. The reason for this is that, otherwise, by Lemma 3.22, h can be extended to a homomorphism h_p from J_p to $core(chase(I_p, \sigma))$, which is not a null renaming on J^* either. But this contradicts the above assumption that J^* is a maximal connected subinstance of $core(chase(I_p, \sigma))$.

By the definition of $a(\sigma)$, we clearly have $|I_{p_0}| \leq a(\sigma)$. If $|J^*_{p_0}| \geq |J|$, then we are done with $I' = I_{p_0}$ and $J' = J^*_{p_0}$. Otherwise, our sequence of k^+-increment steps keeps adding clones so that eventually a pattern p_i is reached for which $|J^*_{p_i}| \geq |J|$ holds. We thus set $I' = I_{p_i}$ and $J' = J^*_{p_i}$. It remains to show that $|I_{p_i}| \leq a(\sigma)|J|$ holds. To this end, we use the property shown in Lemma 3.17, that the

sequence of k^+-increment steps adds clones in a "top-down" fashion, i.e., whenever a subtree t is added, then all subtrees t' of t have at most k clones in t. In particular, this means that every k^+-increment step adds at most $a(\sigma)$ atoms to the source instance, i.e., $|I_{p_i} \setminus I_{p_{i-1}}| \leq a(\sigma)$ for every i. Moreover, the k^+-increment steps may be arranged in arbitrary order as long as supertrees are added before any subtrees of them are added. Hence, we may arrange the k^+-increment steps in such an order that subtrees containing at least one atom from J^* are added before the subtrees that contain no such atom. In this way, it is guaranteed that at most $|J| - 1$ increment steps are needed to reach a subinstance of size at least $|J|$. In total, this proves that the desired source instance I_{p_i} with $|J^*_{p_i}| \geq |J|$ indeed satisfies the condition $|I_{p_i}| \leq a(\sigma)|J|$. □

B UNDECIDABILITY PROOFS FOR SECTION 5

In this section, we give the correctness proof for Theorem 5.1, as well as the proof for Theorem 5.2. We start with the former:

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. (Correctness). Having already shown the construction, we now proceed to showing the correctness of our reduction in two parts.

Part 1 (If TM does not halt, then σ has unbounded f-block size). Assume that TM does not halt. To prove that σ has unbounded f-block size, we show that for an arbitrary integer K, we can find a source instance I such that the core of the canonical solution for I under σ has f-block size of at least K. In particular, we show that for a computation of length n of TM, we can construct instance I such that the core of the canonical universal solution has f-block size of at least n (in actuality, at least $n^2/2$, but the exact size is of no immediate concern here). Since a non-halting Turing machine can have computations of arbitrary length, it then follows that σ has unbounded f-block size.

Let $C_0, C_1, \ldots, C_n$ denote the sequence of configurations of the run of TM with $n + 1$ configurations, where we pad blanks so that for every C_i, we represent $n + 1$ tape cells. That is, for each time instant $0 \leq x \leq n$ and tape position $0 \leq y \leq n$, we have tape symbol a_{xy} and cursor information c_{xy}. We also have for every time instant x the corresponding state q_x. Given such a run of TM, we now construct a corresponding source instance I. We first represent the tape atoms:

$$\begin{array}{llll} \mathsf{tape}(0,0,a_{00},c_{00}), & & & \\ \mathsf{tape}(1,0,a_{10},c_{10}), & \mathsf{tape}(1,1,a_{01},c_{01}), & & \\ \vdots & \vdots & \ddots & \\ \mathsf{tape}(n,0,a_{n0},c_{n0}), & \mathsf{tape}(n,1,a_{n1},c_{n1}), & \ldots & \mathsf{tape}(n,n,a_{nn},c_{nn}), \end{array}$$

We now represent the rest of the atoms contained in I:

$$\begin{array}{l} \mathsf{state}(0,q_0), \mathsf{state}(1,q_1), \ldots, \mathsf{state}(n,q_n), \\ Z(0), S(0,1), \ldots, S(n-1,n), \\ A_0(a_0), \ldots, A_k(a_k), \quad Q_0(q_0), \ldots, Q_l(q_l), \quad C_0(0), C_1(1). \end{array}$$

We now show that $J = \mathit{chase}(I, \sigma)$ has f-block size at least n. Intuitively, the structure of J can be seen in Figure 14.

Specifically, J contains the following atoms:

$$\begin{array}{l} O(\hat{f}(0,0)), \\ N(\hat{f}(0,0), \hat{f}(1,1)), \end{array}$$

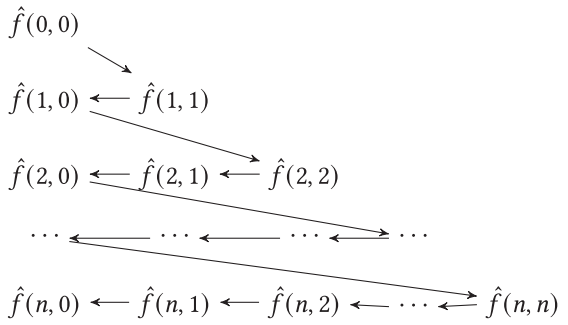

Fig. 14. Structure of a prototypical target instance.

