Journal of Computer and System Sciences 79 (2013) 763–784

Contents lists available at SciVerse ScienceDirect

Journal of Computer and System Sciences

www.elsevier.com/locate/jcss

The language of plain SO-tgds: Composition, inversion and structural properties

Marcelo Arenas[a], Jorge Pérez[b,*], Juan Reutter[c], Cristian Riveros[d]

[a] Computer Science Department, Pontificia Universidad Católica de Chile, Chile
[b] Computer Science Department, Universidad de Chile, Chile
[c] School of Informatics, University of Edinburgh, United Kingdom
[d] Department of Computer Science, University of Oxford, United Kingdom

ARTICLE INFO

Article history:
Received 6 January 2012
Received in revised form 10 April 2012
Accepted 16 January 2013
Available online 21 January 2013

Keywords:
Schema mappings
Inverse
Composition
Plain SO-tgds
Data exchange

ABSTRACT

The problems of composing and inverting schema mappings specified by source-to-target tuple-generating dependencies (st-tgds) have attracted a lot of attention, as they are of fundamental importance for the development of Bernstein's metadata management framework. In the case of the composition operator, a natural semantics has been proposed and the language of second-order tuple generating dependencies (SO-tgds) has been identified as the right language to express it. In the case of the inverse operator, several semantics have been proposed, most notably the maximum recovery, the only inverse notion that guarantees that every mapping specified by st-tgds is invertible. Unfortunately, less attention has been paid to combining both operators, which is the motivation of this paper. More precisely, we start our investigation by showing that SO-tgds are not good for inversion, as there exist mappings specified by SO-tgds that are not invertible under any of the notions of inversion proposed in the literature. To overcome this limitation, we borrow the notion of CQ-composition, which is a relaxation obtained by parameterizing the composition of mappings by the class of conjunctive queries (CQ), and we propose a restriction over the class of SO-tgds that gives rise to the language of *plain* SO-tgds. Then we show that plain SO-tgds are the right language to express the CQ-composition of mappings given by st-tgds, in the same sense that SO-tgds are the right language to express the composition of st-tgds, and we prove that every mapping specified by a plain SO-tgd admits a maximum recovery, thus showing that plain SO-tgds have a good behavior w.r.t. inversion. Moreover, we show that the language of plain SO-tgds shares some fundamental structural properties with the language of st-tgds, but being much more expressive, and we provide a polynomial-time algorithm to compute maximum recoveries for mappings specified by plain SO-tgds (which can also be used to compute maximum recoveries for mappings given by st-tgds). All these results suggest that the language of plain SO-tgds is a good alternative to be implemented in data exchange and data integration applications.

© 2013 Elsevier Inc. All rights reserved.

* Corresponding author.
E-mail addresses: marenas@ing.puc.cl (M. Arenas), jperez@dcc.uchile.cl (J. Pérez), juan.reutter@ed.ac.uk (J. Reutter), cristian.riveros@cs.ox.ac.uk (C. Riveros).

0022-0000/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jcss.2013.01.002

1. Introduction

A schema mapping is a specification that describes how data from a source schema is to be mapped to a target schema. These specifications have been essential for several data-interoperability tasks such as data exchange [12] and data integration [20]. However, the need for manipulating these artefacts has been recently pointed out in the literature [6,23,24,7]. Thus, schema mappings have become an object of study as first class citizens in a framework where they are expressed in a declarative form and high level algebraic operators are used to manipulate them [6].

Consequently, the study of algebraic operators between schema mappings has become crucial for metadata management. Following Bernstein's framework [6], most of the work in metadata management has focused on the definition of the semantics of algebraic operators that are used to manipulate schema mappings. Among the most important schema mapping operators, we find the composition and inverse operators. Both have been studied thoroughly, resulting on a clear understanding on the definability of these operators, and the existence of mappings satisfying the definition of these operators, for a wide variety of classes of schema mappings [14,10,15,5,3]. The basis of this work has invariantly been mappings that are defined by sets of source-to-target tuple-generating dependencies (st-tgds), the most commonly used mapping language for data exchange and integration [12,20]. Nevertheless, less attention has been paid to schema mapping languages that are good for both operators, composition and inversion. In this paper, we explore the relationship between good mapping languages that extend st-tgds and have good properties regarding both composition and inversion.

