Composition of Mappings Given by Embedded Dependencies

ALAN NASH
University of California, San Diego
PHILIP A. BERNSTEIN
Microsoft Research
and
SERGEY MELNIK
Microsoft Research

Composition of mappings between schemas is essential to support schema evolution, data exchange, data integration, and other data management tasks. In many applications, mappings are given by embedded dependencies. In this article, we study the issues involved in composing such mappings.

Our algorithms and results extend those of Fagin et al. [2004], who studied the composition of mappings given by several kinds of constraints. In particular, they proved that full source-to-target tuple-generating dependencies (tgds) are closed under composition, but embedded source-to-target tgds are not. They introduced a class of second-order constraints, *SO tgds*, that is closed under composition and has desirable properties for data exchange.

We study constraints that need not be source-to-target and we concentrate on obtaining (first-order) embedded dependencies. As part of this study, we also consider full dependencies and second-order constraints that arise from Skolemizing embedded dependencies. For each of the three classes of mappings that we study, we provide: (a) an algorithm that attempts to compute the composition; and (b) sufficient conditions on the input mappings which guarantee that the algorithm will succeed.

In addition, we give several negative results. In particular, we show that full and second-order dependencies that are not limited to be source-to-target are not closed under composition (for the latter, under the additional restriction that no new function symbols are introduced). Furthermore, we show that determining whether the composition can be given by these kinds of dependencies is undecidable.

Categories and Subject Descriptors: F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

The research of A. Nash was conducted in part at Microsoft Research during an internship and in part at UCSD supported by a Microsoft Research Fellowship.

Authors' addresses: A. Nash (current affiliation): IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120; email: anash@us.ibm.com; A. Nash (former affiliation): Departments of Mathematics and Computer Science, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093; email: anash@cs.ucsd.edu; P. A. Bernstein and S. Melnik, Microsoft Research, One Microsoft Way, Redmond, WA 98052; email: {philbe,melnik}@microsoft.com.

General Terms: Algorithms, Management, Theory

Additional Key Words and Phrases: Metadata management, database theory

ACM Reference Format:

Nash, A., Bernstein, P. A. and Melnik, S. 2007. Composition of mappings given by embedded dependencies. ACM Trans. Database Syst. 32, 1, Article 4 (Mar. 2007), 51 pages. DOI = 10.1145/1206049.1206053 http://doi.acm.org/10.1145/ 1206049.1206053

1. INTRODUCTION

Many data management tasks, such as data translation, information integration, and database design, require manipulation of database schemas and mappings between schemas. A *schema mapping* (or just *mapping*) describes the relationship between the data instances of two schemas. Examples of schema mappings include SQL views, XSL transformations, integration constraints on schemas [Lenzerini 2002], and GLAV assertions in peer-to-peer systems [Halevy et al. 2003]. *Mapping composition* refers to combining two mappings into a single one. If m_{12} is a mapping between schemas σ_1 and σ_2, and m_{23} is a mapping between schemas σ_2 and σ_3, then the composition $m_{12} \circ m_{23}$ of m_{12} and m_{23} is a mapping that captures the same relationship between σ_1 and σ_3 as the two mappings m_{12} and m_{23}. This article studies two problems of mapping composition for certain classes of mapping languages: under what conditions the composition of two mappings can be expressed in the same langauge as the input mappings (i.e., the composition is closed), and how to compute the composition.

One application of mapping composition is schema evolution. For example, consider a mapping m between schemas σ and v, for example, m is a view defined on σ. Suppose that σ evolves into σ' and n is the mapping between σ' and σ. The updated mapping m' between σ' and v can be obtained as $m' = n \circ m$. As another example of schema evolution, assume that $\sigma_1, \sigma_2, \sigma_3, \sigma_4$ are the versions of a schema used in successive releases of a product. The mappings m_{12} (between σ_1 and σ_2), m_{23} (between σ_2 and σ_3), and m_{34} (between σ_3 and σ_4) are used for data migration for each subsequent release. A typical, time-consuming way of migrating the data from σ_1 to σ_4 is by executing m_{12}, m_{23}, m_{34} one-by-one. Composing $m_{12} \circ m_{23} \circ m_{34}$ to get m_{14} allows us to migrate the data from σ_1 to σ_4 in a single step, using m_{14}.

Another well-known application of mapping composition is the processing of queries over views, which is done in most commercial database systems. We can look at a set of views $\mathcal{V}$ over some schema σ_1 as a mapping from σ_1 to the schema σ_2 consisting of the name of the views in $\mathcal{V}$. Similarly, we can look at a query Q over $\mathcal{V}$ as a mapping from σ_2 to a schema σ_3 with a single relation symbol consisting of the query's name. The standard technique for processing such a query Q over the set of views $\mathcal{V}$ is to compose Q with the view definitions in $\mathcal{V}$ to obtain a new query Q'. This is known as *view unfolding* or *query modification* [Stonebraker 1975] and consists of replacing references to σ_2 in the query by the definition of the corresponding view in $\mathcal{V}$, thereby yielding a query Q' over σ_1. In terms of the definition of mapping composition given before, m_{12} is the view definition and m_{23} is the query. Then, composition of m_{12} and m_{23} is a mapping m_{13} from σ_1 to σ_3 which is given by Q'. Sets of views (or queries) give functional mappings which can be easily composed by substitution (unfolding).

It is known that first-order queries are closed under composition. Thus, in view unfolding, if m_{12} and m_{23} are both first-order queries, then their composition is a first-order query. The same holds for conjunctive queries and unions of conjunctive queries.

Query composition corresponds to composition of functional mappings. In a more general setting, the mappings to be composed may be nonfunctional, and this makes the problem of composing them harder. For example, answering queries using views involves the composition of the query and the inverse of the view. But inverting a functional mapping often yields a nonfunctional mapping.

A similar scenario arises in schema evolution. If we have a query m from schema σ to schema v, and an evolution mapping n from σ to σ', then to obtain the updated mapping m' from σ' to v, we need to compose m with the inverse of n, which may not be a function.

Composition was recently studied by Madhavan and Halevy [2003] and by Fagin [2004] (see “Related Work”, to follow). We address a similar set of questions as Fagin et al., but for different mapping languages. To be precise, they focus on mappings given by tuple-generating dependencies (tgds) and by second-order constraints, both of which are restricted to be source-to-target, while we study constraints that are first-order and need not be source-to-target. All of these terms will be defined shortly. For the time being, we can think of a source-to-target tgd as expressing the inclusion of two conjunctive queries $Q_1 \subseteq Q_2$, where Q_1 uses only symbols from the source schema and Q_2 uses only symbols from the target schema.

We have several motivations for pursuing this direction, including the following:

- The ability to deploy composition mappings in existing database system products: Currently, second-order constraints are not supported by any database system. While it is possible to develop a data exchange solution that chases a source database with second-order constraints so as to populate the target database, such a solution may not be practical for performance reasons. Ultimately, second-order constraints need to be translated into first-order data transformations (e.g., SQL queries) if they are to be executed by a DBMS efficiently.
- Allow schema constraints: If they are present, composition yields mappings that may include functional or inclusion dependencies that are not source-to-target. In particular, key and foreign-key constraints are commonplace in real-world schemas. As an example, let $R \subseteq S$ denote the mapping between $\sigma_1 = \{R(A, B)\}$ and $\sigma_2 = \{S(\underline{A}, B)\}$, where A is a key of S, and let $S \subseteq T$ denote a mapping between σ_2 and $\sigma_3 = \{T(A, B)\}$. Both mappings are specified as source-to-target tgds. However, the composition mapping cannot be expressed by any set of source-to-target tgds because in addition to $R \subseteq T$, it needs to include the constraint that A is key of R.
- Support mapping scenarios that reach beyond data integration settings that focus on certain answers: Using source-to-target tgds for data exchange

turned out to be very challenging. As explained in Fagin et al. [2003b, p. 1], source-to-target tgds suffer from the inherent problem of underspecifying the relationship between the source and target. To illustrate, source-to-target tgds are not capable of expressing even the most primitive data exchange mappings that simply copy data: A source-to-target tgd that expresses the constraint $S \subseteq T$ allows constructing the target database $\{T(a, b), T(b, a)\}$ for the source database $\{S(a, b)\}$; clearly, this is undesirable. To address this issue, Fagin et al. introduced universal solutions [2003a] and cores 2003b, which can be viewed as providing special semantics for source-to-target tgds, resembling minimal model semantics for Datalog programs. In contrast, a mapping that copies data can be expressed naturally by dependencies $\{S \subseteq T, T \subseteq S\}$, one of which is source-to-target and another target-to-source.

—The ability to efficiently validate the mapping constraints for a given pair of databases: The work of Fagin et al. [2004] showed that mappings given by second-order tgds can encode three-colorability, an NP-complete problem. In other words, given a source and a target database, the problem of deciding whether they satisfy a mapping given by second-order tgds may in general require exponential time in the size of input databases.

—Obtain closure under most basic mapping operators, in particular, composition and inverse: An operator is *closed* relative to a language $\mathcal{L}$ if its results are expressible in $\mathcal{L}$ for all inputs in $\mathcal{L}$. Specifically, closure is desirable for supporting n-way composition.

One important contribution of this article is an algorithm for composing the mappings given by embedded dependencies. Upon a successful execution, the algorithm produces a mapping that is also given by embedded dependencies. The algorithm we present is the best one we are aware of. It is used as a core part of the implementation in Bernstein et al. [2006]. Nevertheless, it has some inherent limitations. First, it may fail to produce a result, even if there exists a set of embedded dependencies that expresses the composition mapping. Second, it may generate a set of dependencies that is exponentially larger than the input. We show that these difficulties are intrinsic and not an artifact of the algorithm. We address them in part by providing sufficient conditions on the input mappings which guarantee that the algorithm will succeed. Furthermore, we devote significant attention to the novel and most challenging component of our algorithm, which performs "de-Skolemization" to obtain first-order constraints from second-order constraints.

To summarize our contributions in detail, we first explain the previous contributions of Madhavan and Halevy [2003] and Fagin et al. [2004], covered in the next section.

1.1 Related Work

A first semantics for composition of mappings was proposed in the pioneering work by Madhavan and Halevy [2003]. Under their definition, m_{13} is a composition of m_{12} and m_{23} if the certain answers obtained by way of m_{13} for any query in a class of queries $\mathcal{L}$ against schema σ_3 are precisely those that can be obtained by using m_{23} and m_{12} in sequence. Notice that in their definition, composition depends on the query class $\mathcal{L}$.

Madhavan and Halevy focused on the relational case and considered mappings given by a certain class of embedded dependencies. An embedded dependency [Fagin 1977, 1982; Nicolas 1987] is a formula which says that the presence of certain tuples implies the presence of certain other tuples or certain equality conditions. In the standard formal notation, they are of the form $\forall\bar{x}(\phi(\bar{x}) \to \exists\bar{y}\,\xi(\bar{x},\bar{y}))$, where $\bar{x}$ and $\bar{y}$ are sets of variables, $\phi(\bar{x})$ is a conjunction of relation atoms, and $\xi(\bar{x},\bar{y})$ is a conjunction of relation or equality atoms that uses all the variables of $\bar{y}$. For example, $\forall x(R(x) \to \exists y S(x, y))$ is an embedded dependency, where $R(x)$ and $S(x, y)$ are relation atoms. Madhavan and Halevy's [2003] class of interest are source-to-target tuple-generating dependencies (st-tgds), which correspond to GLAV formulas. A st-tgd is an embedded dependency where ϕ refers only to relations of the source schema, ξ refers only to relations of the target schema (which is disjoint from the source schema), and there are no equality atoms in ξ (e.g., $x_i = y_j$). For example, if R is a relation symbol from the source schema and S is a relation symbol from the target schema, then $\forall x(R(x) \to \exists y S(x, y))$ is a st-tgd.

Madhavan and Halevy showed that the result of composition may be an infinite set of formulas when the query language $\mathcal{L}$ is that of conjunctive queries. The authors proposed algorithms for the cases when composition can be done. Their definition has some disadvantages. In particular, the result of composition varies, depending on the choice of query language $\mathcal{L}$. Also, the definition is asymmetric, that is, it is based on queries over σ_3 and does not consider queries over σ_1.

An alternative, language-invariant semantics for mapping composition was proposed independently by Fagin et al. [2004] and Melnik [2004, Chap. 4]. They considered mappings as binary relations on instances of schemas and defined mapping composition as a set-theoretic composition of such binary relations. This semantics makes the result of mapping composition unique and does not depend on a specific logical formalism chosen for representing mappings and queries.

Fagin et al. [2004] were the first to embark on a systematic investigation of mapping composition under these natural semantics. They presented many fundamental results; we survey only some of them here. First, they showed that *full* st-tgds are closed under composition, but that *embedded* st-tgds are not (a full st-tgd is an embedded st-tgd without existentially quantified variables, i.e., where $\bar{y}$ is empty). To obtain closure in a more general setting, they introduced *SO tgds*, a second-order extension of st-tgds which allows for existentially quantified function symbols and equalities. These arise from Skolemizing embedded st-tgds, that is, by replacing the existentially quantified variables in $\bar{y}$ by existentially quantified function symbols. They showed that SO tgds are strictly more expressive than st-tgds, and closed under composition. This makes them a suitable mapping language for data exchange and query-rewriting scenarios [Fagin et al. 2003a; Yu and Popa 2004]. Further results in Fagin et al. [2004] are that the composition of mappings given by st-tgds may give mappings undefinable in the finite-variable infinitary logic $L^{\omega}_{\infty\omega}$ and that composition of first-order mappings may give uncomputable mappings.

Mapping composition is closely related to the constraint implication problem for views (see, e.g., Klug and Price [1982] and Abiteboul et al. [1995, pp. 221–222]). Inferring view dependencies corresponds to computing the range of a functional mapping, which we show to be reducible to mapping composition.

1.2 Contributions

We extend the seminal work of Fagin et al. [2004] in two principal directions: (1) We study constraints that need not be source-to-target, and (2) we concentrate on obtaining embedded dependencies (which are first-order). Very roughly speaking, the main two challenges that we face involve recursion and de-Skolemization.

We study the composition of three related kinds of mappings:

(1) FullD-mappings (given by full dependencies)
(2) ED-mappings (given by embedded dependencies)
(3) SkED-mappings (given by second-order constraints)

and the corresponding mappings without equality: FullTGD, TGD, and SkTGD. ED-mappings subsume st-tgds, functional dependencies, and inclusion dependencies, and can express view definitions. SkED constraints subsume the SO tgds of Fagin et al. [2004], which in our terminology are source-to-target SkED constraints.

The case of most interest to us is that of ED-mappings. We show that one way to compose them is to:

(1) Skolemize the ED-mappings to get SkED-mappings;
(2) find a finite SkED axiomatization of all SkED constraints that hold for the composition; and
(3) de-Skolemize the finite SkED axiomatization to get an ED-mapping.

The first step is easy; the difficulties arise in Steps 2 and 3. In the work of Fagin et al. [2004], the source-to-target restriction simplifies Step 2. In Step 3, our goal is to obtain embedded dependencies, which requires eliminating second-order quantifiers. By contrast, the composition considered by Fagin et al. [2004] is given by second-order constraints, so in their case, Step 3 does not apply.

For SkTGD, we consider both *restricted* composition, in which we are not allowed to introduce new function symbols, and *unrestricted* composition, in which we are free to do so. Since no function symbols appear in the other kinds of constraints we consider, restricted and unrestricted composition coincide for them.

Observe that our mapping languages are capable of expressing within-schema constraints, such as inclusion and functional dependencies. In this article, we assume that schema constraints are part of each mapping which mentions the schema. So, we do not need to refer to them explicitly.

To list our contributions, we need to refer to many classes of mappings. To simplify presentation, we use the following convention. Whenever we refer to a class of constraints without equality (e.g., TGD), we imply that the result also holds for the corresponding class of constraints with equality (e.g., ED) unless otherwise stated. In contrast, whenever we refer to a class of constraints with equality, we do not imply that the result holds for the corresponding class of constraints without equality. Furthermore, our negative results do not require the use of constants and our positive results allow them. Our contributions include the following:

Negative Results:

(1) We show that FullTGD-mappings are not closed under composition and that SkTGD-mappings are not closed under restricted composition (Theorem 1).

(2) We show that the problem of determining whether the composition of two FullTGD-mappings is a FullTGD-mapping is undecidable (Theorem 2). This result carries over to TGD-mappings and to restricted composition of SkTGD-mappings.

(3) Expressing the composition of two FullTGD-mappings may require FullTGD constraints that are exponentially larger in size than the input mappings, even over fixed schemas (Example 5). This result carries over to TGD-mappings and SkTGD-mappings. We show that there are TGD-mappings which require exponentially larger expressions in TGD than in SkTGD (Theorem 8); that is, an exponential increase in size may occur independently at each of the Steps 2 and 3 of the procedure outlined earlier. We develop a novel inexpressibility mechanism that allows us to show this result (Section 7).

Positive Results:

(4) We present necessary and sufficient (but uncomputable) conditions for composition of FullTGD- and restricted composition SkTGD-mappings (Theorems 3 and 6).

(5) We present algorithms that compute the composition of FullTGD- and restricted composition SkTGD-mappings whenever they terminate (Corollary 1). These algorithms are very similar and can be seen as an extension of that in Fagin et al. [2004] to handle mappings that are not restricted to being source-to-target.

(6) We introduce exponential-time sufficient conditions for the preceding algorithms to terminate (Theorem 4).

(7) We present an algorithm to compute the composition of TGD-mappings, which consists of three steps, as outlined previously: (1) Skolemize, (2) invoke the restricted composition algorithm for SkTGD-mappings, and (3) de-Skolemize.

(8) The de-Skolemization step may fail (after all, SkTGD has more expressive power than TGD, since, as shown in Fagin et al. [2004], it can encode NP-complete problems). We show how to check in polynomial time whether it will succeed and whether its output will be exponentially larger than its input (Proposition 3).

(9) We identify exponential-time recognizable subsets of FullTGD and SkTGD that are closed under composition (restricted for the latter) and inverse, and that include source-to-target constraints as well as constraints which

express view definitions (Theorem 5). We do not provide such a subset of TGD, since the conditions on it would be very restrictive (to ensure that de-Skolemization succeeds).

In addition, we identify two different kinds of composition of SkTGD-mappings: restricted and unrestricted. In Section 8, we will see that SkTGD constraints require special semantics for the function symbols. We introduce another fragment of second-order logic, namely, ∃SOED, and we show that every finite set of source-to-target SkED constraints is equivalent, under the special semantics, to a finite set of source-to-target ∃SOED constraints under the usual semantics for ∃SO (Theorem 11).

Composition is only one of many useful operations on mappings. Bernstein et al. [Bernstein 2003; Bernstein et al. 2000] introduced a general framework called *model management*, where operators on schemas and mappings are used to simplify the development of metadata-intensive applications. The basic operators include domain, range, composition, and inverse. Further operators are discussed in Melnik et al. [2003, 2005].

(10) We show that domain, range, and composition are closely related and can be reduced to each other (Proposition 5).

This is one reason why, in this context, composition and inverse are fundamental. The latter is easy if we use symmetric restrictions on mapping languages. Thus, FullD-, ED-, and SkED-mappings have trivial inverse mappings. On the other hand, composition turns out to be very hard and is the primary subject of this article.

1.3 Outline

This article is structured as follows. In Section 2, we give a formal definition of mapping composition and specify the mapping languages that we consider. In Section 3 we present the deductive system used in our proofs. In Sections 4, 5, and 6, we study the composition of mappings given by, respectively, full, second-order, and embedded dependencies. In Section 7, we present formal tools for showing inexpressibility results for embedded dependencies. In Section 8 we examine the semantics of SkED. In Section 9 we briefly consider how some other basic operators on mappings, such as domain and range, relate to composition. Section 10 is the conclusion.

2. PRELIMINARIES

A *term* is a constant, variable, or expression $f(t_1, \dots t_m)$, where f is a function symbol of arity m and $t_1, \dots t_m$ are terms. A *signature* is a function from a set of relation symbols to positive integers which give their arities. In this work, we use the terms signature and schema synonymously. Given a relation symbol R of arity n, a *relation atom* is an expression of the form $R(t_1, \dots, t_n)$, where $t_1, \dots, t_n$ are terms.