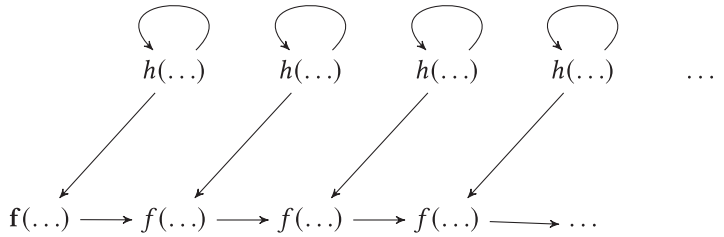

Fig. 15. Abstract representation of the structure of J. Edges denote N facts. Bold font denotes the O fact.

$$N(\hat{f}(1,1),\hat{f}(1,0)), N(\hat{f}(1,0),\hat{f}(2,2)),$$
$$N(\hat{f}(2,2),\hat{f}(2,1)), N(\hat{f}(2,1),\hat{f}(2,0)), N(\hat{f}(2,0),\hat{f}(3,3)),$$
$$\ldots$$
$$N(\hat{f}(n,n),\hat{f}(n,n-1)),\ldots,N(\hat{f}(n,1),\hat{f}(n,0)).$$

We will call these atoms J_b. Already the first N atoms in each line of J_b witness the claim of an f-block of size at least n, while the rest of each line shows that the atoms are indeed connected through a "chain" of nulls. Intuitively, these N atoms contain $\hat{f}(i,j)$ nulls, which correspond to the contents of tape cell j at time instant i. Each N atom defines a connection between two cells, such that the whole configuration is enumerated.

What we have shown so far is that the canonical solution J_b contains an f-block of size at least n. What remains to show is that J_b is also part of the core $J^* = core(J)$. First, observe that no guards fire on I. That is, apart from the atoms in J_b, J only contains the following "self-loop" for each labeled null X, $N(X,X)$, as required by the unfoundedness trap.

We now show that J is indeed a core ($J = J^*$). In Figure 15, we give an abstract illustration of the structure of J. Let us call an N-fact x the predecessor of an N-fact y, if—with N considered as a graph—it is possible to reach y from x. Towards a contradiction, assume that there exists a proper endomorphism mapping some N-fact η to another N-fact η'. Whenever this happens, also all predecessors of η must be mapped to respective predecessors of η'. However, this is not possible for the deepest predecessor, namely, the N-fact that has no predecessor: If η' precedes η, there are simply not enough predecessors of η', and if η' succeeds η, there is no fact connected to an O-fact in the chain. Also, the self-loops using the h function symbol cannot be used as the images of our proper endomorphism, since they do not start with an O-fact. That is, no proper endomorphism of J can exist. Therefore, J is indeed a core and with that, $J_b \subseteq J$ witnesses an f-block of size at least n. Thus, follows the claim.

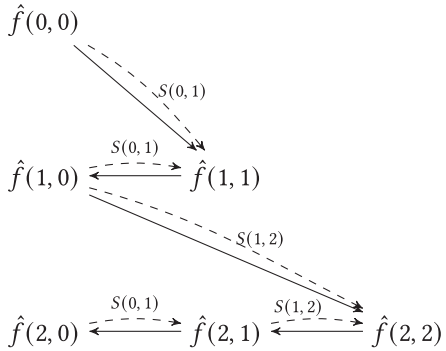

Fig. 16. Direction of N relation (solid) and S relation (dashed) edges.

Part 2 (If σ has unbounded f-block size, then TM does not halt). Assume that σ has unbounded f-block size. We then have to show that TM does not halt. We will prove this in two steps: First, we show that given that the size of the core of the universal solutions under σ is unbounded, there is an N-path of unbounded length in some core of σ (Claim A). Then, we show that if there exists an N-path of unbounded length in some core of σ, then TM has a computation of unbounded length (Claim B). We can then conclude that TM does not halt.

Let us first formalize some common constructs: We say that the size of the core under σ is *unbounded*, if there is no integer b, such that for any source instance I, the number of atoms contained in $core(chase(I, \sigma))$ is at most b. We say that there exists *an N-path of length n* in some core of σ, if there are distinct labeled nulls $x_1, x_2, \ldots, x_n$ such that the atoms $N(x_1, x_2), N(x_2, x_3), \ldots, N(x_{n-1}, x_n)$ are contained in $core(chase(I, \Sigma))$ for some source instance I. We say that there exists *an N-path of unbounded length* in the core of σ, if for any integer b there exists an N-path of length $b' \geq b$ in some core of σ.

Claim A. If the size of the core under σ is unbounded, then there exists an N-path of unbounded length in some core of σ.