In [14], Fagin et al. show that the language of SO-tgds, an extension of st-tgds with second-order existential quantification, is closed under composition and prove several results that show that it is the right language to compose mappings. In particular, they show that SO-tgds are the *smallest* language capable of expressing the composition of st-tgds, since for every SO-tgd λ one can find a sequence of st-tgds such that their composition is specified precisely by λ. Despite the good properties of SO-tgds with respect to composition, none of the notions of inverse proposed for schema mappings [10,15,5,3] have been considered together with the composition of schema mappings, that is, for the case of SO-tgds. In this paper, we show that, unfortunately, SO-tgds are not appropriate for our study; there exist mappings specified by SO-tgds that have no CQ-maximum recoveries, which is the weakest notion of inverse defined in the literature [4], implying that there exist mappings specified by SO-tgds that are not suitable for inversion under any of the notions of inverse proposed until now [3]. Thus, although the language of SO-tgds is the right language for composition, its behavior is not the ideal regarding inversion. Moreover, since every SO-tgd is known to represent a composition of a finite number of st-tgds, we conclude that, in general, the composition of st-tgds cannot be inverted under any of the notions of inverse proposed in the literature.

To overcome the limitations of SO-tgds with respect to inversion, we borrow the notion of composition w.r.t. conjunctive queries (CQ-composition), introduced by Madhavan and Halevy [22] and show that the CQ-composition of any number of st-tgds is guaranteed to have a maximum recovery. To do this, we propose to study the language of *plain* SO-tgds, a slightly restricted version of SO-tgds. In particular, we show that this language can specify the CQ-composition of st-tgds and, moreover, that every mapping given by plain SO-tgds is invertible when the notion of inverse considered is the notion of maximum recovery [5].

The good behavior of plain SO-tgds for combining composition and inversion raises questions regarding the possibility that this mapping language behaves just as good when performing other data interoperability tasks. In order to further study this language, we follow the approach of ten Cate and Kolaitis [26,27], and analyze the *structural properties* of plain SO-tgds, comparing them with the properties of other common mapping languages such as st-tgds and SO-tgds. We show that, in terms of ten Cate and Kolaitis taxonomy [26], the behavior of plain SO-tgds can be considered *closer* to st-tgds than to SO-tgds, in the sense that plain SO-tgds share with st-tgds all the properties of mapping languages that were noted in [27] as crucial for data exchange and data integration; plain SO-tgds *allow for rewriting of conjunctive queries*, they always *admit universal solutions*, and are *closed under target homomorphisms*. We also show that plain SO-tgds are strictly more expressive than other well-behaved mapping languages such as st-tgds and the language of nested tgds defined in [19]. It is interesting to note that plain SO-tgds are, to the best of our knowledge, the most expressive mapping language that enjoys all three aforementioned properties. This gives a negative answer to the question raised in [26] of whether all mappings that allow for conjunctive query rewriting, admit universal solutions, and are closed under target homomorphisms can be specified with nested tgds. Therefore, this allows us to restate this question by putting the language of plain SO-tgds as the main candidate for a positive answer.

Towards the end of the paper we show our most interesting algorithmic result regarding inversion; we provide the first polynomial-time algorithm to compute maximum recoveries of mapping specified by plain SO-tgds. Specifically, given a mapping $\mathcal{M}$ specified by a set of plain SO-tgds, our algorithm returns a maximum recovery of $\mathcal{M}$ specified in a language that extends the class of plain SO-tgds. This result is interesting in its own right since our algorithm is the first polynomial-time algorithm for inverting schema mappings and, in particular, for inverting mappings given by st-tgds in polynomial time. It should be noticed that it is open whether the language needed to specify the maximum recovery of a plain SO-tgds is also invertible or closed under composition. However, the study of plain SO-tgds opens new possibilities for studying closure properties of the composition and inversion of st-tgds, which we believe are interesting problems for future research.