2.1 Constraints

A *constraint* or *dependency* is a sentence (i.e., a formula with all variables bound). We denote sets of constraints with capital Greek letters and individual constraints with lowercase Greek letters. We will need to refer to the following kinds of constraint:

Name	Abbreviation	Form
Full tuple-generating dependency	FullTGD	$\forall \bar{x}(\phi(\bar{x}) \to \psi'(\bar{x}))$
Tuple-generating dependencies	TGD	$\forall \bar{x}(\phi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$
Skolemized TGD	SkTGD	$\exists \bar{f} \forall \bar{x}(\phi(\bar{x}), \chi(\bar{x}) \to \psi'(\bar{x}))$
Existential second-order TGD	∃SOTGD	$\exists \bar{R} \forall \bar{x}(\phi(\bar{x}) \to \exists \bar{y}\ \psi(\bar{x}, \bar{y}))$

where $\phi(\bar{x})$ is a conjunction of relational atoms with variables in $\bar{x}$, $\chi(\bar{x})$ is a set of equalities either between variables or between a variable and a term, $\psi'(\bar{x})$ is a conjunction of relational atoms with variables in $\bar{x}$, $\psi(\bar{x}, \bar{y})$ is a conjunction of relational atoms with variables in $\bar{x}$ and $\bar{y}$, $\bar{f}$ is a sequence of function symbols, and $\bar{R}$ is a sequence of relation symbols. We allow constants and the degenerate case where ϕ and χ are missing.

Terms are built from constants, variables, and functions in $\bar{f}$. We require that every variable in $\bar{x}$ be *safe*. A variable is safe if it appears in a relational atom in $\phi(\bar{x})$ or alone on one side of an equation in χ, where the other side is a term constructed from safe variables.

We define *full dependency* (FullD), *embedded dependency* (ED), *Skolemized embedded dependency* (SkED), and *existential second-order embedded dependency* (∃SOED) similar to, respectively, FullTGD, TGD, SkTGD, and ∃SOTGD, except that we also allow equalities on the righthand-side of →.

The *Skolemization* of a universal-existential formula is the result of applying the following replacement: For every occurrence of a first-order existentially quantified variable v, remove $\exists v$ and replace the quantified variable wherever it appears in the scope of the quantifier with a new term of the form $f(\bar{x})$, where f is a new function symbol and $\bar{x}$ are the universally quantified variables. In addition, introduce a second-order existentially quantified function symbol f just outside the scope of $\bar{x}$. For example, the Skolemization of $\forall xy(R(x, y) \to \exists z\, S(y, z))$ is $\exists f \forall xy(R(x, y) \to S(y, f(x, y)))$. The classes of constraints in SkTGD we define here are in *unnested* form, so we would rewrite this as $\exists f \forall xyz(R(x, y), z = f(x, y) \to S(y, z))$. Skolemizing TGD constraints gives SkTGD constraints (but not all SkTGD constraints correspond to Skolemized TGD constraints).

∃SOTGD constraints are obtained by adding existential second-order quantification over relations to TGD. The existential second-order quantification in SkTGD and ∃SOTGD apply to a finite *set* of constraints, not necessarily to just one (this is not easy to illustrate in the previous table). Formally, we achieve this through a single sentence, which is the conjunction of the constraints in this finite set.

Given a source signature σ_S and a target signature σ_T disjoint from σ_S, a constraint is *source-to-target (ST)* if all the relational atoms in ϕ are over σ_S or $\bar{R}$ and all relational atoms in ψ or ξ are over σ_T or $\bar{R}$. Similarly, a constraint is *target-to-target* if it is over σ_T. We call mappings given by source-to-target constraints *GLAV* mappings and mappings given by source-to-target and target-to-target constraints *GLAVT* mappings.

Given a set of constraints Σ over the signature $\sigma_1 \cup \sigma_2$, $\Sigma|_{\sigma_1}$ is the set of constraints in Σ which contain only relation symbols in σ_1.

2.2 Mappings

A schema mapping is a binary relation on instances of database schemas (an *instance* of a database schema is a database that conforms to this schema). Given a class of constraints $\mathcal{L}$, we associate to every expression of the form $(\sigma_1, \sigma_2, \Sigma_{12})$ the mapping

$$\{\langle A, B\rangle : (A, B) \models \Sigma_{12}\}.$$

Here, Σ_{12} is a finite subset of $\mathcal{L}$ over the signature $\sigma_1 \cup \sigma_2$, σ_1 is the *input* (or *source*) *signature*, σ_2 is the *output* (or *target*) *signature*, A is a database with signature σ_1, and B is a database with signature σ_2. To simplify the presentation, we require that σ_1 and σ_2 be disjoint (otherwise, we do some renaming). Moreover, (A, B) is the database with signature $\sigma_1 \cup \sigma_2$ obtained from taking all the relations in A and B together. Its domain is the union of the domains of A and B.

We say that m is *given* by expression $E = (\sigma_1, \sigma_2, \Sigma_{12})$ if the mapping associated to E is m. Furthermore, we say that m is an *$\mathcal{L}$-mapping* if m is given by an expression $(\sigma_1, \sigma_2, \Sigma_{12})$, where Σ_{12} is a finite subset of $\mathcal{L}$.

2.3 Composition and Inverse

Given two mappings m_{12} and m_{23}, the composition $m_{12} \circ m_{23}$ is the mapping

$$\{\langle A, C\rangle : \exists B(\langle A, B\rangle \in m_{12} \wedge \langle B, C\rangle \in m_{23})\}.$$

We study the following problem: Given two expressions of the aforementioned form, specified above, find an expression for the composition. In other words, we are concerned with the syntactic counterpart to the semantic operation defined earlier. We say that two $\mathcal{L}$-mappings given by the expressions $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_3, \sigma_4, \Sigma_{12})$ are *compatible* if $\sigma_2 = \sigma_3$ and $\sigma_1, \sigma_2, \sigma_4$ are pairwise disjoint. We only consider composition of compatible $\mathcal{L}$-mappings and therefore, we have only a partial composition operation on expressions. We say that $\mathcal{L}$ is *closed under composition* if the composition of any two compatible $\mathcal{L}$-mappings is an $\mathcal{L}$-mapping.

Given a mapping m_{12} between σ_1 and σ_2, we define the inverse m_{12}^{-1} of m_{12} to satisfy

$$(A, B) \in m_{12} \text{ iff } (B, A) \in m_{12}^{-1}.$$

We use a simple definition of inverse given by "swapping" the domain and range of a mapping (in contrast to the definition in Fagin [2006], which serves a different purpose). Closure under inverse is defined similarly to closure under composition.

Notice that under these definitions, it is possible to get a language which is closed under both inverse and composition by considering the set of SkTGD-mappings consisting of mappings of the following three kinds: the mapping given by the empty set of constraints (unconstrained mappings), mappings given by a set of source-to-target constraints (ST mappings), and mappings given by a set of target-to-source constraints (TS mappings). The inverse of an ST mapping is a TS mapping, and conversely. The inverse of the unconstrained mapping is the unconstrained mapping. Furthermore, the composition of ST with ST gives ST, of TS with TS gives TS, and that of any other combination gives the unconstrained mapping (these facts are not obvious, but follow from our results in subsequent sections).

3. DEDUCTIONS

In some of the following results and algorithms, we will need to refer to some specific deductive system. Here we outline its basics; the details are not essential.

We write FullTGD or SkTGD constraints augmented with constants as *rules* of the form $\phi(\bar{x}), \chi(\bar{x}) \to \psi(\bar{x})$, leaving the second-order quantifiers over functions $\exists \bar{f}$ and the first-order universal quantifiers $\forall \bar{x}$ implicit. We call $\phi(\bar{x}), \chi(\bar{x})$ the *premise* and $\psi(\bar{x})$ the *conclusion* (similarly for FullD or SkED constraints). If the premise is empty, we write only the conclusion. We call rules of the form $\psi(\bar{c})$, where $\bar{c}$ is a tuple of constants, *facts*. In most cases we will assume, without loss of generality, that our rules have a single atom in the conclusion, since every rule with k atoms in the conclusion can always be rewritten as k rules, each with a single atom in the conclusion.

Definition 1. A *deduction* from rules Σ is a sequence of rules, each obtained in one of three ways:

(1) by copying a rule from Σ,

(2) by applying expand/rename on a rule appearing earlier in the sequence, or

(3) by applying resolution on two rules appearing earlier in the sequence.

We call such rules *axiom* rules, *expand/rename* rules, and *resolution* rules, respectively. We say that a deduction *has length n* if it consists of n lines.

A rule r obtained by expand/rename from rule r' may have additional atoms in the premise, may have variables replaced (consistently and simultaneously) by arbitrary terms,[1] may have equalities of the form $v = t$ between a variable v and a term t removed whenever v does not appear elsewhere in the rule, and may have replacements in the conclusion consistent with equalities in the premise.

A rule r obtained by resolution of the two rules s and t has as premise the atoms in the premises of s and t, except for those atoms in the conclusion of s, and as conclusion the atoms in the conclusion of t.

A rule ξ is a *variant* of ξ' if ξ can be deduced from ξ' without using resolution, and conversely. Since each rule has a single atom in the conclusion, a rule r obtained by resolution from rules p, q consists of the conclusion of q and the premises in p and q that do not appear in the conclusion of p.

To illustrate the deductions introduced in Definition 1, consider the following examples: The rule $R(x, y), z = f(x, y) \to S(x, z)$ is a valid result of applying expand/rename to $R(u, v) \to S(u, f(u, v))$. The rule $R(x, y), S(y, z) \to S(x, z)$ is the result of applying resolution to rules $R(x, y) \to S(x, y)$ and $S(x, y), S(y, z) \to S(x, z)$.

We call a resolution step a σ_2-*resolution* if it involves the elimination of an atom with a relation symbol from σ_2. In the previous example, if σ_2 contains S, then we have a σ_2-resolution.

[1] We allow distinct variables to be replaced by the same term.

We annotate our deductions by numbering the rules in them in ascending order and by adding annotations to each line indicating how this line was obtained. It is enough to annotate a resolution rule with just two numbers and an expand/rename rule with a single number and a variable assignment. Axiom rules are indicated through a lack of any other annotation. A variable assignment is a list of items of the form $x := y$, where x is a variable and y is a term.

Example 1. Given $\Sigma = \{R(x, y) \to S(x, y),\ S(z,z) \to T(z,z)\}$, and $\Delta = \{R(1, 1)\}$, the following is a valid deduction from $\Sigma \cup \Delta$:

1.	$R(1, 1)$	
2.	$R(x, y) \to S(x, y)$	
3.	$R(1, 1) \to S(1, 1)$	[2] $x := 1, y := 1$
4.	$S(1, 1)$	[1, 3]
5.	$S(z,z) \to T(z,z)$	
6.	$S(1, 1) \to T(1, 1)$	[5] $z := 1$
7.	$T(1, 1)$	[4, 6]

Here rules 1, 2, and 5 are axioms, 3 and 6 are expand/rename, and 4 and 7 are resolution.

We call a sequence of (at most) two rename-only steps followed by a resolution step on the results of these steps a *rename-resolution*. In the preceding example, 4 is obtained by rename-resolution from 1 and 2, and 7 is obtained by rename-resolution from 4 and 5. A σ_2-rename-resolution is a rename-resolution where the resolution step is a σ_2-resolution.

If there is a deduction γ from a set of constraints Σ where the last line of γ contains a constraint ξ, we say that ξ *is deduced from* Σ, which we write $\Sigma \vdash \xi$, and that γ *witnesses* $\Sigma \vdash \xi$. We write $\Sigma \vdash \Sigma'$ in the case where $\Sigma \vdash \xi$ for every $\xi \in \Sigma'$. The $\mathcal{L}$*-deductive closure* of Σ is

$$\mathrm{DC}(\mathcal{L}, \Sigma) := \{\xi \in \mathcal{L} : \Sigma \vdash \xi\}.$$

We write $\mathrm{DC}(\Sigma)$ when $\mathcal{L}$ is clear from the context. We write $D \models \Sigma$ if all constraints in Σ are true in D. We write $\Sigma \models \Sigma'$ if for all instances D, $D \models \Sigma$ implies $D \models \Sigma'$. It is easy to check that if $\Sigma \vdash \Sigma'$, then also $\Sigma \models \Sigma'$; namely, the deductive system is *sound*.

We will need the following proposition.

PROPOSITION 1. *If Δ is a set of facts in $\mathcal{L}$ and $\Sigma \cup \{\phi\} \subseteq \mathcal{L}$ for $\mathcal{L} \in \{FullD, SkED\}$, then the following are equivalent* (*notice that ϕ could be a fact*)*:*

(1) $\Sigma \cup \Delta \vdash \phi$.

(2) *There is $\xi \in \mathcal{L}$ such that $\Sigma \vdash \xi$ and $\Delta, \xi \vdash \phi$ with ξ over the signature of Δ.*

PROOF. If (2) holds, then we have deductions γ' and γ'' witnessing $\Sigma \vdash \xi$ and $\Delta, \xi \vdash \phi$. Then γ obtained by appending γ'' to γ' witnesses $\Sigma \cup \Delta \vdash \phi$.

Now, assume that (1) holds and γ witnesses $\Sigma \cup \Delta \vdash \phi$. We set γ' to γ, except for the following replacements, which we make rule-by-rule from the first rule in γ to the last:

(1) If rule r is an axiom rule from Δ, remove it.
(2) If rule r is an axiom rule not from Δ, keep it as it is.
(3) If rule r is obtained by expand/rename from rule i, replace every constant c in r and in the variable assignment for r with a corresponding variable v_c (a different variable for every constant).
(4) If rule r is obtained by resolution from rules i and j and rule i was an axiom rule from Δ, replace r with a trivial expand/rename from rule j.
(5) If rule r is obtained by resolution from rules i and j and rule i was not an axiom rule from Δ, replace r with the result of applying resolution to the new rules i and j.

(See Example 1, shown previously.) Take ξ to be the last rule in γ'. Steps 4 and 5 ensure that ξ is over the signature of Δ. Then γ' is a valid deduction witnessing $\Sigma \vdash \xi$, since axioms from Δ are no longer used and since aforementioned replacements 1 through 5 ensure that rule r is correctly deduced from the previously replaced rules. Also, $\Delta, \xi \vdash \phi$ is witnessed by a deduction γ'', which consists of the following sequence:

—The axioms from Δ that were removed from γ to obtain γ';
—ξ followed by ξ' obtained from ξ by renaming each variable v_c to the corresponding constant c; and
—a sequence of resolution steps using each axiom from Δ and starting with ξ'. □

Example 2. Given the deduction γ from Example 1, γ' is:

1.		
2.	$R(x, y) \rightarrow S(x, y)$	
3.	$R(v_1, v_1) \rightarrow S(v_1, v_1)$	[2] $x := v_1,\ y := v_1$
4.	$R(v_1, v_1) \rightarrow S(v_1, v_1)$	[3]
5.	$S(z, z) \rightarrow T(z, z)$	
6.	$S(v_1, v_1) \rightarrow T(v_1, v_1)$	[5] $z := v_1$
7.	$R(v_1, v_1) \rightarrow T(v_1, v_1)$	[4, 6]

The following replacements have been made for each rule:

(1) Axiom from Δ: removed by 1.
(2) Axiom from Σ; unchanged by 2.
(3) Expand rename: 1 replaced with v_1 by 3.
(4) Resolution on 1,3: trivial expand/rename by 4.
(5) Axiom from Σ; unchanged by 2.
(6) Expand rename: 1 replaced with v_1 by 3.
(7) Resolution on 4 and 6: resolution by 5.

—*Chase*. Here, we define a modified chase procedure which is needed in the proof of several results to follow. The definition is similar, but somewhat different from the usual chase.

Definition 2. Given an instance D, the result of *chasing* D with constraints $\Sigma \subseteq$ SkED and the set of Skolem functions F, denoted chase(D, Σ, F), is the database D'' obtained from $D' := \{R_i(\bar{c}) : \Sigma \cup \Delta \cup \Phi \vdash R_i(\bar{c})\}$, where:

— $\bar{c}$ is a tuple of constants,

— $\Delta := \{R_i(\bar{c}) : D \models R_i(\bar{c})\}$ is the set of facts given by D, and

— $\Phi := \{f(\bar{c}) = a : f \in F, f(\bar{c}) = a\}$ is the set of facts given by F,

as follows. Define $c_0 \equiv c_1$ iff $\Sigma \cup \Delta \cup \Phi \vdash c_0 = c_1$. Now to obtain D'' from D', pick one constant c_0 from every equivalence class and replace every constant in this equivalence class with c_0. In other words, $D'' := D'/\equiv$. All functions in F are required to have the same domain which includes D. If they have finite range, then chase(D, Σ, F) is finite. We write chase(D, Σ) for chase(D, Σ, $\emptyset$).

This definition is a variation on the usual definition, in which the functions in F are constructed during the chase process. In particular, for any chase sequence of the regular chase in which new witnesses are introduced, we set the functions in F accordingly and then, the result of the modified chase defined before is exactly the same as the regular chase. On the other hand, we allow the function in F to "collapse witnesses," so the universality property of the regular chase does not always hold for the chase defined here. We use this definition of the chase because: (1) we need to chase with SkED-constraints, for which the functions in F are given, and (2) because it will be convenient to have a definition of the chase that is closely related to the deductive system we use.

The important property we need of this modified chase is (a similar property holds for the standard chase):

Proposition 2. chase(D, Σ, F) $\models \Sigma$.

Proof. Set $D'' :=$ chase(D, Σ, F). Let D', Δ, and Φ be as in Definition 2. Pick $\xi \in \Sigma$. Assume that ξ is $\forall \bar{x}(\phi(\bar{x}), \chi(\bar{x}, \bar{y}) \to \psi(\bar{x}, \bar{y}))$, where ϕ, χ, ψ are as in the definition of SkED. Pick any tuple of constants $\bar{c}$ and assume that $D'' \models \phi(\bar{c})$. Pick the unique tuple of constants $\bar{d}$ such that $\Phi \vdash \chi(\bar{c}, \bar{d})$. Then $\Sigma \cup \Delta \cup \Phi \vdash \phi(\bar{c}), \chi(\bar{c}, \bar{d})$. Therefore, since $\xi \in \Sigma$, we also have $\Sigma \cup \Delta \cup \Phi \vdash \psi(\bar{c}, \bar{d})$. Without loss of generality, assume that $\psi(\bar{c}, \bar{d})$ is a single relational atom or an equation. In the former case, this relational atom must be in D' and therefore also in D''. In the latter case, this equation equates two constants which are equivalent in D' and therefore equal in D''. In either case, ξ holds in D'' for $\bar{c}$. □

4. FULL DEPENDENCIES

We start by studying the composition of FullD-mappings; that is, mappings given by full dependencies. All results in this section apply to both FullD- and FullTGD-mappings. We will see that the techniques introduced to handle these cases can be extended to handle SkED- and ED-mappings. We first show that FullTGD is not closed under composition (Theorem 1) and furthermore, that determining whether the composition of two FullD-mappings is a FullD-mapping is undecidable (Theorem 2). Then we give necessary and sufficient, noncomputable conditions for the composition of two FullTGD-mappings to be a FullTGD-mapping (Theorem 3). The following algorithm computes the composition of two FullD-mappings given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$. The algorithm, when it terminates, computes the deductive closure of $\Sigma_{12} \cup \Sigma_{23}$, then restricts this deductive closure to those constraints which do not refer to σ_2. Theorem 3 to follow shows that the deductive closure so restricted gives precisely the composition.

Procedure FullD-compose(Σ_{12}, Σ_{23})

```
Set $\Sigma := \Sigma_{12} \cup \Sigma_{23}$
Repeat
  Set $\Sigma' := \emptyset$
  For every pair $\phi, \psi \in \Sigma$
      For every way in which $\phi, \psi$ can be $\sigma_2$-rename-resolved
      to yield $\xi$ and if there is no variant of $\xi$ in $\Sigma$
          set $\Sigma' := \Sigma' \cup \{\xi\}$
  Set $\Sigma := \Sigma \cup \Sigma'$
Until $\Sigma' = \emptyset$
Return $\Sigma_{13} := \Sigma|_{\sigma_{13}}$
```

FullD-compose
terminates on FullD-mappings satisfying the conditions of Theorem 4, which can be checked in exponential time (see also Corollary 1). The obstacle to composition is recursion, yet recursion is not always a problem (Example 3). We also define good-FullD, a subset of FullD, recognizable in exponential time, which is closed under composition (Theorem 5).