We will show the following, which implies the claim: If the S-relation in the source instance I has no "join" nodes (i.e., no nodes with more than one predecessor; in this case the S-relation forms a forest, where some of the roots are possibly not marked by Z), and the N-relation of $core(chase(I, \sigma))$ has k facts, then the N-relation of $core(chase(I, \sigma))$ consists of a founded simple path of size $k/3$ and of $2k/3$ facts representing $k/3$ "traps" of the form $N(\hat{h}(x, y), \hat{h}(x, y)), N(\hat{h}(x, y), \hat{f}(x, y))$, such that each node $\hat{h}(x, y)$ occurs in exactly two "trap" atoms and $\hat{f}(x, y)$ occurs in the simple path. We call this structure a *k-gadget* and illustrate it in Figure 17. The core of the universal solutions under σ is always a k-gadget (for some k). Hence, if the k-gadget has unbounded size, so does the length of this founded simple path ($k/3$). Thus follows the original claim.

Proof of Claim A. Consider a vertex $\hat{f}(x, y)$ in the N-relation of $chase(I, \sigma)$.

Case 1: if $x = y$, then the unique predecessor of $\hat{f}(x, y)$ is $\hat{f}(x', 0)$, where x' is the unique predecessor of x in the S-relation.

Case 2: if $x \neq y$, then the unique predecessor of $\hat{f}(x, y)$ is $\hat{f}(x, y')$, where y' is a successor of y in the S-relation (see Figure 16 for an example of how the directions of the relations may look like in a concrete instance). Assume that y is reachable from an element in Z. We will later discuss the situation of this not being the case. In principle, there could be several successors of y. In our extended example in Figure 18, consider $\hat{f}(2, 1)$, which has as its two predecessors $\hat{f}(2, 2)$ and

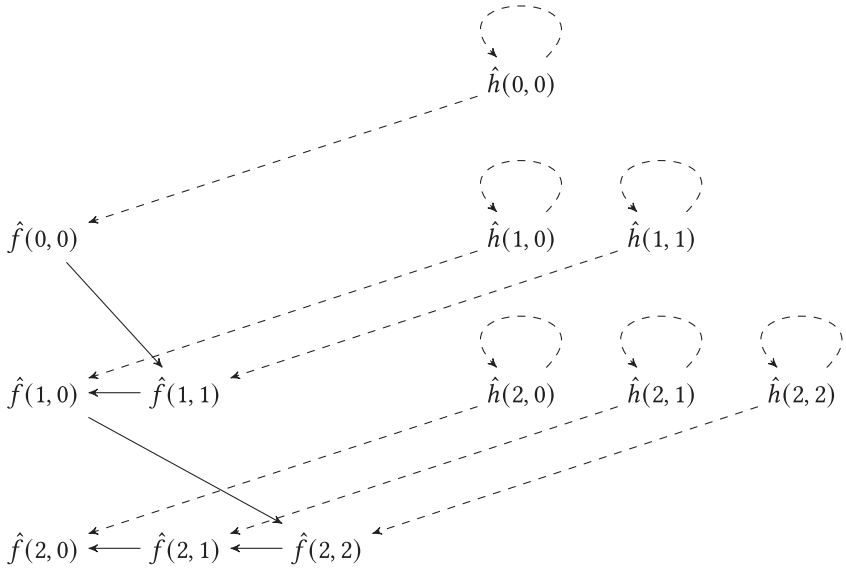

Fig. 17. k-gadget: edges denote the N relation formed of k atoms. Of these, $k/3$ form a founded simple path (solid) and $2k/3$ are not part that founded simple path (dashed).

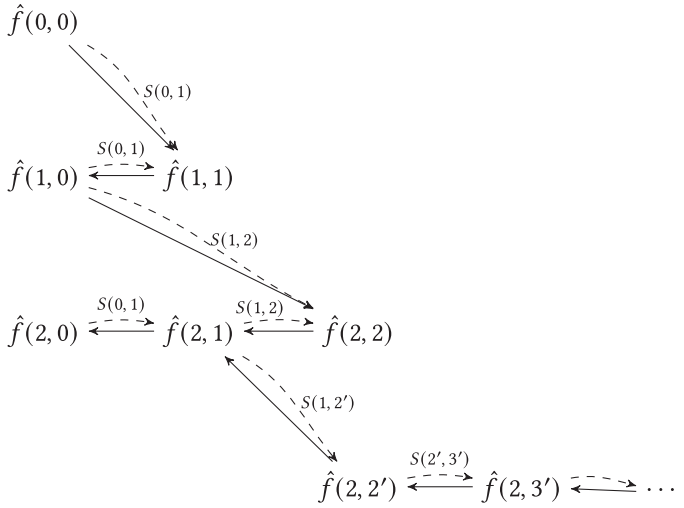

Fig. 18. The lower path splitting at $\hat{f}(2,1)$ must be unfounded. In particular, note that only the upper path has a diagonal $\hat{f}(2,2)$ after the split, while the lower one containing $\hat{f}(2,2')$ never reaches a diagonal after the split.