The rest of the paper is organized as follows. In Section 2, we give the basic notation used in the paper, while in Section 3 we show the negative result regarding SO-tgds, composition and inversion. In Section 4, we present the mapping language of plain SO-tgds as a solution to the problem of combining composition with inversion, and show its good structural properties.

In Section 5, we provide a polynomial-time algorithm to compute maximum recoveries for mappings given by plain SO-tgds. Finally, we give some concluding remarks in Section 6.

Before continuing with the paper, we describe the new material in this article, compared with our previous conference paper [4], and discuss some related work.

1.1. New material in this paper

Preliminary versions of some of the results in this paper appeared in [4]. Nevertheless, this paper contains substantial new material. The negative result regarding the inversion of SO-tgds (presented in Section 3) is new and was informally mentioned in [4]. All the results on the structural properties of plain SO-tgds and, in particular, the relationship with the work by ten Cate and Kolaitis [27] were not included in [4] and are presented in this paper for the first time. These results are included in Section 4. Besides, in this paper we include new examples and also detailed proofs which are not included in [4].

1.2. Related work

In the recent years, there has been a lot of work about the composition and inversion of schema mappings [23,14,10,15,5,4,3,17,1,16,2]. As most of the investigations in data exchange, data integration and schema mapping management, we make the assumption that source instances contain only constant values, while target instances contain constant and null values, the latter to represent missing (incomplete) information (see Section 2 for a formalization of our setting).

While almost no research has been made regarding the inversion of SO-tgds in this scenario, there have been at least two investigations that relax the assumption mentioned above in order to study schema mapping operators. In [16], the authors study a setting in which both source and target instances contain null values, and make the case that inverses should be studied in this *symmetric* setting. Similarly, in [2] the authors study the problem of exchanging incomplete information in a more general setting not only including nulls in the source data, but also considering general representation systems to exchange data. In [2], the authors prove that under this new expressive semantics for mappings in which instances are allowed to specify incomplete information in a general way, mappings specified by SO-tgds always have inverses under the notion of maximum recovery. Nonetheless, neither [16] nor [2] provide algorithms for computing inverses of mappings specified by SO-tgds, thus leaving open the issue of computing inverses of mappings that result from composing st-tgds. In this paper, we follow an alternative approach. Instead of making mappings more expressive (by adding incomplete information in source and target instances), we study the weaker language of plain SO-tgds under the typical semantics in which source instances only contain constant values. We show that mappings given by plain SO-tgds always admit inverses under the notion of maximum recovery, and also provide an efficient algorithm to compute maximum recoveries of mappings specified in this language. This approach can be considered complementary to the approaches proposed in [16,2], as we expect that our results on the classical scenario of data exchange can also be useful in the future research on mappings for databases with incomplete information.

Regarding algorithms for efficiently inverting mappings given by st-tgds, Fagin et al. presented in [16] a polynomial-time algorithm to compute inverses for such mappings in the scenario where incomplete information is considered in both source and target instances. The algorithm proposed in [16] follows a similar approach to ours, using second-order quantification to efficiently express inverses.

The previous works on inversion of schema mappings have mainly focused on inverting mappings specified by st-tgds. The first of these works was by Fagin [10], where the author proposes a notion that was rather restrictive [15] as most mappings specified by st-tgds do not admit an inverse under this notion. Subsequent works [15,2,4] have searched for more relaxed notions of inversion. This approach has been successful for mappings specified by st-tgds, and the literature now provides notions of inversion under which every mapping specified by st-tgds is invertible [2,4,3]. One of the motivations of our work is the need of integrating inversion and composition, i.e. the need of inverting the output of the composition of two mappings specified by st-tgds. Given that SO-tgds are not invertible under any of the main notions proposed so far in the literature (see Theorem 6 and Corollary 7), one possible path is to search for an even weaker notion for inverting SO-tgds. To the best of our knowledge there is no investigation in the literature following this last approach. In this paper, we have decided to follow a different approach as instead of weakening the notion of inversion, we have weakened the notion of composition by considering the CQ-composition. Since we prove that CQ-composition can always be represented in a less expressive mapping language, the language of plain SO-tgds, our approach to investigate the interplay between composition and inversion amounts to investigate of the inversion of plain SO-tgds, which is one of the main topics considered in this paper.