Theorem 1. *There are GLAVT FullTGD-mappings whose composition is not an* FO-*mapping. In particular, GLAVT FullTGD is not closed under composition.*

Proof. Consider the GLAVT FullTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, where

Σ_{12} is	$R(x, y)$	$\rightarrow$	$S(x, y)$
	$S(x, y), S(y, z)$	$\rightarrow$	$S(x, z)$
Σ_{23} is	$S(x, y)$	$\rightarrow$	$T(x, y)$

and where $\sigma_1 = \{R\}, \sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Together, these constraints say that $R \subseteq S \subseteq T$ and that S is transitively closed. The composition $m_{12} \circ m_{23}$ is the set of all pairs (R, T) such that $\mathrm{tc}(R) \subseteq T$. Intuitively, the FullTGD-constraints which express the composition are constraints of the form

$$R(x, v_1), R(v_1, v_2), \ldots, R(v_{i-1}, v_i), R(v_i, y) \rightarrow T(x, y),$$

but no finite set of them expresses $\mathrm{tc}(R) \subseteq T$. In fact, the composition $m_{12} \circ m_{23}$ is not even expressible in first-order logic (FO), since if we had an FO sentence ϕ such that

$$\langle R, T \rangle \in m_{12} \circ m_{23} \text{ iff } (R, T) \models \phi,$$

we could create an FO formula $\psi(x, y)$ obtained by replacing every occurrence of $T(u, v)$ in ϕ with $x \neq u \vee y \neq v$. Then, given a domain D with $R \subseteq D^2$, we would have $R \models \psi[a, b]$ iff $(R, D^2 - \langle a, b \rangle) \models \phi$ iff $\mathrm{tc}(R) \subseteq D^2 - \langle a, b \rangle$ iff $\langle a, b \rangle \notin \mathrm{tc}(R)$. Therefore $\forall x \forall y\, \neg\psi(x, y)$ would say that R is a connected graph, contradicting the fact that this can not be expressed in FO [Fagin 1975] (see, e.g., Example 2.3.8 in Ebbinhaus and Flum [1999]). □

THEOREM 2. *Checking whether the composition of two FullTGD-mappings is a FullTGD-mapping is undecidable (in fact, coRE-hard). Furthermore, the problem is undecidable, even when the second mapping is a single fixed source-to-target TGD.*

PROOF. We reduce Post's correspondence problem (PCP)—known to be undecidable (see, e.g., Sipser [1997])—to the problem of deciding whether $m_{12} \circ m_{23}$ is a FullTGD-mapping where m_{12} and m_{23} are FullTGD mappings. A PCP problem consists of a finite number of *tiles*, each with two strings of 0s and 1s: a top string and a bottom string. The decision problem is to determine whether a PCP problem has a solution. A *solution* is a string S of 0s and 1s such that there is a sequence of tiles for which the concatenation of the top strings and bottom strings of tiles are both equal to S. For example, the PCP problem with the two tiles 00/001 and 11/1 (the string before the slash is the top string and the string after the slash is the bottom string) has a solution: 0011, obtained by the sequence consisting of the first tile followed by the second tile.

The reduction is partially inspired by an undecidability proof by Christoph Koch (Theorem 3.1 in Koch [2002]). Given a PCP problem, we define m_{23} so that there is a solution to the PCP problem iff $m_{12} \circ m_{23}$ is not a FullTGD-mapping. The FullTGD-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, where

—$\sigma_1 = \{A, B, O, I, E\}$,
—$\sigma_2 = \{C, D, Q, J\}$, and
—$\sigma_3 = \{T\}$

with C quaternary, all other relations binary, and with Σ_{12} and Σ_{23} as described to follow. We will use A to mark the beginning and end of a path made of O and I edges, which will correspond to strings of 0s and 1s. We will use B similarly and use E as an undirected graph to detect cycles, which will correspond to reused variables.

Similarly, we will use C (which is quaternary) to mark the beginning and end of two paths, each made of Q and J edges, which will correspond to strings of 0s and 1s (we use Q and J because they resemble O and I). We write

$$\begin{aligned}
&{}^{x}0011^{y} &&\text{for} && A(x, y), O(x, u), O(u, v), I(v, w), I(w, y),\\
&{}_{x'}0011_{y'} &&\text{for} && B(x', y'), O(x', u'), O(u', v'), I(v', w'), I(w', y'), \text{ and}\\
&{}^{x}_{x'}0011^{y}_{y'} &&\text{for} && C(x, x', y, y'), Q(x, u), Q(u, v), J(v, w), J(w, y),\\
&&&&& Q(x', u'), Q(u', v'), J(v', w'), J(w', y').
\end{aligned}$$

(the intermediate variables are not specified in this notation). We will also use D as an auxiliary marker and T as our "target."

We will define constraints Σ_{12} and Σ_{23}, the former independent of the PCP instance and the latter encoding the tiles of a PCP instance such that the composition of m_{12} and m_{23} is given by the following set of full TGDs Σ_{13}:

$$A(x, x), B(x', x') \rightarrow T(x, x') \tag{1}$$

$$A(x, y), B(x', y'), E(z, z') \rightarrow T(x, x') \tag{2}$$

$$^{x}S^{y}, {}_{x}S_{y} \rightarrow T(x, x') \tag{3}$$

for every S which is a solution to the PCP problem, together with constraints (4) to (8) to follow. Any PCP problem with a solution has an infinite number of solutions (the concatenation of a finite number of solutions is a solution), so σ_{13} contains seven constrains for a PCP problem with no solution and infinitely many constraints for a PCP problem with a solution. It is clear that the set containing constraints (1), (2), and all constraints of the form

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \to T(x, x') $$

for all solutions S, except for a solution $\hat{S}$, does not imply that

$$ {}^{x}\hat{S}^{y}, {}_{x'}\hat{S}_{y'} \to T(x, x'). $$

Therefore, the composition of the mappings m_{12} and m_{23} is given by a finite set of full TGDs iff the PCP problem has no solution, as desired.

Σ_{12} contains the following full TGDs:

$$\begin{aligned}
A(x, y), B(x'y') &\to E(x, x') && (4)\\
O(x, y) &\to E(x, y) && (5)\\
I(x, y) &\to E(x, y) && (6)\\
E(y, x) &\to E(x, y) && (7)\\
E(x, y), E(y, z) &\to E(x, z) && (8)\\
A(x, y), B(x', y'), E(z, z') &\to C(x, x', x, x') && (9)\\
A(x, y), B(x', y') &\to D(x, x') && (10)\\
D(x, x'), O(x, y), O(x', y') &\to Q(x, y), Q(x', y'), D(y, y') && (11)\\
D(x, x'), I(x, y), I(x', y') &\to J(x, y), J(x', y'), D(y, y') && (12)\\
A(x, y), B(x', y'), D(y, y') &\to C(x, x', y, y') && (13)
\end{aligned}$$

The first six constraints will be used to detect repeated variables. The last four constraints are sufficient to deduce

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \to {}^{x}_{x'}S^{y}_{y'} $$

for an arbitrary string S, applying (10) once, (11) once for every 0 in S, (12) once for every 1 in S, and then (13) once. Intuitively, we copy one 0 or one 1 from the top and bottom strings at a time from σ_1 to σ_2 and move the D-marker correspondingly, also one step at-a-time.

Σ_{23} depends on the PCP instance. Given the tiles 00/001 and 11/1, it contains the following full TGDs:

$$\begin{aligned}
C(x, x', x, x') &\to T(x, x') && (14)\\
C(x, x', z, z'), Q(y, u), Q(u, z), Q(y', u'), Q(u', v'), J(v', z') &\to C(x, x', y, y') && (15)\\
C(x, x', z, z'), J(y, z), J(y', u'), J(u', z') &\to C(x, x', y, y'). && (16)
\end{aligned}$$

More generally, each of the constraints, except for the first, encodes one tile, with the top string encoded as a path through Q and J edges going from y to z and the bottom string encoded as a path through Q and J edges going from y' to z'. Using these constraints repeatedly corresponding to the tiles that make up a solution S, we obtain

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \to C(x, x', x, x'). $$

Intuitively, applying these constraints corresponds to removing one tile at-a-time from the top and bottom strings and moving the end points in the C-marker until we obtain $C(x, x', x, x')$. Finaly, we apply constraint (14) to obtain

$$^{x}S^{y}, {}_{x'}S_{y'} \to T(x, x').$$

This shows that we can deduce the constraints 3 introduced before.

We can deduce constraint (1) using constraints (10), (13), and (14) and we can deduce constraint (2) using constraints (9) and (14).

To complete the proof, we must show that all other full TGDs over $\sigma_1 \cup \sigma_3$ that can be deduced from $\Sigma_{12} \cup \Sigma_{23}$ are implied by the constraints in Σ_{13}. Clearly, any such constraint must have an atom of the form $T(x, x')$ as its conclusion and therefore, must be obtained using constraint (14) from a constraint with conclusion $C(x, x', x, x')$. Such a conclusion can only be obtained using constraints (9), (13), or the constraints in Σ_{23} encoding tiles. If obtained through constraint (9), then since A, B, and E are over σ_1, such a constraint must be implied by constraint (2). If obtained through constraint (13), then we must have $y = x$ and $y' = x'$ and we must obtain $D(x, x')$, which can only be obtained through constraint (10). Any constraint so obtained must be implied by constraint (1). We are left with only the possibility of a deduction using the constraints that correspond to tiles. It is tedious, but straightforward to verify that this leads to constraints which are implied by a constraint of the form

$$^{x}S^{y}, {}_{x'}S_{y'} \to T(x, x'),$$

where S is a solution to the PCP problem or to some constraint that has a premise of the form

$$^{x}S'^{y}, {}_{x'}S''_{y'} \to T(x, x'),$$

where S' and S'' are not necessarily equal and neither is necessarily a solution, but where at least some variable is shared among them (to verify this, we essentially reverse the process obtained to deduce constraints of the form (3), given earlier). In the latter case, however, there must be a cycle in E and therefore, by constraint (8) (transitive closure), a 1-loop in E. Therefore, any such constraint must be implied by constraint (2).

Finally, notice that we can move the constraints encoding tiles (which are over σ_2) to Σ_{12}, in which case Σ_{23} contains a single fixed source-to-target constraint. □

THEOREM 3. *If the FullD-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ with $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$ and $\sigma_{13} = \sigma_1 \cup \sigma_3$, then the following are equivalent*:

(1) *There is a finite set of constraints $\Sigma_{13} \subseteq$ FullD over the signature σ_{13} such that $m_{13} := m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_{13})$.*

(2) *There is a finite set of constraints $\Sigma_{13} \subseteq$ FullD over the signature σ_{13} such that*

$$\mathrm{DC}(\mathit{FullD}, \Sigma_{123})|_{\sigma_{13}} = \mathrm{DC}(\mathit{FullD}, \Sigma_{13}).$$

(3) *There is k such that for every ξ over σ_{13} satisfying $\Sigma_{123} \vdash \xi$, there is a deduction of ξ from Σ_{123} using (at most) k σ_2-resolutions.*[2]

[2] Notice that no bound is given on the number of σ_1- or σ_3-resolutions.

PROOF. The proof uses Lemmas 1 and 2 to follow. First, we show the equivalence of (1) and (2), then we show the equivalence of (2) and (3).

Assume that (2) holds. Then $\langle A, C\rangle \in m_{12} \cdot m_{23}$
iff $\exists B\,(A, B, C) \models \Sigma_{123}$ (by definition of $\cdot$)
iff $(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$ (by Lemma 1 to follow)
iff $(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{13})$ (since (2) holds)
iff $(A, C) \models \Sigma_{13}$. (since DC is sound)
This shows that (1) holds.

Conversely, assume that (1) holds. Then
$(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$
iff $\exists B\,(A, B, C) \models \Sigma_{123}$ (by Lemma 1 to follow)
iff $\langle A, C\rangle \in m_{12} \cdot m_{23}$ (by definition of $\cdot$)
iff $(A, C) \models \Sigma_{13}$ (since (1) holds)
iff $(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{13})$. (since DC is sound)
This shows that (2) holds.

Now, assume that (3) holds. Set Σ to the set of all constraints in $\mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$ which can be deduced using (at most) k σ_2-resolutions and no other resolutions. Clearly, every constraint in Σ can be obtained by expand/rename from a finite subset $\Sigma_{13} \subseteq \Sigma$. We show that (2) holds. Assume that there is a deduction γ witnessing $\Sigma_{123} \vdash \xi$. Since (3) holds, we can assume that γ has $k' \leq k$ σ_2-resolutions. By Lemma 2 to follow, there is a deduction γ' witnessing $\Sigma_{123} \vdash \xi$ also with k' σ_2-resolutions and with all of them occurring before any other resolutions. Break γ' into two parts: γ'_1—the initial segment of γ up to and including the last σ_2-resolution, and γ'_2—the remainder of γ'. Since the last line of γ' does not contain any symbols from σ_2 and since there are no σ_2-resolutions in γ'_2, we can assume that the only constraints from γ'_1 used in γ'_2 are those which do not contain any symbols from σ_2. Every such constraint in γ'_1 must be in Σ, by definition of Σ. Since every constraint ψ in γ'_2 does not contain any symbols from σ_2 and since $\Sigma_{123}|_{\sigma_{13}} \subseteq \Sigma_{13}$, we have $\Sigma_{13} \vdash \psi$. Therefore, $\Sigma_{13} \vdash \xi$, as desired.

Conversely, assume that (2) holds. Take k to be the total number of σ_2-resolutions needed to deduce every $\psi \in \Sigma_{13}$ from Σ_{123}. Assume that $\Sigma_{123} \vdash \xi$. Then there is a deduction γ witnessing $\Sigma_{13} \vdash \xi$. Clearly, γ has no σ_2-resolutions. From γ, we obtain γ' witnessing $\Sigma_{123} \vdash \xi$ by appending to γ a deduction of every constraint in Σ_{13} and by replacing every line where an axiom from Σ_{13} is used by a vacuous expand/rename of the line where the deduction of this axiom ends. Clearly, γ' has exactly k σ_2-resolutions, as desired. This shows that (3) holds. □

LEMMA 1. *Under the hypotheses of Theorem* 3*, the following are equivalent*:

(1) $(A, C) \models \mathrm{DC}(\mathit{FullD}, \Sigma_{123})|_{\sigma_{13}}$.
(2) $\exists B\,(A, B, C) \models \Sigma_{123}$.

PROOF. Assume $(A, B, C) \models \Sigma_{123}$ for some B. Then $(A, B, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{123})$ (by soundness) and therefore $(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$, since B is not mentioned in $\mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$.

Conversely, assume that $(A, C) \models \mathrm{DC}(\mathrm{FullD}, \Sigma_{123})|_{\sigma_{13}}$. We set

$$(A', B, C') := \mathrm{chase}((A, \emptyset, C), \Sigma_{123}).$$

If this chase terminates and $A = A'$ and $C = C'$, then we have $(A, B, C) \models \Sigma_{123}$ by Proposition 2, which implies $(A, B) \models \Sigma_{12}$ and $(B, C) \models \Sigma_{23}$, as desired.

It is clear that the chase terminates, since no new constants are introduced. Now assume, to get a contradiction, that $A \neq A'$ or $C \neq C'$. Set Δ_{AC} to the set of facts given by A and C. Then we must have

$$\Sigma_{123} \cup \Delta_{AC} \vdash R(\bar{c}),$$

where $\bar{c}$ is a tuple of constants and R is a relation in A or C not containing $\bar{c}$ or

$$\Sigma_{123} \cup \Delta_{AC} \vdash c_0 = c_1,$$

where c_0, c_1 are distinct constants in A or C.

We consider the former case; the latter is similar. We must have $(A, C) \not\models R(\bar{c})$. If $\Sigma_{123} \cup \Delta_{AC} \vdash R(\bar{c})$, then by Proposition 1, there exists $\xi \in$ FullD over σ_{13} such that $\Sigma_{123} \vdash \xi$ and $\Delta_{AC}, \xi \vdash R(\bar{c})$. Since $(A, C) \models \Delta_{AC}$ and $(A, C) \not\models R(\bar{c})$, it follows that $(A, C) \not\models \xi$, contradicting $(A, C) \models \text{DC(FullD, } \Sigma_{123})|_{\sigma_{13}}$. □

LEMMA 2. *Under the hypotheses of Theorem* 3, *if there is a deduction γ witnessing $\Sigma_{123} \vdash \xi$ with* (*at most*) *k σ_2-resolutions, then there is γ' witnessing $\Sigma_{123} \vdash \xi$ with* (*at most*) *k σ_2-resolutions and where furthermore, all σ_2-resolutions occur before all other resolutions.*

PROOF. The basic idea of the proof is to repeatedly swap the first σ_2-resolution which occurs after a non-σ_2-resolution with this resolution until all σ_2-resolutions occur first.

Assume that we have $k < m$ and a deduction $\gamma_{k,\ell}$ witnessing $\Sigma_{123} \vdash \xi$ with:

(1) exactly m σ_2-resolutions,
(2) where the first k resolutions are σ_2-resolutions, and
(3) where there are exactly ℓ non-σ_2-resolutions before the $k+1$-th σ_2-resolution or the end of the deduction.

We proceed by induction on k and ℓ. Given $\gamma_{k,\ell}$ with $\ell > 0$, we show how to obtain the deduction $\gamma_{k,\ell-1}$ satisfying 1, 2, and 3. In the case where $\ell = 0$, we simply set $\gamma_{k+1,\ell'} := \gamma_{k,0}$ picking ℓ' so that $\gamma_{k+1,\ell'}$ satisfies 1, 2, and 3, given earlier. Once we get $\gamma_{m,\ell'}$ for some ℓ', we set $\gamma' := \gamma_{m,\ell'}$ and we are done.

Consider the line s containing δ, the $(k+1)$th σ_2-resolution in $\gamma_{k,\ell}$ of, say, lines i and j containing, respectively, α and β. Consider also the line r containing λ, the ℓ-th non-σ_2-resolution in $\gamma_{k,\ell}$ of, say, lines r_1 and r_2.

Now we have to consider several cases. If $i, j < r < s$, then we can obtain $\gamma_{k,\ell-1}$ by moving line s to just before r. If lines i, j are not derived from line r, then we can obtain $\gamma_{k,\ell-1}$ by first rearranging the deduction $\gamma_{k,\ell}$ to obtain a deduction $\gamma'_{k,\ell}$ such that $i, j < r < s$, then proceeding as before.

Otherwise, either α or β has been obtained through expand/rename from line r. To simplify the presentation, we assume that both have been obtained through a single expand/rename from line r (the other cases are similar). We have $r < i, j < s$. By rearranging $\gamma_{k,\ell}$ if needed, we can assume that $i = r + 1$, $j = r + 2$ and $s = r + 3$. Since α and β can be obtained from r by expand/rename, α_1, α_2 and β_1, β_2 (intuitively, the "unresolved" parts of λ expand/renamed as α and β) can be obtained, respectively, from lines r_1, r_2 by expand/rename so that

α is the resolution of α_1, α_2 and β is the resolution of β_1, β_2. We replace the four contiguous lines r, i, j, s:

$$\begin{array}{lll}
r & \lambda & [r_1, r_2] \\
r+1 & \alpha & [r] \\
r+2 & \beta & [r] \\
r+3 & \delta & [r+1, r+2]
\end{array}$$

with the following seven lines:

$$\begin{array}{lll}
r & \alpha_1 & [r_1] \\
r+1 & \alpha_2 & [r_2] \\
r+2 & \beta_1 & [r_1] \\
r+3 & \beta_2 & [r_2] \\
r+4 & \lambda_1 & [r+1, r+2] \\
r+5 & \lambda_2 & [r, r+4] \\
r+6 & \delta & [r+5, r+3].
\end{array}$$

The important point is that line $r+4$ now contains a σ_2-resolution, since α_2 and β_1 must resolve through a relation symbol of σ_2 because α and β do. Notice that we have δ on line $r+6$, since the result of resolution on $\alpha_1, \alpha_2, \beta_1$, and β_2 is the same as the result of resolution on α and β (this is because the resolution is "associative"). □

Corollary 1. *Under the hypotheses of Theorem* 3, FullD-compose*(*Σ_{12}, Σ_{23}*), whenever it terminates, yields* Σ_{13} *such that* $m_{12} \circ m_{23}$ *is given by* $(\sigma_1, \sigma_3, \Sigma_{13})$.