$\hat{f}(2,2')$. This corresponds to 1 having two successors in the S-relation, namely, 2 and $2'$. However, note that (x,y) is reached by horizontal arcs (directed from right to left) from the diagonal point (x,x). In our example, this diagonal is already one of the predecessors, i.e., $(2,2)$. Hence, we have the inequalities $y < y' < x$, where "$<$" denotes the transitive closure of the S-relation. Note that there can be only one S-successor of y from which x is reachable, since, otherwise, we would have first a "fork" (i.e., a node with two or more successors) and then a "join" (i.e., a node with two or

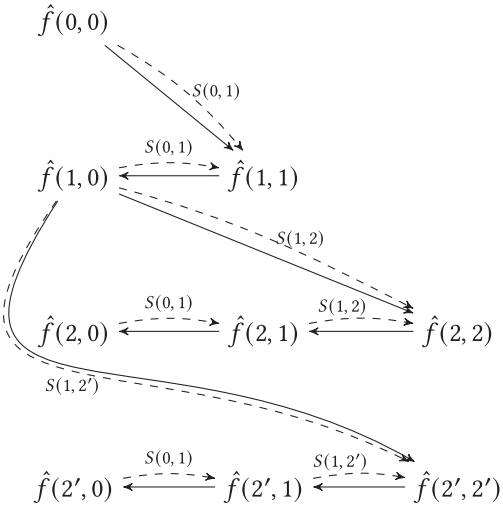

Fig. 19. Part of a canonical universal solution where a split occurs. In contrast to Figure 18, note that the direction of the N-edge at the split is in the opposite direction, and the split occurs due to the first (time) argument of $\hat{f}(\cdot,\cdot)$. Thus, we do reach a diagonal in both split paths. Since the direction of the N-paths at the split are different, this split yields a tree.

more predecessors). Consider the S-successors of y from which x is not reachable. In our example, this is $2'$. Each of these S-successors gives rise to a simple N-path leading to $\hat{f}(x,y)$, which we claim is not founded (i.e., cannot contain a node marked with O).

We now show that each node in the path connected to $\hat{f}(x,y)$ has the form $\hat{f}(x,z)$ for some z. Indeed, the only point where the first argument v in $\hat{f}(v,w)$ changes is when $v = w$ (at the diagonal). In our example, this is $\hat{f}(2,2)$. However, by our assumption, x is not reachable from y, so we can never come to a node $\hat{f}(x,x)$. In our example, we see this in the lower branch containing $\hat{f}(2,2')$, and we know that 2 (x) is not reachable from 2 (y). Moreover, it holds that $y < z$, therefore z never coincides with the least element of the $<$ order, thus we cannot reach the origin (by traversing the S relation backwards). We thus know that y' is uniquely determined by x and y and, therefore, also $\hat{f}(x,y')$ is the unique N-predecessor of $\hat{f}(x,y)$, which is reachable from an origin (see Figure 18 for an intuitive illustration of the situation discussed in this paragraph).

We thus have shown that each vertex $\hat{f}(x,y)$ in the N-relation of $chase(I,\sigma)$ has a unique predecessor reachable from an origin. Note that all predecessors not reachable from an origin cannot be part of the core, since they will be mapped to "traps." In particular, this includes the situation mentioned before in Case 2, where the assumption that y is reachable from an element if Z does not hold. See Figure 19 for an example of part of such a chase result. Therefore, $core(chase(I,\sigma))$ is a forest (plus traps connected to each non-origin node of the forest. Further note that all founded paths in the forest collapse to the longest founded path. We therefore have shown that the N-relation of $core(chase(I,\sigma))$ can be decomposed into a founded simple path and a trap for each non-origin node. ◁

Claim B. If there exists an N-path of unbounded length in some core of σ, then TM has a computation of unbounded length.

Proof Sketch of Claim B. Assume that J contains a path of length $\hat{K}$. If $\hat{K} \geq \frac{n(n+1)}{2} + 1$, then the source instance contains an encoding of a correct TM computation of length n. The formula is

derived from the size of the lower triangle of the configuration matrix including the main diagonal, plus a single O fact.

We proceed by induction on $\hat{K}$. We use as induction hypothesis that all entries of the configuration matrix corresponding to the path of length i are correct (i.e., correspond to the actual configuration of the Turing machine).

For the base case, $i = 2$, we know that no guard constraint with conclusion "fail" has fired and the guard constraints fully specify the first three configuration matrix entries (intuitively at $(0, 0)$, $(1, 0)$, and $(1, 1)$). Thus, all entries of the configuration matrix up to that point are correct.