2. Preliminaries

A *schema* $\mathbf{R}$ is a finite set $\{R_1, \ldots, R_k\}$ of relation symbols, with each R_i having a fixed arity $n_i \geqslant 1$. Let $\mathbf{D}$ be a countably infinite domain. An instance I of $\mathbf{R}$ assigns to each relation symbol R_i of $\mathbf{R}$ a finite n_i-ary relation $R_i^I \subseteq \mathbf{D}^{n_i}$. The *domain* $\mathrm{dom}(I)$ of instance I is the set of all elements that occur in any of the relations R_i^I. $\mathrm{Inst}(\mathbf{R})$ is defined to be the set of all instances of $\mathbf{R}$. If a tuple $\bar{a}$ belongs to R_i^I, we say that $R_i(\bar{a})$ is a *fact* in I. We sometimes describe an instance as a set of facts.

As is customary in the data exchange literature, we consider instances with two types of values: constants and nulls [12,10,15]. More precisely, let $\mathbf{C}$ and $\mathbf{N}$ be infinite and disjoint sets of constants and nulls, respectively, and assume that $\mathbf{D} = \mathbf{C} \cup \mathbf{N}$. If we refer to a schema $\mathbf{S}$ as a *source* schema, then $\text{Inst}(\mathbf{S})$ is defined to be the set of all instances of $\mathbf{S}$ that are constructed by using only elements from $\mathbf{C}$, and if we refer to a schema $\mathbf{T}$ as a *target* schema, then instances of $\mathbf{T}$ are constructed by using elements from both $\mathbf{C}$ and $\mathbf{N}$. Source instances are instances of a source schema, and target instances are instances of a target schema.

2.1. Schema mappings and universal solutions

We use a general representation of mappings; given two schemas $\mathbf{R}_1$ and $\mathbf{R}_2$, a mapping $\mathcal{M}$ from $\mathbf{R}_1$ to $\mathbf{R}_2$ is a set of pairs (I, J), where I is an instance of $\mathbf{R}_1$, and J is an instance of $\mathbf{R}_2$. Our results are mainly focused on a special class of mappings that we call *source-to-target* mappings (st-mappings). A mapping $\mathcal{M}$ from $\mathbf{R}_1$ to $\mathbf{R}_2$ is an *st-mapping*, if $\mathbf{R}_1$ is a source schema (that is, instances of $\mathbf{R}_1$ are constructed using only elements from $\mathbf{C}$) and $\mathbf{R}_2$ is a target schema (that is, instances of $\mathbf{R}_2$ are constructed by using elements from $\mathbf{C}$ and $\mathbf{N}$). Unless otherwise noted, we always assume that mappings are st-mappings. Thus, we often abuse the notation and speak of $\mathbf{R}_1$ as the *source schema* of $\mathcal{M}$ and $\mathbf{R}_2$ as the *target schema* of $\mathcal{M}$. Likewise, when $\mathcal{M}$ is understood from context, we refer to instances of $\mathbf{R}_1$ as *source instances*, and instances of $\mathbf{R}_2$ as *target instances*. The class of *target-to-source* mappings, or ts-mappings, is defined analogously.