The constraints in the proof of Theorem 1 fail to satisfy (3) of Theorem 3 and therefore, FullD-compose(Σ_{12}, Σ_{23}) will not terminate when Σ_{12} and Σ_{23} are as in the proof of Theorem 1 for input. In contrast, FullD-compose(Σ_{12}, Σ_{23}) will terminate on Σ_{12}, Σ_{23} from the example to follow, which does satisfy (3) of Theorem 3, so recursion is not always bad.

Example 3. Consider the FullTGD-mappings m_{12} and m_{23} given by (σ_1, σ_2, Σ_{12}) and (σ_2, σ_3, Σ_{23}), where

Σ_{12} is	$R(x, y)$	$\rightarrow$	$S(x, y)$
	$S(x, y), S(y, z)$	$\rightarrow$	$R(x, z)$
Σ_{23} is	$S(x, y)$	$\rightarrow$	$T(x, y)$

and where $\sigma_1 = \{R\}, \sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Together, these constraints say that $R \subseteq S \subseteq T$, and that R and S are transitively closed (because the constraints

$$\begin{array}{rcl}
S(x, y), S(y, z) & \rightarrow & S(x, z) \\
R(x, y), R(y, z) & \rightarrow & R(x, z)
\end{array}$$

can be deduced from Σ_{12}). The constraints

$$\begin{array}{rcl}
R(x, y), R(y, z) & \rightarrow & R(x, z) \\
R(x, y) & \rightarrow & T(x, y)
\end{array}$$

express exactly the composition $m_{12} \circ m_{23}$, and are exactly those found by FullD-compose(Σ_{12}, Σ_{23}).

The coRE-hardness from Theorem 2 implies that algorithm FullD-compose may not terminate, even when the composition is a FullD-mapping. This happens, for example, in the following case.

Example 4. Consider the FullTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, where

$$\begin{array}{lrcl}
\hline
\Sigma_{12} \text{ is} & R(x, y) & \rightarrow & S(x, y) \\
 & R(x, y), R(y, z) & \rightarrow & R(x, z) \\
 & S(x, y), S(y, z) & \rightarrow & S(x, z) \\
\Sigma_{23} \text{ is} & S(x, y) & \rightarrow & T(x, y) \\
\hline
\end{array}$$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$.
The constraints

$$\begin{array}{rcl}
R(x, y), R(y, z) & \rightarrow & R(x, z) \\
R(x, y) & \rightarrow & T(x, y)
\end{array}$$

express exactly the composition $m_{12} \circ m_{23}$, but algorithm FULLD-COMPOSE will never terminate, since it will deduce at least the infinitely many constraints it would deduce in the proof of Theorem 1. This is because Σ_{12} here includes all the constraints in Σ_{12} there.

Even if the algorithm terminates, it may produce a result which is exponential in size of the input mappings. This is unavoidable, as the following example shows.[3]

Example 5. There is a FullTGD-mapping m_{12} and a sequence of FullTGD-mappings m_{23}^k given by Σ_{12} and Σ_{23}^k over fixed signatures $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$, where R, S, and T are binary relations such that the composition $m_{12} \circ m_{23}^k$ grows exponentially in the size of Σ_{23}^k.

The mapping m_{12} is given by:

$$\begin{array}{rcl}
\hline
R(x, y), R(y, x) & \rightarrow & S(x, y) \\
R(x, y), R(x, x) & \rightarrow & S(x, y) \\
\hline
\end{array}$$

and the family of mappings m_{23}^k is given by Σ_{23}^k, which contains the single constraint

$$\begin{array}{rcl}
\hline
S(x, u_1), S(u_1, u_2), \ldots, S(u_{k-1}, y) & \rightarrow & T(x, y) \\
\hline
\end{array}$$

saying that if there is a path of length k in S, then there is an edge in T. For each atom $S(u, v)$ in the premise of this last constraint, we can substitute either $R(u, v), R(v, u)$ or $R(u, v), R(u, u)$ to obtain a constraint over $\sigma_1 \cup \sigma_3$ which gives 2^k constraints in the composition.

The following conditions are sufficient for algorithm FULLD-COMPOSE to terminate. On the other hand, Example 6, to follow, illustrates a case where these conditions are violated. Intuitively, these conditions say that there is no "nontrivial" recursion on some atom in σ_2. It would be nice to have simpler termination conditions of wide applicability, but we are not aware of any such. Items 3 and 4 guarantee this "nontriviality." If either fails, then the recursion can only proceed for a finite number of steps.

[3] This is essentially a result on query unfolding [Sagiv and Yannakakis 1980]. Lucian Popa first brought this to our attention through an example that required a varying schema.

THEOREM 4. *Under the hypotheses of Theorem* 3*, if no constraint of the form* $\phi(\bar{z}), S(\bar{y}) \rightarrow S(\bar{x})$ *can be deduced from* Σ_{123} *using only* σ_2*-rename-resolutions such that*

(1) $\phi(\bar{z})$ *is a conjunction of atoms over* σ_{123}*,*
(2) *there is no atom* $S(\bar{w})$ *in* $\phi(\bar{z})$ *where* $\bar{w}$ *contains all the variables in* $\bar{x}$*,*
(3) *there is a variable in* $\bar{x}$ *which is not in* $\bar{y}$*, and*
(4) S *is a relation symbol in* σ_2

then FULLD-COMPOSE(Σ_{12}, Σ_{23}) *terminates and therefore,* $m_{12} \circ m_{23}$ *is a FullD-mapping. Furthermore, these conditions can be verified in exponential time in the size of* $\Sigma_{12} \cup \Sigma_{23}$.

PROOF. The conditions can be checked in exponential time as follows. Run $k_{\sigma_2} - 1$ iterations of the main loop of FULLD-COMPOSE(Σ_{12}, Σ_{23}), where k_{σ_2} is the number of relation symbols in σ_2, and check whether a constraint of the form given previously is in Σ.

For the termination claim, assume the hypotheses hold. Consider any deduction γ witnessing $\Sigma_{123} \vdash \xi$, which uses only σ_2-rename-resolutions where ξ contains a σ_2 atom in the conclusion. Assume without loss of generality that all rename operations are performed first, followed by all resolution operations. Assume also that every rule in γ contains a single atom in its conclusion and that every rule is used in, at most, one resolution step.

Such a deduction can be represented as a tree T, where every node is one atom. Every nonleaf node in T is the conclusion of some rule r. The children of r are the atoms in the premise of rule r. The premise of ξ consists of all the leaves of T and its conclusion is the root of T. It is easy to check that any subtree T' of T which contains, for every node, either all its children in T or none of them, can be converted into a deduction γ' witnessing $\Sigma_{123} \vdash \xi'$, where the premise of ξ' consists of all the leaves of T' and its conclusion is the root of T'.

Since the hypothesis holds, no such subtree may contain $S(\bar{x})$ as its root and $S(\bar{y})$ as a leaf where S is a relation symbol in σ_2 and $\bar{x} \not\subseteq \bar{y}$. Therefore, any path from the leaves to the root in T containing a node $S(\bar{y})$ may only contain, at most, $2^r r!$ other nodes with S atoms, where r is the arity of S. This means that any such path must have length bounded by $k_{\sigma_2} 2^{r_{\sigma_2}} r_{\sigma_2}!$, where k_{σ_2} is the number of relation symbols in σ_2 and r_{σ_2} is the maximum arity of a relation symbols in σ_2. As a result, up to variants, there is only a finite number of conclusions of Σ_{123} obtainable through σ_2-rename-resolutions and this implies that FULLD-COMPOSE(Σ_{12}, Σ_{23}) terminates. □

Definition 3. A FullD-mapping is a good-FullD-mapping if it is given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and no constraint of the form $\phi(\bar{z}), S(\bar{y}) \rightarrow S'(\bar{x})$, where

(1) $\phi(\bar{z})$ is a conjunction of atoms over $\sigma_1 \cup \sigma_2$,
(2) there is no atom $S(\bar{w})$ in $\phi(\bar{z})$ where $\bar{w}$ contains all the variables in $\bar{x}$,
(3) there is a variable in $\bar{x}$ which is not in $\bar{y}$, and
(4) S and S' are both relation symbols in σ_1 or both in σ_2

can be deduced from Σ_{12} using only σ_1-rename-resolutions or only σ_2-rename-resolutions. We define good-FullTGD similarly.

We can check whether an FullD-mapping is a good-FullD-mapping in exponential time in the size of constraints as in the proof of Theorem 4.

Theorem 5. *Both good-FullD and good-FullTGD are closed under composition and inverse.*

Proof. Assume that two good-FullD-mappings are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$. Set $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$. Assume constraints ξ of the form

$$\phi(\bar{z}), S(\bar{y}) \to S'(\bar{x})$$

and ξ' of the form

$$\psi(\bar{v}), S'(\bar{x}) \to S''(\bar{u}).$$

Each fails at least one of the conditions 2–4 (with the appropriate subsitutions of signatures and sets of constraints for ξ') of Definition 3 with S' in σ_2. Then resolution on ξ and ξ' gives ξ'':

$$\phi(\bar{z}), \psi(\bar{v}), S(\bar{y}) \to S''(\bar{u}).$$

which also fails at least one of the conditions 2–4 as follows. If ξ' fails 4, then S'' is not in σ_2 and therefore ξ'' fails 4. If ξ' fails 2, then ξ'' fails 2 as well. Therefore, assume that ξ' fails 3. In other words, every variable in $\bar{u}$ is in $\bar{x}$. Therefore if ξ fails 2, then ξ'' also fails 2. Also, if ξ fails 3, then every variable in $\bar{x}$ is in $\bar{y}$ so ξ'' fails 3. Finally, if ξ fails 4, then S is not in σ_2 so ξ'' also fails 4.

Then it is easy to verify (by induction on the length of proofs) that no constraint ξ'' satisfying conditions 2, 3, and 4, where both S and S' are in σ_2 of Definition 3, can be deduced using only σ_2-rename-resolutions from the constraints specifying the two mappings. Therefore, the hypotheses of Theorem 4 hold so the composition exists and furthermore, satisfies the conditions of a good-FullD-mapping. A similar proof works for good-FullTGD-mappings.

The inverse of a good-FullD-mapping given by $(\sigma_1, \sigma_2, \Sigma_{12})$ is given by $(\sigma_2, \sigma_1, \Sigma_{12})$ which is also a good-FullD-mapping. The same holds for good-FullTGD-mappings. □

We examined many other subsets of FullD for closure under composition and inverse, but were unable to find more natural conditions of similarly wide applicability. Since source-to-target FullD-constraints are total and surjective, it is natural to wonder whether the set of all total and surjective FullD-mappings is closed under composition. The following example shows that it is not.

Example 6. Consider the FullTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, where

Σ_{12} is	$R(x, y)$	$\to$	$S(x, y)$
	$R(x, y), S(y, z)$	$\to$	$S(x, z)$
Σ_{23} is	$S(x, y)$	$\to$	$T(x, y)$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Here, m_{12} and m_{23} are total and surjective and their composition says that $\text{tc}(R) \subseteq T$, which, as we have seen in the proof of Theorem 1, is not expressible even in FO. □

5. SECOND-ORDER DEPENDENCIES

In order to handle existential quantifiers in an ED-mapping, we will first convert the ED constraints which specify the mapping into SkED constraints (by Skolemizing) and this will give us SkED-mappings. Therefore, in this section we focus on the composition of SkED-mappings; in the next section we consider how to convert SkED-mappings back to ED-mappings. There are two cases of composition to consider. Unrestricted composition, in which we are allowed to introduce additional existentially quantified functions in order to express the composition, and restricted composition, in which we are only allowed to use function symbols from the input mappings. In this section we concentrate on restricted composition. SkED constraints require special semantics, which we examine in Section 8. All results in this section apply to both SkED- and SkTGD-mappings (the former correspond to SO tgds, which are not restricted to being source-to-target).

Theorems 1 and 2 from the previous section show that SkTGD is not closed under restricted composition and that determining whether the restricted composition of two SkTGD-mappings is an SkTGD-mapping is undecidable. This is because in restricted composition, we are not allowed to add function symbols, so SkTGD does not add any power toward the restricted composition of FullTGD mappings. As in the case of FullD-mappings, we give necessary and sufficient, undecidable conditions for two SkED-mappings to have restricted composition (Theorem 6), and we give sufficient conditions for restricted composition that can be checked efficiently.

Theorem 6 suggests essentially the same algorithm for composition of SkED-mappings as FULLD-COMPOSE; we call this SKED-COMPOSE. The only difference between them is that SKED-COMPOSE operates on SkED constraints, while FULLD-COMPOSE operates on FullD constraints. The correctness of SKED-COMPOSE, sufficient conditions for its termination, and good-SkED-mappings are defined for SkED just like for FullD.

THEOREM 6. *If the SkED-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ with $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$ and $\sigma_{13} = \sigma_1 \cup \sigma_3$, then the following are equivalent*:

(1) *There is a finite set of constraints $\Sigma_{13} \subseteq$ SkED over the signature σ_{13} such that $m := m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_{13})$, where Σ_{13} has no function symbols or constants other than those appearing in Σ_{123}.*

(2) *There is a finite set of constraints $\Sigma_{13} \subseteq$ SkED over the signature σ_{13} such that*

$$\mathrm{DC}(\mathit{SkED}, \Sigma_{123})|_{\sigma_{13}} = \mathrm{DC}(\mathit{SkED}, \Sigma_{13}),$$

where Σ_{13} has no function symbols or constants other than those appearing in Σ_{123}.

(3) *There is k such that for every ξ over σ_{13} satisfying $\Sigma_{123} \vdash \xi$, there is a deduction of ξ from Σ_{123} using, at most, k σ_2-resolutions.*

PROOF. Essentially the same as that of Theorem 3, using Lemma 3 to follow, instead of Lemma 1. □

LEMMA 3. *Under the hypotheses of Theorem* 6, *the following are equivalent*:

(1) $(A, C) \models \mathrm{DC}(\mathit{SkED}, \Sigma_{123})|_{\sigma_{13}}$.
(2) $\exists B\, (A, B, C) \models \Sigma_{123}$.

PROOF. Assume that $(A, B, C) \models \Sigma_{123}$ for some B. Then $(A, B, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})$ (by soundness) and therefore $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$, since B is not mentioned in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$.

Conversely, assume $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. In particular, this implies the existence of all Skolem functions mentioned in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. There may be additional Skolem functions mentioned in Σ_{123}, but not in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. We give arbitrary values to these additional Skolem functions. We define F to be the set containing all these Skolem functions and Φ to be the set of all facts in F.

We set $(A', B, C') := \mathrm{chase}((A, \emptyset, C), \Sigma_{123}, F)$.

If the chase terminates and $A = A'$ and $B = B'$, then we have $(A, B, C) \models \Sigma_{123}$ by Proposition 2, which implies that $(A, B) \models \Sigma_{12}$ and $(B, C) \models \Sigma_{23}$, as desired.

The chase terminates because of the required safety condition (which is preserved by the deduction rules). Therefore, (A', B, C') is well-defined. Now, assume, so as to get a contradiction, that $A \neq A'$ or $C \neq C'$. Set Δ_{AC} to the set of facts given by A and C. Then we must have

$$\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash R(\bar{c}),$$

where $\bar{c}$ is a tuple of constants and R is a relation in A or C not containing $\bar{c}$ or

$$\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash c_0 = c_1,$$

where c_0, c_1 are distinct constants in A or C.

We consider the former case; the latter is similar. We must have $(A, C) \not\models R(\bar{c})$. If $\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash R(\bar{c})$, then by Proposition 1 there exists $\xi \in \mathrm{SkED}$ over σ_{13} such that $\Sigma_{123} \vdash \xi$ and $\Delta_{AC} \cup \Phi \cup \{\xi\} \vdash R(\bar{c})$. Since $(A, C) \models \Delta_{AC}$ and $(A, C) \not\models R(\bar{c})$, it follows that $(A, C) \not\models \xi$, contradicting $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. □

6. EMBEDDED DEPENDENCIES

Now we consider composition of ED-mappings; that is, mappings given by embedded dependencies. To compute the composition of two ED-mappings m_{12}, m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, we will proceed in three steps, as follows.

```
Procedure ED-COMPOSE($\Sigma_{12}$, $\Sigma_{23}$)
(1) $\Sigma'_{12}$ := SKOLEMIZE($\Sigma_{12}$)
    $\Sigma'_{23}$ := SKOLEMIZE($\Sigma_{23}$)
(2) $\Sigma'_{13}$ := SKED-COMPOSE($\Sigma'_{12}$, $\Sigma'_{23}$)
(3) Return DESKOLEMIZE($\Sigma'_{13}$)
```

The first step, SKOLEMIZE, is straightforward and the second step, SKED-COMPOSE, has been discussed in the previous section, so here we concentrate on the third step, de-Skolemization. We provide a sound (but not complete) algorithm for de-Skolemization: DESKOLEMIZE. Even if the second step succeeds, it may be impossible to find $\Sigma_{13} \subseteq \mathrm{ED}$ such that $\Sigma'_{13} \equiv \Sigma_{13}$ (Example 8), so we identify necessary and sufficient, polynomial-time checkable conditions for our algorithm to succeed (Proposition 3). DESKOLEMIZE may produce a result of size exponential in the size of its input; we show that in the general case, this is unavoidable (Theorem 8), but we also provide polynomial-time checkable conditions for DESKOLEMIZE to run in polynomial time in the size of its input (Proposition 3). The algorithm consists of 12 steps and is as follows;[4] we provide a detailed description of each step with examples at the end of this section.

```
Procedure DESKOLEMIZE($\Sigma$)
 (1) Unnest
 (2) Check for cycles
 (3) Check for repeated function symbols
 (4) Align variables
 (5) Eliminate restricting atoms
 (6) Eliminate restricted constraints
 (7) Check for remaining restricted constraints
 (8) Check for dependencies
 (9) Combine dependencies
(10) Remove redundant constraints
(11) Replace functions with $\exists$-variables
(12) Eliminate unnecessary $\exists$-variables
```

We prove the following correctness and efficiency results for DESKOLEMIZE at the end of this section.

THEOREM 7. *If* DESKOLEMIZE(Σ) *succeeds on input* $\Sigma \subseteq$ *SkED giving* Σ'*, then*

$$\Sigma' \subseteq \textit{ED} \text{ and } \Sigma' \equiv \Sigma.$$

PROPOSITION 3.

(1) DESKOLEMIZE(Σ) *succeeds on input* $\Sigma \subseteq$ *SkED iff it reaches Step* 9*, which can be checked in polynomial time in the size of* Σ.
(2) *Furthermore, for every constant* ℓ*,* DESKOLEMIZE *runs in polynomial time on any set of constraints* Σ *such that by the end of Step* 8*, there are no more than* ℓ *constraints containing any one function symbol* f*. This can be checked in polynomial time in the size of* Σ. □

Intuitively, DESKOLEMIZE attempts to put the constraints in its input Σ into a form where they are the obvious result of Skolemization, then it reverses this Skolemization in the obvious way. Most of the work is done in bringing the constraints to such a form.

Step 11 is where function symbols are actually replaced by existentially quantified variables. For it to work properly, constraints must be combined as done in Step 9. These two steps are, in some sense, the main steps in the algorithm. Steps 10 and 12 are just "clean-up" steps. The remaining steps 1 through 8 ensure that the constraints are in the proper form for Steps 9 and 11. Procedure DESKOLEMIZE may abort at Steps 2, 3, 4, 7, or 8. Except for Step 4, these steps only check that the constraints are in the desired form and abort

[4] These steps can also be executed in some other orders. In particular, Steps 5, 6, and 7 can be executed before Steps 3 and 4 and Steps 3 and 4 can be combined into a single step.

if they are not. Before going through the steps of DeSkolemize in detail, let us look at a simple example to give some intuition for the algorithm.