For the induction step, assume that all entries of the configuration matrix up to i are correct. We then have to show that entry $i + 1$ is also correct. Configuration matrix entry $i + 1$ depends only on three specific entries of the configuration matrix. By construction of the enumeration, we know that the three configuration matrix entries that entry $i + 1$ depends on precede entry $i + 1$ in the enumeration. In particular, by the fact that J contains only a single path defining that enumeration (alternate paths stemming from forks or joins are prohibited by source constraints and guards as shown in Claim A), these three entries in the configuration matrix are uniquely defined. Thus, by assumption, we know that they are correct. The guard constraints fully specify the configuration matrix entry $i + 1$ by the fact that the Turing machine starts from the empty tape and thus only has a single possible run. Therefore, we know that also entry $i + 1$ is correct. □

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.*

Proof. In this proof, we use the same Turing machine simulation technique as described previously. Namely, the given TM is simulated as a plain SO-tgd σ, which, when run on an intended source instance, produces a target instance whose core is a chain of binary N-facts, founded in an O-fact. It remains to show that σ is logically equivalent to a nested GLAV mapping if and only if the simulated Turing machine halts on the empty input.

Suppose that TM does not halt. We show that in this case, the simulating plain SO-tgd σ cannot be logically equivalent to any nested GLAV mapping. To this end, we recall several facts from the literature.

First, we recall the property of SO-tgds [11], stating that the chase with this class of dependencies always terminates and delivers a universal solution. Since nested tgds are SO-tgds, this property also holds for them. We further recall the concept of *CQ-equivalence* [9]. It follows from Proposition 3.5 in Reference [9] that for mappings specified by SO-tgds (and thus also for mappings specified by nested tgds) CQ-equivalence can be defined as follows: The mappings $\mathcal{M}_1$, $\mathcal{M}_2$ are CQ-equivalent iff for arbitrary source instance I, $core(chase(I, \mathcal{M}_1)) \cong core(chase(I, \mathcal{M}_2))$ holds (where $\cong$ denotes isomorphism). By Proposition 3.6 in Reference [9], we know that logical equivalence implies CQ-equivalence and, hence, also the failure of CQ-equivalence implies the failure of logical equivalence. It thus suffices to show that for the mapping $\mathcal{M}_\sigma$ specified by the plain SO-tgd σ simulating a non-halting TM, there exists a source instance I such that for any nested GLAV mapping $\mathcal{M}$, $core(chase(I, \mathcal{M}_\sigma)) \ncong core(chase(I, \mathcal{M}))$ holds.

Let C be a class of "intended" source instances as described in the proof of Theorem 5.1, with a forest in the S relation, in which all tree roots are marked by Z-facts. Assume that C contains a family of instances $\{I_1, I_2, \ldots\}$, in which every I_n contains a tree of depth n in the S relation. As shown in the proof of Theorem 5.1, for every n, $core(chase(I_n, \sigma))$ consists of a single f-block with f-degree at most three. This is because each fact in the N-chain shares nulls with at most two other N-facts belonging to the same chain, namely, with the predecessor and the successor facts. The

initial N-fact of the chain is marked with an O-fact. Finally, every other fact of the N-chain shares a null with a single N-fact from a "self-loop trap." From the proof of Theorem 5.1 it is easy to see that the length of the N-chain and thus the f-block size of $core(chase(I_n, \sigma))$ is greater than n, and thus, we have that σ has bounded f-degree and unbounded f-block size on C.

From Theorem 4.18, we know that any nested GLAV mapping has bounded f-degree on C if and only if its f-block size on C is bounded as well. Specifically, if $\mathcal{M}$ is a nested GLAV mapping, by Claim 2 of Lemma 4.19, we know that for each integer k, there exists an integer m_k, such that for arbitrary source instance I, if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_k$, then it has f-degree $g \geq k$. Observe that, in our case, we have f-degree 3. Thus, we know that if $core(chase(I, \mathcal{M}))$ has f-block size $b \geq m_3$, then it has f-degree $g \geq 3$. Given such number m_3, it suffices to choose $n > m_3$ to ensure that f-block size of $core(chase(I_n, \mathcal{M}_\sigma))$ is greater than m_3 as well. But then, $core(chase(I_n, \mathcal{M})) \ncong core(chase(I_n, \mathcal{M}_\sigma))$ holds, since we have seen above that f-degree of $\mathcal{M}_\sigma$ is bounded by three on C, and $I_n \in C$. The failure of CQ-equivalence between $\mathcal{M}_\sigma$ and arbitrary nested GLAV mapping $\mathcal{M}$ is demonstrated, as required. As a consequence, we obtain that no nested GLAV mapping is logically equivalent to the mapping specified by σ, which simulates a non-halting TM.

Now, suppose that the TM halts. In this case, $core(chase(I, \mathcal{M}_\sigma))$ has bounded f-block size, as shown in the proof of Theorem 5.1. But then, σ is even logically equivalent to some GLAV mapping, according to Reference [28], Theorem 3.11.