Example 7. Assume that the input to DeSkolemize consists of the single constraint

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

which is obtained from Skolemizing the TGD-constraint

$$R(x, y) \to \exists u\, S(x, u), S(u, y)$$

which says that for every edge in R, there is a path of length 2 in S. Strictly speaking, this is not an SkTGD constraint as defined in the preliminaries (since the function symbol f appears in the conclusion), but is close enough. Step 1 gives the following set of SkTGD constraints:

$$\begin{array}{rcl}\hline R(x, y), u = f(x, y) & \to & S(x, u) \\ R(x, y), u = f(x, y) & \to & S(u, y) \\ \hline\end{array}$$

We call x and y base variables and u a term variable. Steps 2 and 3 succeed and, in this case, Step 4 does nothing. However, if the constraints had been

$$\begin{array}{rcl}\hline R(x, y), u = f(x, y) & \to & S(x, u) \\ R(w, z), v = f(w, z) & \to & S(w, z) \\ \hline\end{array}$$

then Step 4 would have replaced variables w, z, v with x, y, u in the second constraint. The goal of this step is to have every equation in which a given Skolem function appears be identical. In this case, there are no equalities that restrict the values of the Skolem functions, so Steps 5 and 6 do nothing and Step 7 succeeds. Step 8 checks that every base variable which appears in a conclusion also appears as an argument to every Skolem function in the premise (we will discuss the case of nested Skolem functions later). In this case, both x and y satisfy this condition. If we were to apply Step 11 at this point, we would obtain the constraints

$$\begin{array}{rcl}\hline R(x, y) & \to & \exists u\, S(x, u) \\ R(x, y) & \to & \exists u\, S(u, y) \\ \hline\end{array}$$

which say that for every edge in R from x to y, there is an outgoing S edge from x and an incoming S edge into y. This is not quite the same as saying that there is a path of length 2 from x to y because these edges may not meet. In other words, we may have values of u witnessing that the first constraint holds which are different from those values of u which witness that the second constraint holds. This is why we first apply Step 9 (Step 4 is intended to make Step 9 possible), which gives the following constrains:

$$\begin{array}{rcl}\hline R(x, y), u = f(x, y) & \to & S(x, u) \\ R(x, y), u = f(x, y) & \to & S(u, y) \\ R(x, y), u = f(x, y) & \to & S(x, u), S(u, y) \\ \hline\end{array}$$

Now, Step 10 simplifies this to the single constraint:

$$\begin{array}{rcl}\hline R(x, y), u = f(x, y) & \to & S(x, u), S(u, y) \\ \hline\end{array}$$

since the other two easily follow from it, and Step 11 yields

$$\begin{array}{lcl} \hline R(x,y) & \rightarrow & \exists u\, S(x,u), S(u,y) \\ \hline \end{array}$$

as desired. Notice that at every step, we have constraints that are equivalent to those at the previous step.

Of course, this is a simple example; we will see soon that things can get more complicated. We abort at some steps of the algorithm if the constraints cannot be put into the desired form for subsequent steps. By doing so, we may fail to de-Skolemize some constraints which, in fact, are equivalent to embedded dependencies. However, it seems likely that deciding whether some SkED constraints are equivalent to ED constraints is undecidable, so we do not hope for a complete algorithm. The following example from Fagin et al. [2004] shows that de-Skolemization is not always possible.

Example 8. Consider the ED-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$, where

$$\begin{array}{lrcl} \hline \Sigma_{12} \text{ is} & E(x,y) & \rightarrow & F(x,y) \\ & E(x,y) & \rightarrow & \exists u\, C(x,u) \\ & E(x,y) & \rightarrow & \exists v\, C(y,v) \\ \Sigma_{23} \text{ is} & F(x,y), C(x,u), C(y,v) & \rightarrow & D(u,v) \\ \hline \end{array}$$

and where $\sigma_1 = \{E\}$, $\sigma_2 = \{F, C\}$, and $\sigma_3 = \{D\}$. Here, Steps 1 and 2 of ED-COMPOSE succeed, but Step 3 fails. In fact, no algorithm for de-Skolemization can succeed on this composition, since $m_{12} \circ m_{23}$ is not an ED-mapping, as shown in Fagin et al. [2004].

Since DESKOLEMIZE(Σ) may produce a result of size exponential in the size of Σ due to Step 9, ED-COMPOSE(Σ_{12}, Σ_{23}) may produce a result of size exponential in the size of $\Sigma_{12} \cup \Sigma_{23}$ due to de-Skolemization, even when the preceding composition steps yield a polynomial-size result. The following theorem shows that in the general case, this is unavoidable.

THEOREM 8. *There are two sequences of TGD-mappings m^k_{12} and m^k_{23} given by Σ^k_{12} and Σ^k_{23} such that the TGD-composition $m^k_{12} \circ m^k_{23}$ grows exponentially in the size of $\Sigma^k_{12} \cup \Sigma^k_{23}$, but the SkTGD-composition $m^k_{12} \circ m^k_{23}$ grows linearly in the size of $\Sigma^k_{12} \cup \Sigma^k_{23}$.*

This algorithm in fact can be applied to any set of SkED-constraints, but since we are interested in those SkED-constraints obtained from ED-mappings by SKOLEMIZE and SKED-COMPOSE, we add some observations that apply to this special case. In the special case we are interested in, procedure DESKOLEMIZE may abort only at Steps 3, 7, or 8. In particular, the following result follows from these observations.

THEOREM 9. *ED-COMPOSE generalizes view unfolding. In other words, if*

(1) *$\sigma_2 = \{V_1, \ldots, V_k\}$ and each V_i is a view given by the conjunctive query with equations $\exists \bar{y}_i \phi_i(\bar{x}_i, \bar{y}_i)$ over σ_1, where $\phi_i(\bar{x}_i, \bar{y}_i)$ is a conjunction of atoms (including equations) over $\bar{x}_i$, $\bar{y}_i$,*

(2) *$\sigma_3 = \{W_1, \ldots, W_k\}$ and each W_i is a view given by the conjunctive query with equations $\exists \bar{v}_i \psi_i(\bar{u}_i, \bar{v}_i)$ over σ_2, where $\psi_i(\bar{u}_i, \bar{v}_i)$ is a conjunction of atoms (including equations) over $\bar{u}_i$, $\bar{y}_i$,*

(3) Σ_{12} *is a functional mapping from* σ_1 *to* σ_2 *given by*

$$(\alpha_i) \quad \phi_i(\bar{x}_i, \bar{y}_i) \to V_i(\bar{x}_i)$$
$$(\beta_i) \quad V_i(\bar{x}_i) \to \exists \bar{y}_i \phi_i(\bar{x}_i, \bar{y}_i), \qquad \textit{and}$$

(4) Σ_{23} *is a functional mapping from* σ_2 *to* σ_3 *given by*

$$(\gamma_i) \quad \psi_i(\bar{u}_i, \bar{v}_i) \to W_i(\bar{u}_i)$$
$$(\delta_i) \quad W_i(\bar{u}_i) \to \exists \bar{v}_i \psi_i(\bar{u}_i, \bar{v}_i),$$

then ED-COMPOSE *correctly computes their composition*

$$\psi_i^{\bar{\phi}}(\bar{u}_i, \bar{v}_i) \to W_i(\bar{u}_i)$$
$$W_i(\bar{u}_i) \to \exists \bar{v}_i \psi_i^{\bar{\phi}}(\bar{u}_i, \bar{v}_i),$$

where $\psi_i^{\bar{\phi}}$ *denotes the result of substituting in* ψ_i *the conjunctions* $\phi_1, \ldots, \phi_k$ *for the ocurrences of* $V_1, \ldots, V_k$.

PROOF (SKETCH). Since SKED-COMPOSE does resolution only through σ_2, the following resolution patterns are possible:

(1) α_i with β_i,
(2) one or more constraints from $\{\alpha_1, \ldots, \alpha_k\}$ with γ_i,
(3) δ_i with β_j, and
(4) δ_i with γ_i.

In particular, SKED-COMPOSE terminates. It is easy (but tedious) to verify that: (a) the constraints obtained by resolution of δ_i with γ_i can be deduced from the others and that (b) the remaining constraints can be deskolemized by DESKOLEMIZE. □

The algorithm DESKOLEMIZE depends on $\vdash^*$, which is used in Steps 5 and 10. Here, $\vdash^*$ is *some* sound polynomial-time approximation of $\models$. In other words, if $\Sigma \vdash^* \phi$, then $\Sigma \models \phi$. Of course, the converse may not hold, but we require that if $\phi \in \Sigma$, then $\Sigma \vdash^* \phi$. Its use in Step 10 is nonessential; there, we simply take advantage of the fact that $\vdash^*$ is available. In Step 5 we do use it essentially; however, even if $\Sigma \vdash^* \phi$ is only true when $\phi \in \Sigma$, DESKOLEMIZE will succeed on a large class of inputs.

There are known cases in which there exists such $\vdash^*$ that is also complete, but has complexity NP. For example, when Σ has *stratified witnesses* (see Deutsch and Tannen [2003] and Fagin et al. [2003a]), then $\vdash^*$ is complete and can be computed in NP by first chasing the premise of ϕ, then looking for a homomorphism of the conclusion of ϕ into the result of this chase. (The chase needed in this case is a straightforward adaptation to the case where existential quantification is replaced by Skolem functions; the functions are treated as uninterpreted symbols.) The fact that Σ has stratified witnesses ensures that the chase terminates and that the result of chasing ϕ is polynomial in the size of ϕ. The NP complexity comes from looking for a homomorphism. A more detailed discussion for the options in the implementation of $\vdash^*$ would take us too far away from our main concerns here. For the purposes of the algorithm presented next, $\vdash^*$ can be treated as a black box. DESKOLEMIZE works for any $\vdash^*$ which is sound; the more complete $\vdash^*$ is, the larger the set of inputs on which DESKOLEMIZE succeeds.

We proceed to discuss every step in more detail. For each step, we provide a brief explanation and, where appropriate, an example.

(1) **Unnest:**
The goal of this step is to bring the constraints into a normal form so as to make it easier for subsequent steps to operate on them.
Set $\Lambda_1 := \{\psi : \phi \in \Sigma\}$, where ψ is equivalent to and obtained from ϕ by "unnesting" terms and eliminating nonvariable terms from relational atoms and from the conclusion so that in ψ:
(a) Function symbols occur only in equalities in the premise.
(b) Every term $f(\bar{x})$ occurs in only one atom.
(c) Every equation is of the form $y = z$ or of the form $u = f(\bar{x})$, where $u, \bar{x}, y$, and z are variables and f is a function symbol. We call the latter a *defining* equation for u. Furthermore, u (which we call a term variable for f) does not appear in any relational atom or on the lefthand-side of any other defining equation. We call variables which are not term variables *base* variables.
(d) The conclusion contains, at most, one atom.

(2) **Check for cycles:**
The goal of this step is to abort the computation for constraints which contain cyclic dependencies among Skolem terms. Such constraints can not be de-Skolemized.

For every $\phi \in \Lambda_1$, construct the graph G_ϕ where the edges are variables in ϕ and where there is an edge (v, u) iff there is an equation of the form $v = f(\ldots u \ldots)$. We say that variable v depends on u if there is a path in G_ϕ from v to u. If G_ϕ has a cycle, abort. Otherwise, set $\Lambda_2 := \Lambda_1$.

In the general case, a term variable may depend on other term variables, or even on itself. This step is intended to rule out the latter case. For example, this happens in the following constraint:

$$R(x, y), u = f(x, v), v = g(y, u) \to S(u, v).$$

In the special case of constraints arising in ED-COMPOSE, Lemma 4 to follow allows us to assume that term variables depend only on base variables. In other words, there are no equalities of the form $v = f(\ldots u \ldots)$ where u is a term variable. In particular, this guarantees that G_ϕ will have no cycles.

Notice that (20) can be obtained by resolution from (18) and

$$\phi(j\bar{y}), S(j\bar{y}), \bar{v} = \bar{g}(j\bar{y}) \to \tau(j\bar{y}, \bar{v}). \tag{17}$$

Here, (17) is obtained from (19) by the substitution $\bar{y} \mapsto j\bar{y}$.

(3) **Check for repeated function symbols:**
The goal of this step is to abort the computation if any constraints contain two atoms with the same function symbol. While in some special cases, it is possible to exploit some symmetries in order to de-Skolemize such constraints, we take the easy way out and give up. These are not constraints obtained directly from Skolemizing first-order constraints, since in such constraints every appearance of a Skolem function would have exactly the same arguments.

For every $\phi \in \Lambda_2$, check that ϕ does not contain two atoms with the same function symbol. If it does, abort. Otherwise, set $\Lambda_3 := \Lambda_2$. This is the step in which DeSkolemize fails on the mappings in Example 8. The Skolemized constraints from Example 8 are:

Σ_{12} is	$E(x, y)$	$\rightarrow$	$F(x, y)$
	$E(x, y), u = f(x, y)$	$\rightarrow$	$C(x, u)$
	$E(x, y), v = g(x, y)$	$\rightarrow$	$C(y, v)$
Σ_{23} is	$F(x, y), C(x, u), C(y, v)$	$\rightarrow$	$D(u, v)$

SkED-compose gives the following four constraints:

$E(x, y), E(x, w), E(y, z), u = f(x, w), v = f(y, z)$	$\rightarrow$	$D(u, v)$
$E(x, y), E(x, w), E(z, y), u = f(x, y), v = g(z, y)$	$\rightarrow$	$D(u, v)$
$E(x, y), E(w, x), E(y, z), u = g(w, x), v = f(y, z)$	$\rightarrow$	$D(u, v)$
$E(x, y), E(w, x), E(z, y), u = g(w, x), v = g(z, y)$	$\rightarrow$	$D(u, v)$

(4) **Align variables:**
The goal of this step is to get all ocurrences of a Skolem term to be the same. When we Skolemize constraints, this must be the case.

Rename the variables in Λ_3 to obtain Λ_4 satisfying:

(a) For every function symbol f and any two equalities of the form $u = f(\bar{x})$ and $v = f(\bar{y})$ in Λ_4, u is the same variable as v and $\bar{x}$ is the same sequence of variables as $\bar{y}$.

(b) For every two different function symbols f and g and any two equalities of the form $u = f(\bar{x})$ and $v = g(\bar{y})$ in Λ_4, u and v are different variables.

If this is not possible, abort. After this step, there is a unique term variable v_f associated to the function symbol f.

This steps fails, for example, on the following constraints:

$R(x, y), u = f(x, y), v = g(x, y)$	$\rightarrow$	$S(u, v)$
$R(x, y), u = f(x, y), v = g(y, x)$	$\rightarrow$	$T(u, v)$

In the special case of constraints arising in ED-compose, this step will always succeed. This follows from Lemma 4, presented next, and the following considerations. After the Skolemize steps, every function symbol appears in exactly one constraint if we allow multiple atoms in the conclusion, or equivalently, in constraints with identical premises. Clearly, we can always align variables on such constraints. Also, we can clearly align variables after an expand/rename step, so let us consider a resolution step. Assume that we have a set of constraints Σ in which the variables are aligned and that to this set Σ we add the constraint $\phi(\bar{x}), \psi(\bar{z}) \rightarrow \rho(\bar{x}, \bar{z})$, obtained by resolution from $\phi(\bar{x}) \rightarrow S(\bar{y})$ and $S(\bar{y}), \psi(\bar{z}) \rightarrow \rho(\bar{y}, \bar{z})$, both in Σ, where the variables in $\bar{y}$ are also in $\bar{x}$ and where $\phi(\bar{x})$ and $\psi(\bar{z})$ are conjunctions of atoms with variables from $\bar{x}$ and $\bar{z}$, respectively. We need to consider Skolem functions in $\phi(\bar{x})$ and in $\psi(\bar{z})$. Lemma 4 shows that those in $\psi(\bar{z})$ can be replaced with new Skolem functions, which depend only on the base variables. On the other hand, those in $\phi(\bar{x})$ have not changed and are already aligned. Therefore, the variables in Σ' consisting of Σ and the new constraint are also aligned. Since variables in the input constraints to SkED-compose can be aligned (because they have been obtained by Skolemization), it follows that the variables in the output constraints of SkED-compose can also be aligned.

(5) **Eliminate restricting atoms:**
If u is a term variable for f and u appears in any other atom, we call this atom an *f-restriction*. If ϕ has an f-restriction in the premise, we say that f restricts ϕ. If ϕ has an f-restriction in the conclusion, we say that ϕ restricts f. If there is any function f which restricts ϕ, we say that ϕ is restricted. This step as well as the next two deal with restrictions.

Set $\Lambda_5 := \{\phi' : \phi \in \Lambda_4\}$, where ϕ' is ϕ with the maximal set of restrictions removed from the premise which gives $\Lambda_4 \vdash^* \phi'$. It is easy to verify that such a maximal set always exists and is unique. Consider, for example, the constraints

ϕ_1	$R(x)$	$\to$	$\exists y\, S(x, y)$
ϕ_2	$S(x, y)$	$\to$	$U(x, y)$
ϕ_3	$S(x, x)$	$\to$	$T(x)$

where $\sigma_2 = \{S\}$. Skolemization, basic composition, and the first few steps of DeSkolemize give the following constraints Λ_4:

ψ_1	$R(x), y = f(x)$	$\to$	$U(x, y)$
ψ_2	$R(x), y = f(x), x = y$	$\to$	$T(x)$

In this case, $\Lambda_5 = \Lambda_4$. These constraints can be de-Skolemized to yield

ψ'_1	$R(x)$	$\to$	$\exists y\, U(x, y)$
ψ'_2	$\forall y(U(x, y) \to x = y)$	$\to$	$T(x)$

which, however, are not ED-constraints. On the other hand, if we add T' to σ_3 and the constraints

ϕ_4	$S(x, y)$	$\to$	$T'(x)$
ϕ_5	$T'(x)$	$\to$	$T(x)$

then Λ_5 is

ψ_1	$R(x), y = f(x)$	$\to$	$U(x, y)$
ψ_3	$R(x), y = f(x)$	$\to$	$T(x)$
ψ_4	$R(x), y = f(x)$	$\to$	$T'(x)$
ψ_5	$T'(x)$	$\to$	$T(x)$

Notice that the restriction on ψ_2 has been eliminated to give ψ_3, since ψ_4 and ψ_5 imply ψ_3. The basic intuition is that we do not know how to de-Skolemize a constraint of the form

$$\phi(\bar{x}), u = f(\bar{x}), v = g(\bar{x}), u = v \to S(\bar{y})$$

because we do not know how to express the restriction $u = v$ on the Skolem functions once these are replaced by existentially quantified variables ($\exists$-variables, for short). Part of the problem is that the restriction is in the premise, but after the replacement, the $\exists$-variables corresponding to the Skolem functions appear only in the conclusion. A similar problem occurs with a constraint of the form

$$\phi(\bar{x}), u = f(\bar{x}), v = x_i \to S(\bar{y}).$$

(6) **Eliminate restricted constraints:**
In this step, we eliminate some constraints as follows. We first classify all function symbols as either free or restricted. Free function symbols will be those for which we are free to pick any values. We recursively define the restricted function symbols as follows: f is restricted if there is a constraint ϕ which restricts f and such that all functions which restrict ϕ are restricted. In particular, f is restricted if a constraint ϕ restricts f and no function restricts ϕ. All other functions are free. Now, we set Λ_6 to be the set of constraints $\phi \in \Lambda_5$ such that no free function restricts ϕ. The rationale for this step is that if a free function f restricts ϕ, then we can choose the values of f such that a restricting equation for v_f in the premise never holds.

For example, in the following constraints

ϕ_1	$R(x, y), u = f(x, y), v = g(x, y), u = y$	$\rightarrow$	$T(x, v)$
ϕ_2	$R(x, y), u = f(x, y), v = g(x, y), v = y$	$\rightarrow$	$T(x, u)$

f restricts ϕ_1, which restricts g and g restricts ϕ_2, which restricts f. Therefore, both f and g are free and we can eliminate both constraints. It is clear that this is sound, since we can set f and g such that the range of f and g are disjoint from each other and from the values appearing in R and T (see the discussion on semantics of SkED constraints in Section 8). Then the premises of ϕ_1 and ϕ_2 will never hold and both constraints will always be satisfied, regardless of the choice of R and T.

(7) **Check for remaining restricted constraints:**
If there are any restricted constraints in Λ_6, abort. Otherwise, set $\Lambda_7 := \Lambda_6$.

(8) **Check for dependencies:**
For every $\phi \in \Lambda_7$ and every term variable v in ϕ, define $D_{\phi,v}$ to be the set of base variables on which v depends. Set V_ϕ to the set of base variables which appear in the conclusion of ϕ. Now, for every term variable v in the conclusion of ϕ, check that $V_\phi \subseteq D_{\phi,v}$. If this fails, abort. Otherwise, set $\Lambda_8 := \Lambda_7$.

In this step and the next, we make sure that once we replace Skolem functions with $\exists$-variables (which will happen in Step 11), we get equivalent constraints. One direction of this equivalence is straightforward: We can set the $\exists$-variables to the values of the corresponding Skolem functions. The difficulty is in the other direction. We must make sure that, given values for the $\exists$-variables witnessing that the ED-constraints hold, we can set the Skolem functions to also witness that the corresponding SkED-constraints hold.

To understand why we must check dependencies, consider the constraints

Σ_{12} is	$A(x), y = f(x)$	$\rightarrow$	$F(x, y)$
	$B(u), v = g(u)$	$\rightarrow$	$G(u, v)$
Σ_{23} is	$F(x, y), G(u, v)$	$\rightarrow$	$T(x, y, u, v)$

which, when composed, yield the single constraint ϕ:

$$A(x), B(u), y = f(x), v = g(u) \rightarrow T(x, y, u, v).$$

In this case, $D_{\phi,y} := x$, $D_{\phi,v} := u$, $V_\phi := \{x, u\}$ and the check fails. The problem with the obvious replacement of Skolem functions with ∃-variables is that in ϕ'

$$A(x), B(u) \to \exists y, v\, T(x, y, u, v)$$

and y depends on both x and u, instead of only on x, as desired, and v also depends on both x and u, instead of only on u, as desired. In fact, ϕ and ϕ' are not equivalent, as witnessed by the relations $A := \{1, 2\}$, $B := \{3\}$, and $T := \{\langle 1537\rangle, \langle 2538\rangle\}$ for which ϕ' holds, but not ϕ. The problem here is that $x = 1$ forces $g(3) := 7$, yet $x = 2$ forces $g(3) = 8$, and these choices are incompatible. Interestingly, the following set of two first-order sentences Φ is equivalent to ϕ:

$$A(x) \to \exists y \forall u(B(u) \to T(x, y, u, v))$$
$$B(u) \to \exists v \forall x(A(x) \to T(x, y, u, v))$$

However, these are not ED-constraints, which is what our algorithm tries to produce.

(9) **Combine dependencies:**
Set $\Lambda_9 := \{\psi_\Phi : \emptyset \neq \Phi \subseteq \Lambda_8\}$, where ψ_Φ is defined as follows. If there is a function f which appears in every $\phi \in \Phi$, then the premise of ψ_Φ consists of those atoms in all the premises in Φ and the conclusion of ψ_Φ consists of those atoms in all the conclusions of Φ (remove duplicate atoms). Otherwise, ψ_Φ is some constraint in Φ. Notice that $\Lambda_9 \supseteq \Lambda_8$, since $\psi_{\{\phi\}} = \phi$.

We have already seen the need for this step in Example 7. There, Λ_8 is

ϕ_1 is	$R(x, y), u = f(x, y)$	$\to$	$S(x, u)$
ϕ_2 is	$R(x, y), u = f(x, y)$	$\to$	$S(u, y)$

and Λ_9 is

ψ_{ϕ_1}	$R(x, y), u = f(x, y)$	$\to$	$S(x, u)$
ψ_{ϕ_2}	$R(x, y), u = f(x, y)$	$\to$	$S(u, y)$
ψ_{ϕ_1,ϕ_2}	$R(x, y), u = f(x, y)$	$\to$	$S(x, u), S(u, y)$

This step is not always as trivial as it looks in Example 7. Consider, for example, the constraints from the proof of Theorem 8, to follow, where $1 \leq i \leq k$:

Σ_{12}^k is	$R_0(x)$	$\to$	$\exists y\, S_0(x, y)$
	$R_i(x)$	$\to$	$S_i(x)$
Σ_{23}^k is	$S_0(xy), S_i(x)$	$\to$	$T_i(y)$

In this case, Λ_8 consists of k constraints of the form

$$R_0(x),\ y = f(x), R_i(x) \to T_i(y)$$

and Λ_9 consists of $2^k - 1$ constraints of the form

$$R_0(x),\ y = f(x), R_Z(x) \to T_Z(y),$$

where Z is a nonempty subset of $\{1, \ldots, k\}$ and where $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$ and similarly for T_Z.

(10) **Remove redundant constraints:**
Pick some set $\Lambda_{10} \subseteq \Lambda_9$ such that $\Lambda_{10} \vdash^* \phi$ for every $\phi \in \Lambda_9$, and such that this does not hold for any proper subset of Λ_{10}.

We have seen earlier that in Example 7, Λ_9 is

ψ_{ϕ_1}	$R(x, y), u = f(x, y)$	$\rightarrow$	$S(x, u)$
ψ_{ϕ_2}	$R(x, y), u = f(x, y)$	$\rightarrow$	$S(u, y)$
ψ_{ϕ_1,ϕ_2}	$R(x, y), u = f(x, y)$	$\rightarrow$	$S(x, u), S(u, y)$

In this case, $\Lambda_{10} := \{\psi_{\phi_1,\phi_2}\}$, since both ψ_{ϕ_1} and ψ_{ϕ_2} follow from ψ_{ϕ_1,ϕ_2}. This happens because the premises of ψ_{ϕ_1} and ψ_{ϕ_2} are the same, but this is not always the case. In particular, $\Lambda_{10} = \Lambda_9$ in the case of the constraints from the proof of Theorem 8, discussed before.

(11) **Replace functions with ∃-variables:**
Set $\Lambda_{11} := \{\phi' : \phi \in \Lambda_{10}\}$, where the premise of ϕ' is the premise of ϕ with all equalities removed and where the conclusion of ϕ' is the conclusion of ϕ, with all variables appearing on the left of equalities in ϕ existentially quantified.

This step is where the elimination of Skolem functions actually takes place, but since most of the preparatory work has already been done, it is very simple. For example, it converts

$$R(x, y), u = f(x, y) \rightarrow S(x, u), S(u, y) \qquad \text{to}$$
$$R(x, y) \rightarrow \exists u\, S(x, u), S(u, y).$$

(12) **Eliminate unnecessary ∃-variables:**
Set $\Lambda_{12} := \{\phi' : \phi \in \Lambda_{11}\}$ and return Λ_{12}, where ϕ' is like ϕ, but where existentially quantified variables which do not appear in the conclusion atom have been removed (with their corresponding existential quantifier).

If we apply Step 11 to the following constraint:

$$R(x, y), u = f(x, y), v = g(x, y) \rightarrow S(x, u), S(u, y)$$

we would obtain

$$R(x, y) \rightarrow \exists u, v\, S(x, u), S(u, y).$$

Clearly, v is not needed, so in this step, we replace the aforementioned constraint with

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

Example 9. Consider three runs of the algorithm DeSkolemize(Σ^i_{13}), for $i \in \{1, 2, 3\}$. Let $\Sigma^i_{13} = \{\gamma_1, \ldots, \gamma_i\}$ be a set of the following (unnested) SkTGD constraints:

γ_1	$R_1(y), R_2(x), y = f(x)$	$\rightarrow$	$T_1(x)$
γ_2	$R_2(x), y = f(x)$	$\rightarrow$	$T_2(y)$
γ_3	$R_2(x), y = f(x)$	$\rightarrow$	$R_1(y)$

For completeness, we note that each Σ^i_{13} is obtained by first de-Skolemizing the ED-mappings given by Σ^i_{12} and Σ^i_{23}, which are shown next, and then invoking

SkED-compose:

i	Σ_{12}^{i}	Σ_{23}^{i}	Σ_{13}^{i}
1.	$\{\alpha_1, \alpha_2\}$	$\{\beta_2\}$	$\{\gamma_1\}$
2.	$\{\alpha_1, \alpha_2\}$	$\{\beta_1, \beta_2\}$	$\{\gamma_1, \gamma_2\}$
3.	$\{\alpha_1, \alpha_2, \alpha_3\}$	$\{\beta_1, \beta_2\}$	$\{\gamma_1, \gamma_2, \gamma_3\}$

Dependencies $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2$ are specified as:

α_1	$R_1(x)$	$\rightarrow$	$S_1(x)$
α_2	$R_2(y)$	$\rightarrow$	$\exists z(S_2(zy))$
α_3	$S_2(zy)$	$\rightarrow$	$R_1(z)$
β_1	$S_2(zy)$	$\rightarrow$	$T_2(z)$
β_2	$S_1(x), S_2(xy)$	$\rightarrow$	$T_1(y)$

In all three runs of DeSkolemize(Σ_{13}^{i}), Steps 2 and 3 pass, since each of $\gamma_1, \gamma_2, \gamma_3$ is cycle-free and has no multiple atoms with the same function symbol. Step 4 has no effect, since the variable names of the dependencies are already aligned. The remaining steps are explained next.

In the run DeSkolemize($\{\gamma_1\}$), Step 5 has no effect because $\{\gamma_1\}$ is a singleton set. Its only member γ_1 gets eliminated in Step 6, since there are no rules in $\{\gamma_1\}$ with f-restricting atoms in conclusions. Intuitively, γ_1 is a tautology because we can always construct an f whose range is disjoint with R_1. Hence, $\{\gamma_1\}$ is equivalent to the empty set of constraints, which is trivially in TGD.

In the run DeSkolemize($\{\gamma_1, \gamma_2\}$), γ_2 contains an f-restricting atom T_2 in its conclusion. Hence, we cannot eliminate the restricted constraint γ_1 in Step 6, and so de-Skolemization aborts in Step 7.

In the run DeSkolemize($\{\gamma_1, \gamma_2, \gamma_3\}$), we are able to de-Skolemize, despite γ_2. In Step 5, $\Delta_0 = \{\gamma_1, \gamma_2, \gamma_3\}$. By considering the only function symbol f, we get $\Delta_1 = \{\psi, \gamma_2, \gamma_3\} \equiv \Delta_0$, where ψ is obtained by eliminating the restricting atom $R_1(y)$ from the premise of γ_1 as

$$\psi \ := \ R_2(x), \ y = f(x) \rightarrow T_1(x).$$

Clearly, $\Delta_0 \vdash^* \psi$, since $\Delta_0 \supset \{\gamma_1, \gamma_3\} \vdash^* \psi$. Moreover, Δ_1 has no restricting constraints, so Step 6 has no effect and Step 7 passes. Step 8 succeeds with $\Lambda_8 = \Lambda_7 = \{\psi, \gamma_2, \gamma_3\}$, since every dependency in Δ_1 has, at most, one term variable y in its conclusion. Taking γ_3 as an example, we get $V_{\gamma_3} = \{x, y\}$, $D_{\gamma_3,x} = D_{\gamma_3,y} = \bigcup_{u \in V_{\gamma_3}} D_{\gamma_3,u} = \{x\}$.

In Step 9, combining the dependencies for $\Phi = \Lambda_8$ yields

$$\gamma_4 \ := \ R_1(y), R_2(x), \ y = f(x) \rightarrow T_1(x), T_2(y), R_1(y)$$

(combinations resulting from proper subsets of Λ_8 are not shown for brevity). In Step 10, we remove the redundant constraints, which include ψ, γ_2, γ_3, because they share the premise with γ_4 and their conclusion is subsumed by that of γ_4; we obtain $\Lambda_{10} = \{\gamma_4\}$. Finally, replacing function f by an existential variable in γ_4 yields

$$\Lambda_{12} = \{R_2(x) \rightarrow \exists y(T_1(x), T_2(y), R_1(y))\}.$$

Thus, DeSkolemize($\{\gamma_1, \gamma_2, \gamma_3\}$) $\subseteq$ TGD.

The following technical lemma shows that in the case of constraints which arise from Skolemizing ED constraints and running SkED-compose on them, there is no need to consider nested Skolem functions. The main point is that a constraint of the form (20), which follows, can be replaced by a constraint of the form (23), also shown. The technical conditions depicted reflect the case that arises from applying a resolution step, which is the only step in SkED-compose which may give rise to nested Skolem functions. Constraint (20) is obtained by resolution of constraints (18) and (19).

Lemma 10. *If $\Sigma \subseteq \mathrm{SkCQ}^{=}$ consists of the following three constraints:*

$$\psi(\bar x), \bar u = \bar f(\bar x) \rightarrow S(j\,\bar y), \rho(\bar x, \bar u) \tag{18}$$

$$\phi(\bar y), S(\bar y), \bar v = \bar g(\bar y) \rightarrow \tau(\bar y, \bar v) \tag{19}$$

$$\psi(\bar x), \phi(j\,\bar y), \bar u = \bar f(\bar x), \bar v = \bar g(j\,\bar y) \rightarrow \tau(j\,\bar y, \bar v), \tag{20}$$

where

—*$\bar x$, $\bar y$, $\bar u$, and $\bar v$ are disjoint tuples of variables,*

—*S is a relational symbol,*

—*$\psi(\bar z)$, $\phi(\bar z)$, $\rho(\bar z)$, and $\tau(\bar z)$ are conjunctions of atoms with variables from $\bar z$,*

—*$j : \bar y \rightarrow \bar x\bar u$ is a function mapping variables to variables (so $j\bar y$ is a tuple of variables from $\bar x\bar u$, possibly with repetitions),*

—*$\bar f\,\bar x$ is a tuple of functions $f_1, \ldots, f_k$ whose arguments are variables from $\bar x$, and*

—*$\bar g\,\bar y$ is a tuple of functions $g_1, \ldots, g_\ell$ disjoint from $\bar f$, whose arguments are variables from $\bar y$,*

then Σ is equivalent to $\Sigma' \subseteq \mathrm{SkCQ}^{=}$, which consists of

$$\psi(\bar x), \bar u = \bar f'(\bar x) \rightarrow S(j\,\bar y), \rho(\bar x, \bar u) \tag{21}$$

$$\phi(\bar y), S(\bar y), \bar v = \bar g'(\bar y) \rightarrow \tau(\bar y, \bar v) \tag{22}$$

$$\psi(\bar x), \phi(j\,\bar y), \bar u = \bar f(\bar x), \bar v = \bar h(\bar x) \rightarrow \tau(j\,\bar y, \bar v), \tag{23}$$

where $\bar f'$, $\bar g'$, and $\bar h$ are disjoint tuples of functions which do not appear in Σ. Moreover, (20) *is equivalent to* (23)*.*

Proof. If Σ holds, then we have functions $\bar f$ and $\bar g$ witnessing this. Set $\bar f' := \bar f$, $\bar g' := \bar g$ and define functions $\bar h$ by $\bar h(\bar x) := \bar g(j\,\bar y)$, where $\bar u := \bar f(\bar x)$ (so $j\,\bar y$ is uniquely determined by $\bar x$). Then (21), (22), and (23) hold. That is, $\Sigma \models \Sigma'$. A similar argument shows that (20) implies (23). If, on the other hand, Σ holds, then we have functions $\bar f'$, $\bar g'$ witnessing that (21) and (22) hold. Then $\bar f := \bar f'$ and $\bar g := \bar g'$ witness that (18) and (19) hold, and (20) follows from these.

Now, assume that (23) holds. We have $\bar f$ and $\bar h$ witnessing this. We want to show that (20) holds. Define G as follows:

$$G(\bar a) := \{\bar h(\bar x) : \psi(\bar x), \phi(j\,\bar y), \bar u = \bar f(\bar x), j\,\bar y = \bar a\}$$

(recall that $j\,\bar y$ is a tuple of the variables in $\bar x, \bar u$). Set $\bar g(\bar a) := \bar b$ for some arbitrary tuple $\bar b \in G(\bar a)$ if $G(\bar a) \neq \emptyset$. Otherwise, set $\bar g(\bar a) := \bar c$ for some arbitrary tuple $\bar c$.

We must show that $\bar f$ and $\bar g$ as defined witness that (19) and (20) hold. Assume that the premise of (20) holds for some values of $\bar x$. Set $\bar u := f(\bar x)$ and $\bar v := \bar g(j\bar y)$. We need to show that $\tau(j\,\bar y, \bar v)$ holds.

Since the premise of (20) holds, $h(\bar{x}) \in G(j\bar{y})$ and therefore $G(j\bar{y}) \neq \emptyset$. This implies that $\bar{g}(j\bar{y}) \in G(j\bar{y})$. In other words, $\bar{g}(j\bar{y}) = \bar{h}(\bar{z})$ for some $\bar{z}$ such that $\psi(\bar{z}), \phi(j\bar{y})$ and $\bar{x}$, $\bar{f}(\bar{x})$ and $\bar{z}$, $\bar{f}(\bar{z})$ coincide on the range of j. Since $\bar{g}(j\bar{y}) = \bar{h}(\bar{z})$, the premise of

$$\psi(\bar{z}), \phi(j\,\bar{y}), \bar{v} = \bar{h}(\bar{z}) \rightarrow \tau(j\,\bar{y}, \bar{v}) \tag{24}$$

holds. Furthermore, (24) is obtained from (23) by the substitution $\bar{x} \mapsto \bar{z}$. Since (23) holds, (24) holds. Therefore $\tau(j\bar{y}, \bar{v})$ holds, as desired. □

Proof (Theorem 7). At the beginning and end of every step of the DeSkolemize, we have constraints that are equivalent to each other. This is obvious for some steps, which do nothing other than verify that some condition holds, and is easy to verify for all other steps except the step which replaces functions with $\exists$-variables. We will call the constraints with functions just before this step Λ and those with $\exists$-variables just after this step Λ'. We need to show that $\Lambda \equiv \Lambda'$. The direction $\Lambda \models \Lambda'$ is easy; all we need to do is set the $\exists$-variables to those values given by the corresponding functions. The direction $\Lambda' \models \Lambda$ is harder. We have done some of the previous steps, particularly the combining of dependencies, to ensure that this holds. So, suppose $D \models \Lambda'$ and that v is the $\exists$-variable which corresponds to the function f_v. We set $f_v(\bar{c})$ to a value of v which witnesses that a constraint ψ_Φ holds for $\bar{c}$, where ψ_Φ is as defined in the step "combine dependencies" and where the premise of ψ_Φ holds for $\bar{c}$, yet the premise of no constraint $\psi_{\Phi'}$ with $\Phi \subset \Phi'$ holds for $\bar{c}$. Clearly, there is a unique such set Φ, since if both ψ_{Φ_1} and ψ_{Φ_2} hold for $\bar{c}$, then $\psi_{\Phi_1 \cup \Phi_2}$ also holds for $\bar{c}$. Now, assume that the premise of some constraint $\phi \in \Lambda$ (in which f appears) holds for a tuple $\bar{c}$. Then, we must have $\phi \in \Phi$ and since ψ_Φ holds and its premise holds for $\bar{c}$, its conclusion must also hold for $\bar{c}$. Since we have set $f_v(\bar{c})$ to a value that witnesses this, the conclusion of ϕ must also hold for $\bar{c}$. □

Proof (Theorem 3). Procedure DeSkolemize may only abort at Steps 2, 3, 4, 7, or 8 and has no loops that may not terminate. Therefore, if Step 9 is reached, DeSkolemize will terminate. Furthermore, all steps can be carried out in polynomial time, except for Step 9, which may give an exponential increase in the size of constraints. However, if the hypotheses of Part 2 hold, then no such exponential increase can occur. □

Proof (Theorem 8). Set $[k] := \{1, \ldots, k\}$. Consider the TGD-mappings m_{12}^k and m_{23}^k given by $(\sigma_1^k, \sigma_2^k, \Sigma_{12}^k)$ and $(\sigma_2^k, \sigma_3^k, \Sigma_{23}^k)$, where

Σ_{12}^k is	$R_0(x)$	$\rightarrow$	$\exists y\, S_0(x, y)$
	$R_i(x)$	$\rightarrow$	$S_i(x)$
Σ_{23}^k is	$S_0(xy), S_i(x)$	$\rightarrow$	$T_i(y)$

for $i \in [k]$ and where $\sigma_1^k = \{R_i : i \in \{0, \ldots, k\}\}$, $\sigma_2^k = \{S_i : i \in \{0, \ldots, k\}\}$, and $\sigma_3^k = \{T_i : i \in [k]\}$. The SkTGD-composition $m_{13}^k := m_{12}^k \circ m_{13}^k$ is given by the set Σ_{13}^k of constraints

$$R_0(x),\, y = f(x),\, R_i(x) \rightarrow T_i(y)$$

for $i \in [k]$, which grows linearly in the size of $\Sigma_{12}^k \cup \Sigma_{23}^k$.