Thus, we have shown that TM halts iff the mapping based on its simulating plain SO-tgd σ is logically equivalent to a nested GLAV mapping. The claim of the theorem follows from the undecidability of the halting problem. □

C PROOF OF LEMMAS 5.5 AND 5.7

Lemma 5.5. *Let K_1 and K_2 be instances and let θ be a c-homomorphism from K_1 to K_2. Let Σ be a set of egds and let K_1' denote an instance that is obtained from K_1 by a sequence S_1 of applications of egds from Σ. Then there exists a sequence S_2 of applications of egds from Σ to instance K_2, transforming K_2 into instance K_2', such that there exists a c-homomorphism θ' from K_1' to K_2'. Moreover, whenever S_1 leads to the unification of two domain elements $s_1, s_2 \in dom(K_1)$, then also S_2 leads to the unification of $\theta(s_1), \theta(s_2) \in dom(K_2)$.*

Proof. Let $K_1^0, \ldots, K_1^m$ be the sequence of instances corresponding to the chase sequence S_1 where m is the length of S_1: $K_1^0 = K_1$ and $K_1^m = K_1'$. For each $i \geq m$, K_1^i is transformed into K_1^{i+1} by a unification of two constants, which we denote $a^i, b^i \in dom(K_1^i)$ such that $K_1^{i+1} = K_1^i[a^i \mapsto b^i]$.

We now recursively define a sequence of c-homomorphisms translating the instances produced by the chase of K_1 into instances produced in the chase of K_2. We define $\theta^0 = \theta$ and $\theta^{i+1} = [\theta^i(a^i) \mapsto \theta^i(b^i)] \circ \theta^i$. Finally, we obtain a sequence (K_2^i) of instances corresponding to the chase of K_2. We set $K_2 = K_2^0$ and $K_2^{i+1} = K_2^i[\theta^i(a^i) \mapsto \theta^i(b^i)]$, for $0 < i < m$. In total, we have

(1) a sequence of instances $(K_1^0 = K_1), \ldots, (K_1^m = K_1')$,
(2) a sequence of constant pairs $(a^0, b^0), \ldots, (a^{m-1}, b^{m-1})$ such that $K_1^{i+1} = K_1^i[a^i \mapsto b^i]$,
(3) a sequence of c-homomorphisms $(\theta^0 = \theta), \ldots, \theta^{i+1} = \theta^i \circ [\theta^i(a^i) \mapsto \theta^i(b^i)]$,
(4) a sequence of instances $(K_2^0 = K_2), \ldots, K_2^{i+1} = K_2^i[\theta^i(a^i) \mapsto \theta^i(b^i)]$.

By induction on i, we show that the instances $K_2^0, \ldots, K_2^i$ make up a beginning of a chase sequence of K_2 with Σ, in which the same egds as in S_1 are applied. (Note that it may happen that the equality $\theta^i(a^i) = \theta^i(b^i)$ already holds in K_2^i, in which case the substitution $[\theta^i(a^i) \mapsto \theta^i(b^i)]$ is void and $K_2^{i+1} = K_2^i$). Furthermore, we show that θ^i is indeed a c-homomorphism from K_1^i to K_2^i.

The base case $i = 0$ is by the assumption of the lemma that $\theta = \theta^0$ is a c-homomorphism from $K_1 = K_1^0$ to $K_2 = K_2^0$, the corresponding chase sequence is empty.

Now assume that the claim holds for the first i egd applications in S_1, namely, that θ^i is a c-homomorphism from K_1^i to K_2^i and that K_2^i can be obtained from K_2 via i egd applications, where the same egds as in the sequence S_1 trigger. At step $i + 1$, an egd $\tau = \forall\vec{x}(\phi(\vec{x}) \rightarrow x_r = x_q)$ triggers on K_1^{i+1} with an assignment $\vec{d}$ for $\vec{x}$, in which the rth element is a^i and the qth element is b^i yielding the instance $K_1^{i+1} = K_1^i[a^i \mapsto b^i]$. Since $K_2^i = \theta^i(K_1^i)$ and $\phi(\vec{x})$ does not contain constants, $K_2^i \models \phi(\theta^i(\vec{d}))$ and thus the same egd τ applied to K_2^i yields the instance $K_2^{i+1} = K_2^i[\theta^i(a^i) \mapsto \theta^i(b^i)] = \theta^{i+1}(K_1^i) = \theta^{i+1}(K_1^{i+1})$, as desired.

At every step the substitution $a^i \mapsto b^i$ in S_1 corresponds to the substitution $\theta^i(a^i) \mapsto \theta^i(b^i)$ and that $\theta^i = \theta \circ \epsilon_i$ holds, where ϵ_i mimics the constant unifications in the chase sequence S_2, i.e., $\epsilon_i = [\theta^{i-1}(a^{i-1}) \mapsto \theta^{i-1}(b^{i-1})] \circ \cdots \circ [\theta^0(a^0) \mapsto \theta^0(b^0)]$. The desired c-homomorphism θ' from K_1' to K_2' is obtained as $\theta^m = \theta \circ \epsilon_m$, where ϵ_m mimics all unifications enforced by the sequence S_2 of applications of egds from Σ to K_2.

Moreover, applying the substitution $\theta^i(a^i) \mapsto \theta^i(b^i)$ also enforces the equality $\theta(a^i) = \theta(b^i)$. Ultimately, the sequence S_2 of applications of egds from Σ to K_2 results in the instance $K_2' = K_2^m$, in which the equalities $\theta(a^i) = \theta(b^i)$ have been enforced for every $i \in \{0 \ldots m-1\}$. Note that S_1 may enforce further equalities by transitivity, e.g.: suppose that some b^i and b^j with $i \neq j$ are the same element from $dom(K_1)$. Then S_1 also enforces the unification $a_i = a_j$. But then the analogous unifications (e.g., $\theta(a_i) = \theta(a_j)$ in our example) are of course also enforced by transitivity of the equalities due to sequence S_2. □

Lemma 5.7. *Let σ be a nested tgd and let Σ_s be a set of source egds. Let p be a k-pattern of σ with $k \geq 2$ and consider two isomorphic (complete) subtrees T_1 and T_2 in p such that the roots of T_1 and T_2 are siblings. Let I_p denote the canonical source instance of p and let $\vec{b}$ (respectively, $\vec{c}$) denote the local constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p. Let I_p^s denote the legal, canonical source instance of p and let $\vec{b}'$ (respectively, $\vec{c}\,'$) denote the pseudolocal constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p^s. We write b_i' (respectively, c_i') to refer to the ith component of $\vec{b}'$ (respectively, $\vec{c}\,'$). Then $\vec{b}'$ and $\vec{c}\,'$ fulfill the following properties:*

(1) for all i, j, if $b_i' = b_j'$ in I_p^s, then also $c_i' = c_j'$ holds;
(2) for all i, j, if $b_i' = c_j'$ in I_p^s, then also $c_i' = c_j'$ holds.

Proof. Consider the canonical source instance I_p of p and let $\vec{b}$ (respectively, $\vec{c}$) denote the local constants used in the variable instantiations in T_1 (respectively, in T_2) to construct I_p. We write b_i (respectively, c_i) to refer to the components of $\vec{b}$ (respectively, $\vec{c}$). Moreover, we write $\mathcal{B}$ (respectively, C) to denote the facts in I_p obtained from instantiating the antecedents of the parts in T_1 (respectively, in T_2). Clearly, the facts in $\mathcal{B}$ (respectively, C) contain as arguments either local constants of $\mathcal{B}$ (respectively, C) or constants used for instantiating variables from the antecedents in parts at common ancestors of T_1 and T_2. Recall from Section 3 that there is a one-to-one correspondence between the parts in T_1 and T_2. In particular, there exists a c-isomorphism θ between $\mathcal{B}$ and C, such that $\theta(b_i) = c_i$ for every i. Moreover, θ can be extended to a c-homomorphism from I_p to $I_p \setminus \mathcal{B}$ by setting $\theta(s) = s$ for all $s \notin \vec{b}$.

We now prove Property (1). Suppose that $b_i' = b_j'$, i.e.: there is a chase sequence S_1 with the egds in Σ, which transforms I_p into I_p^s, s.t. b_i and b_j are unified. We have to show that the chase of I_p with Σ also unifies c_i and c_j. By considering the c-homomorphism θ and applying Lemma 5.5, we conclude that there exists a sequence S_2 of applications of egds from Σ to transform I_p

into some instance I^*, s.t. there exists a homomorphism θ' from I_p^s to I^*. Moreover, whenever the sequence S_1 unifies elements s_1 and s_2 from $dom(I_p)$, then S_2 unifies $\theta(s_1)$ and $\theta(s_2)$ from $dom(\theta(I_p))$. We are assuming that $b_1' = b_2'$ holds, i.e., the chase of I_p with Σ unifies b_1 and b_2. But then the chase of $\theta(I_p)$ with Σ unifies $c_1 = \theta(b_1)$ and $c_2 = \theta(b_2)$. Note that $\theta(I_p) \subseteq I_p$. Hence, the sequence of egd firings in $\theta(I_p)$ is also applicable in I_p. Hence, the unification of c_1 and c_2 must also be enforced when we construct I_p^s from I_p via the chase with Σ, i.e., $c_i' = c_j'$ indeed holds in I_p^s.

To prove Property (2), we again consider the c-homomorphism θ and apply Lemma 5.5. Note that $\theta(b_i) = c_i$ and $\theta(c_j) = c_j$. By assumption, the sequence S_1 of egd applications to I_p unifies the elements $s_1 = b_i$ and $s_2 = c_j$ from $dom(I_p)$. Hence, the sequence S_2 of egd applications in $\theta(I_p)$ unifies c_i and c_j. By $\theta(I_p) \subseteq I_p$, we conclude that the chase of I_p with Σ also unifies c_i and c_j. □

REFERENCES

[1] Marcelo Arenas, Pablo Barceló, Leonid Libkin, and Filip Murlak. 2014. *Foundations of Data Exchange*. Cambridge University Press, Cambridge, UK.