The TGD-composition $m_{13}^k := m_{12}^k \circ m_{13}^k$ can be obtained by de-Skolemizing Σ_{13}^k. It is given by the set Σ'^k_{13} of $2^k - 1$ constraints

$$R_0(x), R_Z(x) \to \exists y\ T_Z(y)$$

such that $\emptyset \neq Z \subseteq [k]$, where $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$ and $T_Z(x) := \bigwedge_{i \in Z} T_i(x)$. On the other hand, m_{13}^k cannot be expressed by any $(\sigma_1, \sigma_3, \Sigma)$, $\Sigma \subseteq$ TGD) where Σ has fewer than 2^{k-1} constraints. This inexpressibility result is given at the end of Section 7, in which we introduce a mechanism that enables us to complete this proof. □

7. INEXPRESSIBILITY TOOL FOR EMBEDDED DEPENDENCIES

In this section we develop a formal vehicle for proving inexpressibility results for ED, TGD, FullD, and FullTGD mappings. We use it to show the inexpressibility claim of Theorem 8. However, the tools presented in this section may be of independent value and could be used for obtaining inexpressibility results for other problems in database theory.

To illustrate the intuition behind this mechanism, consider a set Σ of logical sentences over σ, and some structure A_0 over σ. The truth value of Σ in the structure A_0 is $A_0 \models \Sigma$. Now, suppose we add one or more tuples to some relation in A_0 and obtain the structure A_1. The truth value of Σ in A_1 may remain the same, or may flip from "true" to "false" or from "false" to "true." As we keep adding tuples, we obtain a chain of successively larger structures $A_0, \ldots, A_n, \ldots$ with the corresponding truth values of Σ for each structure in the chain. The truth values of Σ form contiguous segments within which Σ remains "true" (positive segments) or "false" (negative segments). For example, the truth values ("true," "true," "false," "false," "true.") for a chain of structures $(A_0, A_1, A_2, A_3, A_4)$ partition the chain into three segments: a positive segment (A_0, A_1) followed by a negative segment (A_2, A_3) followed by a positive segment (A_4). To characterize Σ, we count the maximal number n of negative segments for *any* such chain of structures over Σ. If this number is finite, we call Σ n-monotonic, and nonmonotonic otherwise. To characterize a class of constraints, we study the monotonicity properties of its constituent sentences.

Next we give formal definitions of chains, segments, and n-monotonic sentences.

Definition 4. Let $\mathcal{K}$ be a set of structures over σ and Σ be a set of sentences over σ. Then, for each pair of structures $A, B \in \mathcal{K}$:

(1) $A \subseteq B$ if for all relation symbols R over σ, $R^A \subseteq R^B$;
(2) $A \subset B$ if $A \subseteq B$ and $A \neq B$; and
(3) $A \simeq_\Sigma B$, if ($A \models \Sigma$ iff $B \models \Sigma$) and $\forall C \in \mathcal{K}(A \subseteq C \subseteq B \vee B \subseteq C \subseteq A \to A \models \Sigma$ iff $C \models \Sigma)$.

Definition 5. A set $\mathcal{K}$ of structures over signature σ is a *chain* if $(\mathcal{K}, \subset)$ is a total order.

Definition 6. A *segment* is an equivalence class of $(\mathcal{K}, \simeq_\Sigma)$. Segment $\mathcal{S}$ is *positive* for Σ if $A \models \Sigma$ for all $A \in \mathcal{S}$, and *negative* otherwise.

Definition 7. Let Σ be a set of sentences. Then:

(1) Σ is *n-monotonic* if every chain for Σ has (at most) n negative segments
(2) Σ is *strictly n-monotonic* if Σ is n-monotonic and there exists a chain for Σ with exactly n negative segments; and
(3) Σ is *monotonic* if it is n-monotonic for some $n \in \mathbb{N}$.

We proceed to study the monotonicity properties of the mapping languages that we focus on in this article. First, we illustrate some 0-monotonic, 1-monotonic, and nonmonotonic sentences so as to familiarize the reader with the concept.

Example 10. Here, $\Sigma = \{R(x) \to R(x)\}$ is 0-monotonic. More generally, Σ is 0-monotonic if and only if Σ is a tautology. In fact, if Σ is a tautology, then $A \models \Sigma$ for all A over the signature of Σ, that is, each chain of structures for Σ contains a single positive segment. Conversely, if no chain contains a negative segment, then $A \models \Sigma$ for all A and hence Σ is a tautology. □

Example 11. Here, $\Sigma = \{R(x) \to \exists y\, S(y)\}$ is 1-monotonic. The sentence in Σ is equivalent to $R \neq \emptyset \to S \neq \emptyset$. Hence, Σ partitions each chain $\mathcal{K}$ into, at most, three segments containing structures $(\emptyset, \emptyset), (R, \emptyset), (R', S)$, respectively, for some nonempty R, R', S, $R \subseteq R'$. The structure $(R, \emptyset)$ belongs to the only negative segment in such a chain. □

Example 12. Here, $\Sigma = \{R(x) \to S(x)\}$ is nonmonotonic. Let $A_k = (\{c_0, \dots, c_k\}, \{c_0, \dots, c_{k-1}\})$, $B_k = (\{c_0, \dots, c_k\}, \{c_0, \dots, c_k\})$ be structures over $\sigma = \{R, S\}$, where all c_i constants are distinct. Clearly, $A_k \not\models \Sigma$, $B_k \models \Sigma$, $A_k \subset B_k \subset A_{k+1}$. In other words, there exists a chain A_0, B_0, A_1, B_1, $\dots$, A_k, B_k, $\dots$ of singleton segments that witnesses the nonmonotonicity of Σ. □

We generalize the preceding examples for several classes of sentences. First, we consider tuple-generating dependencies that may only contain dependent (see the definition to follow) existential variables. We show that such sentences are 0-monotonic or nonmonotonic (Lemma 5). We will see that this class of sentences, which subsumes full tuple-generating dependencies, is the only source of nonmonotonic dependencies that make up the constraints in our mapping languages.

Second, we examine tuple-generating dependencies that express inclusions of Boolean conjunctive queries, and prove that these are 1-monotonic (Lemma 8). Third, we prove the same monotonicity property for equality-generating dependencies (Lemma 9). After examining the aforementioned classes of constraint, we prove Lemma 11, which establishes a monotonicity bound for a set of sentences based on the monotonicity of its members. These lemmas lead to the main result of this section, which states the monotonicity properties for the (first-order) mapping languages that we consider.

We call a variable $y \in \{\bar{y}\}$ *dependent* in a tgd $\varphi = \forall \bar{x}(P(\bar{x}) \to \exists \bar{y}\, Q(\bar{x}, \bar{y}))$ if there exist variables $u_1, \dots, u_n$ such that u_i and u_{i+1} appear in the same atom of Q, $1 \leq i < n$, and $u_1 = y$, $u_n \in \{\bar{x}\}$. Otherwise, y is called independent. We start with tuple-generating dependencies φ that may only contain dependent existential variables.

LEMMA 5. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \rightarrow \exists\bar{y}\, Q(\bar{x}, \bar{y}))$ be a tuple-generating dependency such that each $y \in \{\bar{y}\}$ is dependent. Then, φ is 0-monotonic or nonmonotonic.*

PROOF. Assume that φ is not 0-monotonic, namely, it is not a tautology. The proof of nonmonotonicity is based on the following two observations:

—If φ is false in some structure A, then there exists a larger structure $B \supset A$ that makes φ true (Lemma 6); and

—if φ is true in some structure B, then there exists a larger structure $A \supset B$ that makes φ false (Lemma 7).

Together, the previous observations assert the existence of an infinite chain of alternating structures that witness the nonmonotonicity of φ. □

We prove the subordinate Lemmas 6 and 7. Notice that Lemma 6 applies to arbitrary tuple-generating dependencies.

LEMMA 6. *If a tuple-generating dependency φ is violated in some structure A, then there exists a larger structure $B \supset A$ that satisfies the dependency. In other words, $\forall A : A \not\models \varphi \rightarrow \exists B(B \models \varphi \wedge A \subset B)$.*

PROOF. Let A be a structure such that $A \not\models \varphi$. Construct a complete structure B in which every relation R^B is a cross-product over the domain of A. Clearly, $B \models \varphi$, $A \subseteq B$. Since $A \not\models \varphi$, thus $A \neq B$ and we obtain $A \subset B$. □

LEMMA 7. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \rightarrow \exists\bar{y}\, Q(\bar{x}, \bar{y}))$ be a tuple-generating dependency such that each $y \in \{\bar{y}\}$ is dependent, and φ is not a tautology. Then, if φ is true in some structure B, then there exists a larger structure $A \supset B$ that violates the dependency. In other words, $\forall B : B \models \varphi \rightarrow \exists A(A \not\models \varphi \wedge B \subset A)$.*

PROOF. Let B be a structure such that $B \models \varphi$. Since φ is not a tautology, there exists a structure F_0 such that $F_0 \not\models \varphi$. F_0 is nonempty (since the empty structure makes φ true). Let F be an isomorphic copy of F_0 that does not have any constants in common with B. Set $A := B \cup F$. Clearly, $A \supset B$. We show that $A \not\models \varphi$.

Since $F_0 \not\models \varphi$, thus $F \not\models \varphi$. Let $\bar{t}$ be a tuple such that $F \models P(\bar{t})$, $F \not\models \exists\bar{y}\, Q(\bar{t}, \bar{y})$. Such a tuple exists since F is nonempty. Given that $F \models P(\bar{t})$, we obtain $A = B \cup F \models P(\bar{t})$ because P is a conjunctive query.

There remains to show that $A = B \cup F \not\models \exists\bar{y}\, Q(\bar{t}, \bar{y})$. Suppose the contrary, that is, $A \models \exists\bar{y}\, Q(\bar{t}, \bar{y})$. Then, there is an atom $R(\bar{z})$ in Q where $\bar{z} \subseteq \bar{x} \cup \bar{y}$, and a tuple $\bar{r}$ such that $F \not\models R(\bar{r})$ and $B \cup F \models R(\bar{r})$. Since B and F do not share constants, $R(\bar{r})$ must contain only constants from B. In other words, $\bar{z} \subseteq \bar{y}$, that is, R contains only $\bar{y}$ variables, and $\bar{z} \neq \emptyset$. By the premise of the lemma, each $\bar{y}$ variable is dependent. In other words, there exists an atom in Q that contains both $\bar{x}$ and $\bar{y}$ variables. However, this atom is satisfied in A only if $B \cap F \neq \emptyset$, a contradiction. □

To illustrate Lemma 7, consider the dependency $\forall xy(R(xy) \rightarrow \exists z(R(xz), S(z))$. Since z is connected, the dependency satisfies the premise of the lemma and is nonmonotonic.

Next we consider sentences φ that express inclusions of Boolean conjunctive queries. We show that such sentences are 1-monotonic.

LEMMA 8. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}\, Q(\bar{y}))$. Then, φ is 1-monotonic.*

PROOF. Observe that φ is equivalent to the sentence $\psi \to \theta$, where $\psi = \exists\bar{x}\, P(\bar{x})$ and $\theta = \exists\bar{y}\, Q(\bar{y})$ are Boolean conjunctive queries.

We need to show that each chain for φ contains at most one negative segment. Assume the opposite, namely, there exists a chain that contains structures $A \subset B \subset C$ such that $A \not\models \varphi$, $B \models \varphi$, $C \not\models \varphi$ (where A and C belong to two distinct negative segments). Then, the following formula must be true:

$$(A \models \psi \wedge A \not\models \theta) \wedge (B \not\models \psi \vee B \models \theta) \wedge (C \models \psi \wedge C \not\models \theta).$$

Since $A \subset B$, thus $A \models \psi$ implies $B \models \psi$. In other words, $B \models \theta$ must hold to make the disjunction true. But since $B \subset C$, $B \models \theta$ implies $C \models \theta$. This contradicts $C \not\models \theta$. Hence, our assumption was false, and φ is 1-monotonic. □

As a last building block, we consider a generalized form of equality-generating dependencies, where multiple equality atoms may appear in the conclusion, and we show that these are 1-monotonic.

LEMMA 9. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \to \psi(\bar{x}))$, where $P(\bar{x})$ is a conjunctive query and $\psi(\bar{x})$ is a set of equalities between variables in $\bar{x}$. Then, φ is 1-monotonic.*

PROOF. We need to show that each chain for φ contains, at most, one negative segment. Consider a chain for φ which contains a structure A such that $A \not\models \varphi$. Then, there exists a tuple t that satisfies the premise $A \models P(t)$, but violates the equality conditions $A \not\models \psi(t)$. Once the equality conditions are violated, $\psi(t)$ remains false in every larger structure. Phrased differently, for every structure $B \supset A$, $B \models P(t)$ and $B \not\models \psi(t)$, and hence $B \not\models \varphi$. Therefore, each chain over φ contains, at most, one negative segment. Hence, φ is 1-monotonic. □

We examined the building blocks of our mapping languages. The following three lemmas explain how to put these building blocks together.

LEMMA 10. *A set Σ of embedded dependencies is equivalent to a set Σ' of embedded dependencies such that each $\psi \in \Sigma'$ satisfies the premise of one of the Lemmas* 5, 8, *or* 9.

PROOF. Let $\varphi \in \Sigma$, $\varphi = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}\,\psi(\bar{x}, \bar{y}))$. Construct φ_1 by eliminating all equality atoms from φ that involve variables from $\bar{y}$, as follows: If $y \in \bar{y}$ appears in an equality atom $y = z$ in ψ, replace all occurrences of y in ψ by z and remove $y = z$. Clearly, $\varphi \equiv \varphi_1$. If φ_1 has no equality atoms left, it is a tuple-generating dependency; set $\Sigma_\varphi = \{\varphi_1\}$. Otherwise, since each equality atom in φ_1 mentions only variables in $\bar{x}$, thus φ_1 is equivalent to $\{\varphi_2, \varphi_3\}$, where φ_2 is a tuple-generating dependency and φ_3 is a generalized equality-generating dependency satisfying the premise of Lemma 9; set $\Sigma_\varphi = \{\varphi_2, \varphi_3\}$.

Let $\varphi' = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}\, Q(\bar{x}, \bar{y}))$ be the tuple-generating dependency in Σ_φ. If each $y \in \{\bar{y}\}$ is dependent (or φ' is a full tgd), then φ' satisfies the premise of Lemma 5. Otherwise, $\varphi' = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}_1, \bar{y}_2(Q_1(\bar{x}, \bar{y}_1), Q_2(\bar{y}_2))$, where $\bar{y}_1$ contains only dependent variables and $\bar{y}_2$ contains only independent variables. Consequently, $\varphi' \equiv \{\varphi_a, \varphi_b\}$, where $\varphi_a = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}_1 Q_1(\bar{x}, \bar{y}_1))$ satisfies the premise of Lemma 5 and $\varphi_b = \forall\bar{x}(P(\bar{x}) \to \exists\bar{y}_2 Q_2(\bar{y}_2))$ satisfies the premise of Lemma 8. □

Lemma 11 establishes an upper bound on the combined monotonicity for a set of arbitrary monotonic sentences.

LEMMA 11. *If every sentence $\varphi \in \Sigma$ is n_φ-monotonic, then Σ is n-monotonic where $n = \sum_{\varphi \in \Sigma} n_\varphi$.*

PROOF. Let $\mathcal{K}$ be a chain for Σ. Here, Σ partitions $\mathcal{K}$ into a set of disjoint segments $(\mathcal{K}, \simeq_\Sigma)$. Each sentence $\varphi_i \in \Sigma$ partitions $\mathcal{K}$ into a set of disjoint segments $(\mathcal{K}, \simeq_{\varphi_i})$. Let $\mathcal{S}$ be a negative segment of $(\mathcal{K}, \simeq_\Sigma)$, and let $A \in \mathcal{S}$, $A \not\models \Sigma$. Hence, there exists some φ_i such that $A \not\models \varphi_i$. Therefore, $\mathcal{S}$ overlaps with some negative segment $\mathcal{S}_i \in (\mathcal{K}, \simeq_{\varphi_i})$ which contains A. Notice that for each $B \in \mathcal{S}_i$, $B \not\models \varphi_i$ implies that $B \not\models \Sigma$. Therefore, $\mathcal{S}_i \subseteq \mathcal{S}$. In other words, each negative segment of $(\mathcal{K}, \simeq_\Sigma)$ fully contains a negative segment of $(\mathcal{K}, \simeq_\varphi)$ for some $\varphi \in \Sigma$. Since all segments of $(\mathcal{K}, \simeq_\Sigma)$ are disjoint, there are (at most) as many negative segments in $(\mathcal{K}, \simeq_\Sigma)$ as the cumulative number n of negative segments in $(\mathcal{K}, \simeq_\varphi)$ for all $\varphi \in \Sigma$. Each $\varphi \in \Sigma$ is n_φ-monotonic, so $(\mathcal{K}, \simeq_\varphi)$ contains, at most, n_φ negative segments. Therefore, $n \le \sum_{\varphi \in \Sigma} n_\varphi$. □

We would like to make an equally general statement for the case where Σ contains a nonmonotonic dependency. However, it seems difficult to do so for an arbitrary class of dependencies. Although a nonmonotonic dependency has chains with an unbounded number of negative segments, it is possible that the monotonic sentences in Σ "wipe out" all but a finite number of negative segments in each such chain. Therefore, in Lemma 12 we focus specifically on embedded dependencies.

LEMMA 12. *Let Σ be a set of embedded dependencies in which each dependency satisfies the premise of one of the Lemmas* 5, 8, *or* 9. *Then Σ is nonmonotonic or n-monotonic where $n = \sum n_\varphi$ for all monotonic $\varphi \in \Sigma$.*

PROOF. If Σ has an implied dependency φ, the monotonicity of Σ is identical with that of $\Sigma - \{\varphi\}$. So, without loss of generality, we assume that Σ does not contain any implied dependencies. If Σ has monotonic dependencies only, the statement of the lemma follows from Lemma 11. Otherwise, the only sources of nonmonotonicity are the dependencies φ satisfying the premise of Lemma 5. Let $\varphi \in \Sigma$ be such a nonmonotonic dependency. Further, let $\Sigma_r = \Sigma - \{\varphi\}$. Since Σ does not contain implied dependencies, thus $\Sigma_r \not\models \varphi$.

To show nonmonotonicity of Σ, we construct an unbounded chain of structures similarly to how this is done in Lemma 5, but using a modified mechanism that preserves the truth value of Σ_r as an invariant in each structure of the chain.

Let A be a structure such that $A \not\models \varphi$, $A \models \Sigma_r$. We construct a structure B such that $B \models \Sigma$, $A \subset B$ by chasing the constraints in Σ, such that the values for existentially quantified variables are drawn from the existing constants in A. Since A is finite, the process terminates, yielding a finite $B \models \Sigma$. The chase adds at least one new tuple to some relation in B to make φ true, hence $A \subset B$.

Now, let B be a structure such that $B \models \Sigma$. Since $\Sigma_r \not\models \varphi$, there exists a structure F_0 such that $F_0 \not\models \varphi$ and $F_0 \models \Sigma_r$. We construct $A \not\models \varphi$ as in Lemma 7 for each B using this fixed F_0. Since $F_0 \models \Sigma_r$, the $A \models \Sigma_r$.

Together, these constructions witness the nonmonotonicity of Σ. □

Now we are ready to state the main result of this section. The case analysis in the following theorem is based on Lemma 10.

THEOREM 10.

(1) *Each FullTGD dependency is* 0*-monotonic or nonmonotonic.*

(2) *Each FullD dependency is* 1*-monotonic or nonmonotonic.*

(3) *Each TGD dependency is* 1*-monotonic or nonmonotonic.*

(4) *Each ED dependency is* 2*-monotonic or nonmonotonic.*

PROOF.

(1) Follows immediately from Lemma 5.

(2) Let $\varphi \in$ FullD. Suppose that $\varphi \notin$ FullTGD. Then, φ is equivalent to a set $\Sigma = \{\psi_1, \psi_2\}$ of constraints, where $\psi_1 \in$ FullTGD and ψ_2 is a generalized equality-generating dependency (possibly with multiple equality atoms in the conclusion). Moreover, ψ_1 is 0-monotonic or nonmonotonic. By Lemma 9, ψ_2 is 1-monotonic. Hence, by Lemmas 11 and 12, φ is 1-monotonic or nonmonotonic.

(3) Let $\varphi \in$ TGD. Then, three cases are possible: (a) $\varphi \in$ FullTGD, (b) φ is an inclusion of Boolean conjunctive queries, or (c) φ is equivalent to $\{\psi_1, \psi_2\}$, where $\psi_1 \in$ FullTGD and ψ_2 is an inclusion of Boolean conjunctive queries. In case (a), φ is 0-monotonic or nonmonotonic. In case (b), φ is 1-monotonic, by Lemma 8. In case (c), φ is 1-monotonic or nonmonotonic by Lemmas 9, 11, and 12. Hence, φ is 1-monotonic or nonmonotonic.