[2] Marcelo Arenas, Jorge Pérez, Juan Reutter, and Cristian Riveros. 2013. The language of plain SO-tgds: Composition, inversion, and structural properties. *J. Comput. Syst. Sci.* 79, 6 (2013), 763–784.

[3] Marcelo Arenas, Jorge Pérez, and Cristian Riveros. 2009. The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Datab. Syst.* 34, 4 (2009), 22:1–22:48.

[4] Patricia C. Arocena, Boris Glavic, and Renee J. Miller. 2013. Value invention in data exchange. In *Proceedings of the SIGMOD Conference*. ACM, 157–168.

[5] Angela Bonifati, Elaine Qing Chang, Aks V. S. Lakshmanan, Terence Ho, and Rachel Pottinger. 2005. HePToX: Marrying XML and heterogeneity in your P2P databases. In *Proceedings of the 31st International Conference on Very Large Data Bases*. 1267–1270.

[6] AnHai Doan, Alon Halevy, and Zachary Ives. 2012. 11 - XML. In *Principles of Data Integration*. Morgan Kaufmann, Boston, MA, 291–323.

[7] Ronald Fagin and Phokion G. Kolaitis. 2012. Local transformations and conjunctive-query equivalence. In *Proceedings of the Symposium on Principles of Database Systems*. 179–190.

[8] Ronald Fagin, Phokion G. Kolaitis, Rene J. Miller, and Lucian Popa. 2005. Data exchange: Semantics and query answering. *Theor. Comput. Sci.* 336, 1 (2005), 89–124.

[9] Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. 2008. Towards a theory of schema-mapping optimization. In *Proceedings of the Symposium on Principles of Database Systems*. 33–42.

[10] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. 2005. Data exchange: Getting to the core. *ACM Trans. Datab. Syst.* 30, 1 (2005), 174–210.

[11] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2005. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Datab. Syst.* 30, 4 (2005), 994–1055.

[12] Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2015. On the undecidability of the equivalence of second-order tuple generating dependencies. *Inf. Syst.* 48 (2015), 113–129.

[13] Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti, and Lucian Popa. 2006. Nested mappings: Schema mapping reloaded. In *Proceedings of the International Conference on Very Large Data Bases*. 67–78.

[14] Georg Gottlob, Reinhard Pichler, and Emanuel Sallinger. 2015. Function symbols in tuple-generating dependencies: Expressive power and computability. In *Proceedings of the Symposium on Principles of Database Systems*. 65–77.

[15] Rihan Hai and Christoph Quix. 2019. Rewriting of plain SO tgds into nested tgds. *Proc. VLDB Endow.* 12, 11 (2019), 1526–1538.

[16] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. 2003. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the World Wide Web Conference*. ACM, 556–567.

[17] Pavol Hell and Jaroslav Nešetřil. 1992. The core of a graph. *Disc. Math.* 109 (1992), 117–126.

[18] Mauricio A. Hernández, Howard Ho, Lucian Popa, Ariel Fuxman, Renée J. Miller, Takeshi Fukuda, and Paolo Papotti. 2007. Creating nested mappings with Clio. In *Proceedings of the IEEE International Conference on Data Engineering*. 1487–1488.

[19] Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt (Eds.). 2013. *Data Exchange, Integration, and Streams*. Dagstuhl Follow-Ups, Vol. 5. Schloss Dagstuhl - Leibniz-Zentrum für Informatik.

[20] Phokion G. Kolaitis, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2014. Nested dependencies: Structure and reasoning. In *Proceedings of the Symposium on Principles of Database Systems*. 176–187.

[21] Phokion G. Kolaitis, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2018. Limits of schema mappings. *Theor. Comput. Syst.* 62, 4 (2018), 899–940.

[22] Maurizio Lenzerini. 2002. Data integration: A theoretical perspective. In *Proceedings of the Symposium on Principles of Database Systems*. 233–246.
[23] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.
[24] Jayant Madhavan and Alon Y. Halevy. 2003. Composing mappings among data sources. In *Proceedings of the International Conference on Very Large Data Bases*. 572–583.
[25] Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. 2013. Relaxed notions of schema mapping equivalence revisited. *Theor. Comput. Syst.* 52, 3 (2013), 483–541.
[26] Alessandro Raffio, Daniele Braga, Stefano Ceri, Paolo Papotti, and Mauricio A. Hernández. 2008. Clip: A visual language for explicit schema mappings. In *Proceedings of the IEEE International Conference on Data Engineering*. 30–39.
[27] Emanuel Sallinger. 2013. Reasoning about schema mappings. In *Data Exchange, Information, and Streams*. Dagstuhl Follow-Ups, Vol. 5. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 97–127.
[28] Balder ten Cate and Phokion G. Kolaitis. 2009. Structural characterizations of schema-mapping languages. In *Proceedings of the International Conference on Database Theory*. 63–72.

Received December 2018; revised September 2019; accepted October 2019