(4) Let $\varphi \in$ ED. Suppose that $\varphi \notin$ TGD. Then, φ is equivalent to a set $\Sigma = \{\psi_1, \psi_2\}$ of constraints, where $\psi_1 \in$ TGD and ψ_2 is a generalized equality-generating dependency. Here, ψ_1 is 1-monotonic or nonmonotonic, and ψ_2 is 1-monotonic. Hence, by Lemmas 11 and 12, φ is 2-monotonic or nonmonotonic. □

Example 13. To illustrate a 2-monotonic ED-dependency, consider $\varphi = \forall x, y(R(x, y) \rightarrow \exists z(S(z), x = y))$. The chain of structures $A_0 = (\{(a, a)\}, \emptyset)$, $A_1 = (\{(a, a)\}, \{b\})$, $A_2 = (\{(a, a), (a, b)\}, \{b\})$ contains two negative segments.

As an application of the inexpressibility tools presented in this section, we complete the proof of Theorem 8 from Section 6.

PROOF (INEXPRESSIBILITY CLAIM OF THEOREM 8). Let $[k] = \{1, \ldots, k\}$ and let φ_Z be a constraint of the form

$$R_0(x), R_Z(x) \rightarrow \exists y\, T_Z(y),$$

where $Z \subseteq [k]$, $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$, and $T_Z(x) := \bigwedge_{i \in Z} T_i(x)$.

Let $\Sigma = \{\varphi_Z : \emptyset \neq Z \subseteq [k]\}$. Each dependency $\varphi_Z \in \Sigma$ specifies inclusion of Boolean conjunctive queries and is hence 1-monotonic by Lemma 8. Moreover, Σ contains a total of $2^k - 1$ dependencies. Therefore, by Lemma 11, Σ is at most $(2^k - 1)$-monotonic.

We show that Σ is strictly $(2^k - 1)$-monotonic. Let Z_c denote the subset of $[k]$, where index c is the binary encoding of Z, namely, $c = \sum_{j \in Z} 2^j$. Then, $Z_0, \ldots, Z_{2^k}$ is an enumeration of subsets of $[k]$. The enumeration has the property that $Z_{c_2} \not\subseteq Z_{c_1}$ for any $c_1 < c_2$. We construct a chain of structures for Σ that contains $2^k - 1$ negative segments by induction.

- —Let A_0 be the empty structure for Σ, $A_0 \models \Sigma$.
- —Let $A_c \models \Sigma$ be a structure obtained in induction step c. Construct B_{c+1} by copying all relations from A_c and adding the constant c to relation R_0 and all relations R_i such that $i \in Z_{c+1}$. The added constant c violates exactly one dependency $\varphi_{Z_{c+1}} \in \Sigma$. No other dependency gets violated, since $Z_{c_{i+1}} \not\subseteq Z_j$ for any $j \leq c$. We have $B_{c+1} \not\models \Sigma$, $B_{c+1} \supset A_c$.
- —Let $B_c \not\models \Sigma$ be a structure obtained in induction step c, $c \geq 1$. Construct A_c by copying all relations from B_c and setting $T_i := R_i$ for all $i \in [k]$. This construction makes all dependencies of Σ satisfied in A_c. Hence, $A_c \models \Sigma$, $A_c \supset B_c$.

The constructed chain $B_1, A_1, \ldots, A_{2^k-1}, B_{2^k}$ witnesses that Σ is strictly $(2^k - 1)$-monotonic. By Theorem 10, each TGD-dependency is 2-monotonic or nonmonotonic. Therefore, by Lemmas 11 and 12, at least $\lceil (2^k - 1)/2 \rceil = 2^{k-1}$ embedded dependencies are needed to express Σ. By the same theorem, Σ is not expressible by any set of full dependencies. □

8. SEMANTICS OF SKED CONSTRAINTS

In this section we examine the semantics of SkED constraints. The semantics of SkED are somewhat special, but seem to be needed to obtain domain independence (Example 14). These semantics have already been discussed in Fagin et al. [2005], but we cover several additional aspects here.

We introduce another fragment of SO, ∃SOED which has the standard second-order semantics. We show that source-to-target SkED-mappings are also source-to-target ∃SOED-mappings (Theorem 11).[5] However, this translation may incur an exponential size increase.

We first discuss the semantics of SkED constraints. The main question is: What is the universe from which the functions can take values? In other words, what is their allowed range? Intuitively, the problem is with the universe of the existentially quantified intermediate database.

Example 14. Consider the FullTGD-mappings m_{12}^k and m_{23}^k given by $(\sigma_1, \sigma_2^k, \Sigma_{12}^k)$ and $(\sigma_2^k, \sigma_3, \Sigma_{23}^k)$, where

$$\begin{array}{lrcl}
\hline
\Sigma_{12}^k \text{ is} & R(x) & \to & \exists y\, S_i(y) \\
\Sigma_{23}^k \text{ is} & S_i(x), S_j(x) & \to & T(x) \\
\hline
\end{array}$$

for $1 \leq i, j \leq k$ and $i \neq j$, where $\sigma_1 = \{R\}$, $\sigma_2^k = \{S_1, \ldots, S_k\}$, and $\sigma_3 = \{T\}$.

Consider the case where $R = \{1, \ldots, k-1\}$ in A and T is empty in C. First, notice that $(A, C) \in m_{12}^k \circ m_{23}^k$, as witnessed by the database B where $S_i = \{i\}$. Skolemizing and composing the aforementioned constraints, we obtain Σ_{13}^k given by the set of constraints

$$\{R(x), R(y),\ f_i(x) = z,\ f_j(y) = z \to T(z) : 1 \leq i, j \leq k, i \neq j\},$$

[5] When restricted to structures with at least two elements in the active domain.

where f_i is the Skolem function corresponding to $\exists y\, S_i(y)$. If we restrict the range of every f_i to fall within the domain of A and C, then one of the preceding constraints must fail, as follows. Consider the case where $x = 1$ and $y = 1$. Then the set $\{f_i(1) : 1 \leq i \leq k\}$ must be a subset of $\{1, \ldots, k-1\}$. By the pigeonhole principle, there must be i and j such that $i \neq j$ and $f_i(1) = f_j(1)$. Then the constraint corresponding to such i, j fails for (A, C), since T is empty in C. Therefore, $(A, C) \not\models \Sigma_{13}^k$. On the other hand, if we keep the same relations R and T, but allow the domain to have at least k values, then we have $(A, C) \models \Sigma_{13}^k$ witnessed by setting $f_i(x) = i$ for all i, x.

This shows that restricting the range of the Skolem functions to be domain of the input structures may yield domain-dependent formulas, even though they satisfy the safety conditions. Certainly, no such domain-dependent formulas can express the composition, since whether (R, T) belong to the composition does not depend on their domains.

Therefore, we require all databases to be finite (i.e., all relations are finite), but to have an implicit countably infinite universe. Notice that no finite domain would work for all constraints, since the previous example gives a family of sets of constraints for which the meaning changes, depending on whether the domain has size less than k. We allow the functions to take any values from this implicit universe.

Since the semantics of SkED are special, it is natural to ask whether the constraints in SkED can be expressed in some fragment of ∃SO under the usual second-order semantics. We show that this is possible for a source-to-target SkED.

Theorem 11. *Every finite set of source-to-target SkED constraints (under the semantics described earlier) is equivalent to a finite set of source-to-target ∃SOED constraints (under the usual second-order semantics) when restricted to instances with at least two elements.*

Proof (Outline). In the case of source-to-target constraints, we know that we do not have "recursive" Skolem terms. Phrased differently, there are no Skolem terms of the form $f(\ldots f \ldots)$ (directly, or indirectly through equalities). Therefore, there is a finite number of values which we can refer to by building Skolem terms on top of the elements of the domain. Intuitively, these are all the elements that the intermediate database needs to have and the worst case is when they are all different. If the domain has n elements and we have p Skolem functions of arity q, then an easy upper bound on the number of elements we can refer to is $\leq n^{(p+q)^p}$ whenever $n \geq 2$ (this is shown by induction depth of the Skolem terms; at each step, we go from $m \geq n$ possible values to

$$m + pm^q \leq (p+1)m^q \leq 2^p m^q \leq n^p m^q \leq m^{(p+q)}$$

possible values). Therefore, we can encode all these values with tuples of arity $r = (p+q)^p$. We encode every value c from the original domain as the tuple $(c, \ldots, c)$; that is, c repeated r times.

Given a finite set Σ of source-to-target SkEDs (which we assume without loss of generality to be in unnested form), we first compute r, then transform each constraint $\phi \in \Sigma$ by replacing every occurrence of an equation of the form $f(\bar{x}) = y$ with $F(\bar{x}, \bar{y})$, where $\bar{y}$ is a tuple of arity r. We also replace y with $\bar{y}$

everywhere except in relational atoms. To every relational atom, we add a set of equalities of the form $y = y_1, \dots, y = y_k$, which we abbreviate as $y = \bar{y}$. Finally, we add constraints of the form

$$\begin{gathered} \dots \to \exists \bar{y}\, F(\bar{x}, \bar{y}) \\ F(\bar{x}, \bar{y}), F(\bar{x}, \bar{y}') \to \bar{y} = \bar{y}', \end{gathered}$$

where ... are obtained from any premises which mention f. Notice that we need both equalities and FO existential quantifiers in the conclusions and that we may incur an exponential increase in size, since we need r to be exponential in p. □

Notice that the proof only requires that we do not have "recursive" Skolem terms. Source-to-target is a strong condition that ensures this, but weaker conditions on the set of constraints $\Sigma_{12} \cup \Sigma_{23}$ suffice. For example, it is enough to require that $\Sigma_{12} \cup \Sigma_{23}$ have stratified witnesses (see Deutsch and Tannen [2003] and Fagin et al. [2003a]). When such conditions hold, we can compose ∃SOED-mappings using a technique similar to that of the proof of Theorem 11. In this case, we do not Skolemize, but replace every relation S of arity s from σ_2 with an existentially quantified relation R of arity rs (where r is as in the proof of Theorem 11). Then, we replace every occurrence of a universally quantified variable v in S with $\bar{v}$ and add the equation $v = \bar{v}$ and replace every occurrence of an existentially quantified variable u with $\bar{u}$. This gives an algorithm for composition of source-to-target ∃SOED-mappings.

9. OTHER BASIC OPERATORS

In addition to composition, we are interested in several other basic operators, including domain, range, intersection, cross-product, and inverse. These operators take as input mappings and models and give as output mappings or models (a *model* is a set of instances). The following table summarizes the definitions of these basic operators:

$$\begin{aligned}
\mathrm{dom}(m) &:= \{A : \exists B\, \langle A, B\rangle \in m\}.\\
\mathrm{rng}(m) &:= \{B : \exists A\, \langle A, B\rangle \in m\}.\\
\mathcal{A} \cap \mathcal{B} &:= \{A : A \in \mathcal{A}, A \in \mathcal{B}\}.\\
\mathrm{id}(\mathcal{A}) &:= \{\langle A, A\rangle : A \in \mathcal{A}\}.\\
\mathcal{A} \times \mathcal{B} &:= \{\langle A, B\rangle : A \in \mathcal{A}, B \in \mathcal{B}\}.\\
m_1 \cap m_2 &:= \{\langle A, B\rangle : \langle A, B\rangle \in m_1, \langle A, B\rangle \in m_2\}.\\
m^{-1} &:= \{\langle B, A\rangle : \langle A, B\rangle \in m\}.
\end{aligned}$$

As in the case of mappings, we say that a model $\mathcal{A}$ is given by (σ_1, Σ_1) if it consists exactly of those databases over the signature σ_1 which satisfy the constraints Σ_1. If furthermore, Σ_1 is finite subset of $\mathcal{L}$, we say that $\mathcal{A}$ is an $\mathcal{L}$-model. As in the case of composition, we say that $\mathcal{L}$ is closed under one of these operators if it produces an $\mathcal{L}$-model or $\mathcal{L}$-mapping whenever the inputs are compatible $\mathcal{L}$-models or $\mathcal{L}$-mappings.

PROPOSITION 4. *Every $\mathcal{L} \supseteq$ FullTGD is closed under identity, cross-product, and intersection.*

PROOF. If m_{12} and m_{34} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_3, \sigma_4, \Sigma_{34})$ and $\mathcal{A}$ and $\mathcal{B}$ are given by (σ_1, Σ_1) and (σ_2, Σ_2), then

—$\mathcal{A} \times \mathcal{B}$ is given by $(\sigma_1, \sigma_2, \Sigma_1 \cup \Sigma_2)$.
—$\mathcal{A} \cap \mathcal{B}$ is given by $(\sigma_1, \Sigma_1 \cup \Sigma_2)$ (here $\sigma_1 = \sigma_2$).
—$m_{12} \cap m_{34}$ is given by $(\sigma_1, \sigma_2, \Sigma_{12} \cup \Sigma_{34})$ (here $\sigma_1 = \sigma_3$ and $\sigma_2 = \sigma_4$).

To express identity, we need to refer to the third auxiliary signature σ_2' (which we normally ignore) which contains, for every relation symbol R in σ_2, a relation symbol R' of the same arity. In this case, $\sigma_1 = \sigma_2$ so $\sigma_2' = \sigma_1'$.

—id($\mathcal{A}$) is given by $(\sigma_1, \sigma_1, \sigma_1', \Sigma_1 \cup \Sigma)$, where Σ consists of two constraints of the form $\forall(\bar{x})(R(\bar{x}) \rightarrow R'(\bar{x}))$, $\forall(\bar{x})(R'(\bar{x}) \rightarrow R(\bar{x}))$ for every R in σ_1. □

PROPOSITION 5. *Each one of the operators composition, range, and domain can be reduced to any one of the others.*

PROOF. If

—m_{12} is given by $(\sigma_1, \sigma_2, \Sigma_{12})$, m_{23} is given by $(\sigma_2, \sigma_3, \Sigma_{23})$,
—m_1 is given by $(\sigma_1 \cup \sigma_3, \sigma_2, \Sigma_{12} \cup \Sigma_{23})$, m_2 is given by $(\sigma_2, \sigma_1 \cup \sigma_3, \Sigma_{12} \cup \Sigma_{23})$,
—dom(m_1) is given $(\sigma_1 \cup \sigma_3, \Sigma_1)$, and rng($m_2$) is given $(\sigma_1 \cup \sigma_3, \Sigma_2)$,

then $m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_1)$ and $(\sigma_1, \sigma_3, \Sigma_2)$. Conversely, if

—m_{12} is given by $(\sigma_1, \sigma_2, \Sigma_{12})$, m_{21} is given by $(\sigma_2, \sigma_1, \emptyset)$,
—$m_{12} \circ m_{21}$ is given by $(\sigma_1, \sigma_1, \Sigma_1)$, and $m_{21} \circ m_{12}$ is given by $(\sigma_2, \sigma_2, \Sigma_2)$,

then dom(m_{12}) and rng(m_{12}) are given, respectively, by (σ_1, Σ_1) and (σ_2, Σ_2). □

Proposition 5 and Theorem 2 give the following.

COROLLARY 2. *Checking whether the domain or range of a FullTGD-mapping is a FullTGD-model is undecidable.*

All the languages we consider satisfy the premises of Proposition 4. Therefore, Proposition 5 indicates that we can concentrate our attention on closure under composition and inverse. Notice that if an $\mathcal{L}$-mapping m is given by $(\sigma_1, \sigma_2, \Sigma_{12})$, then its inverse is given by $(\sigma_2, \sigma_1, \Sigma_{12})$, which is, of course, easy to compute. However, the restrictions on $\mathcal{L}$ may be such that the second expression no longer gives an $\mathcal{L}$-mapping. For example, this happens with source-to-target constraints. This is why we seek restrictions on $\mathcal{L}$ which are symmetric with respect to the input and output signatures and which guarantee closure under composition.

10. CONCLUSIONS

Mapping composition is one of the key operators used for manipulating schemas and mappings between schemas. We studied composition of mappings given by embedded dependencies, which are expressive enough for many data management applications. We addressed challenges that were not considered in prior work, in particular, those due to recursion and de-Skolemization.

ACKNOWLEDGMENTS

We are grateful to Ron Fagin, Phokion Kolaitis, Lucian Popa, and Wang-Chiew Tan for fruitful discussions on mapping composition and for their feedback on a preliminary version of this article. We thank Andreas Blass, who provided the formula ψ in the proof of Theorem 1 and contributed to Proposition 5. We thank anonymous referees for helpful improvement suggestions.

REFERENCES

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley.

BERNSTEIN, P., GREEN, T. J., MELNIK, S., AND NASH, A. 2006. Implementing mapping composition. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*.

BERNSTEIN, P. A. 2003. Applying model management to classical meta-data problems. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. 209–220.

BERNSTEIN, P. A., HALEVY, A. Y., AND POTTINGER, R. 2000. A vision of management of complex models. *SIGMOD Rec. 29*, 4, 55–63.

DEUTSCH, A. AND TANNEN, V. 2003. Reformulation of XML queries and constraints. In *Proceedings of the International Conference on Database Theory (ICDT)*.

EBBINHAUS, H.-D. AND FLUM, J. 1999. *Finite Model Theory*. Springer Verlag.

FAGIN, R. 1975. Monadic generalized spectra. *Zeitschr. Math. Logik Grundlagen Math. 21*, 89–96.

FAGIN, R. 1977. Multivalued dependencies and a new normal form for relational databases. *ACM Trans. Database Syst. 2*, 3, 262–278.

FAGIN, R. 1982. Horn clauses and database dependencies. *J. ACM 29*, 4, 952–985.

FAGIN, R. 2006. Inverting schema mappings. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 50–59.

FAGIN, R., KOLAITIS, P. G., MILLER, R. J., AND POPA, L. 2003a. Data exchange: Semantics and query answering. In *Proceedings of the International Conference on Database Theory (ICDT)*. 207–224.

FAGIN, R., KOLAITIS, P. G., AND POPA, L. 2003b. Data exchange: Getting to the core. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 90–101.

FAGIN, R., KOLAITIS, P. G., POPA, L., AND TAN, W. C. 2004. Composing schema mappings: Second-Order dependencies to the rescue. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 83–94.

FAGIN, R., KOLAITIS, P. G., POPA, L., AND TAN, W. C. 2005. Composing schema mappings: Second-Order dependencies to the rescue. *ACM Trans. Database Syst. 30*, 994–1055.

HALEVY, A. Y., IVES, Z. G., SUCIU, D., AND TATARINOV, I. 2003. Schema mediation in peer data management systems. In *Proceedings of the International Conference on Data Engineering (ICDE)*.

KLUG, A. AND PRICE, R. 1982. Determining view dependencies using tableaux. *ACM Trans. Database Syst. 7*, 361–380.

KOCH, C. 2002. Query rewriting with symmetric constraints. In *Proceedings of the Foundations of Information and Knowledge Systems (FoIKS)*.

LENZERINI, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*. 233–246.

MADHAVAN, J. AND HALEVY, A. Y. 2003. Composing mappings among data sources. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*. 572–583.

MELNIK, S. 2004. Generic model management: Concepts and algorithms. Ph.D. Thesis, University of Leipzig. Lecture Notes in Computer Science, vol. 2967. Springer Verlag.

MELNIK, S., BERNSTEIN, P. A., HALEVY, A. Y., AND RAHM, E. 2005. Supporting executable mappings in model management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 167–178.

MELNIK, S., RAHM, E., AND BERNSTEIN, P. A. 2003. Rondo: A programming platform for generic model management. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 193–204.

NICOLAS, J. M. 1987. First order logic formalization for function, multivalued, and mutual dependencies. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 40–46.

SAGIV, Y. AND YANNAKAKIS, M. 1980. Equivalences among relational expressions with the union and difference operators. *J. ACM 27*, 4, 633–655.

SIPSER, M. 1997. *Introduction to the Theory of Computation*, 2nd ed. PWS Publishing.

STONEBRAKER, M. 1975. Implementation of integrity constraints and views by query modification. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 65–78.

YU, C. AND POPA, L. 2004. Constraint-Based XML query rewriting for data integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 371–382.

Received October 2005; revised May 2006; accepted August 2006