Given a mapping $\mathcal{M}$, we say that J is a *solution for I under $\mathcal{M}$*, if (I, J) is in $\mathcal{M}$. We denote by $\text{Sol}_{\mathcal{M}}(I)$ the set of all solutions for I under $\mathcal{M}$. A special class of solutions, denoted as *universal solutions*, has been pointed out as one of the preferred classes of solutions to materialize in the data exchange scenario. To define them, we need the notion of homomorphisms. Given instances J_1 and J_2 of the same schema, a *homomorphism* h from J_1 to J_2 is a function that is the identity over constants ($h(a) = a$ for every $a \in \mathbf{C}$) maps null values to null or constant values, and for every fact $R(a_1, \ldots, a_n)$ in J_1, it holds that $R(h(a_1), \ldots, h(a_k))$ is a fact in J_2. If there exist homomorphisms from J_1 to J_2 and from J_2 to J_1, then we say that J_1 and J_2 are homomorphically equivalent. Then, an instance J is said to be a universal solution for an instance I under a mapping $\mathcal{M}$ if $J \in \text{Sol}_{\mathcal{M}}(I)$ and for every $J' \in \text{Sol}_{\mathcal{M}}(I)$, there exists a homomorphism from J to J' [12].

2.2. Queries and certain answers

A k-ary query Q over a schema $\mathbf{R}$, with $k \geqslant 0$, is a function that maps every instance $I \in \text{Inst}(\mathbf{R})$ into a k-relation $Q(I) \subseteq \text{dom}(I)^k$. Notice that if $k = 0$ (Q is a Boolean query), then the answer to Q is either the set with one 0-ary tuple (denoted by *true*), or the empty set (denoted by *false*). Thus, if Q is a Boolean query, then $Q(I)$ is either *true* or *false*. As is customary, we assume that queries are closed under isomorphisms, and we use CQ to denote the class of conjunctive queries (that is, queries formed using conjunctions of relational atoms) and UCQ to denote the class of unions of conjunctive queries. If we extend these classes by allowing equalities, then we use superscript $=$.

As usual, the semantics of queries in the presence of schema mappings is defined in terms of the notion of *certain answer*. Assume that $\mathcal{M}$ is a mapping from a schema $\mathbf{R}_1$ to a schema $\mathbf{R}_2$. Then given an instance I of $\mathbf{R}_1$ and a query Q over $\mathbf{R}_2$, the *certain answer of Q for I under $\mathcal{M}$*, denoted by $\text{certain}_{\mathcal{M}}(Q, I)$, is the set of tuples that belong to the evaluation of Q over every possible solution for I under $\mathcal{M}$, that is,

$$\text{certain}_{\mathcal{M}}(Q, I) = \bigcap_{J \in \text{Sol}_{\mathcal{M}}(I)} Q(J).$$

2.3. Dependencies and definability of mappings

A relational atom over $\mathbf{R}$ is a formula of the form $R(\bar{x})$ with $R \in \mathbf{R}$ and $\bar{x}$ a tuple of (not necessarily distinct) variables. Given disjoint schemas $\mathbf{R}_1$ and $\mathbf{R}_2$, a *source-to-target tgd* (st-tgd) from $\mathbf{R}_1$ to $\mathbf{R}_2$ is a sentence of the form:

$$\forall \bar{x} \forall \bar{y} \, \big(\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \, \psi(\bar{x}, \bar{z})\big),$$

where (a) $\varphi(\bar{x}, \bar{y})$ is a conjunction of relational atoms over $\mathbf{R}_1$, and (b) $\psi(\bar{x}, \bar{z})$ is a conjunction of relational atoms over $\mathbf{R}_2$. The left-hand side of the implication in a st-tgd is called the premise, and the right-hand side the conclusion. We usually omit the outermost universal quantifier in st-tgds and thus, for a dependency as the above, we just write $\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \, \psi(\bar{x}, \bar{z})$.

A *second-order tuple-generating dependency* (*SO-tgd*) *from* $\mathbf{R}_1$ *to* $\mathbf{R}_2$ is a formula of the form:

$$\exists \bar{f} \, \big(\forall \bar{x}_1 \, (\varphi_1 \to \psi_1) \wedge \cdots \wedge \forall \bar{x}_n \, (\varphi_n \to \psi_n)\big), \tag{1}$$

where (a) each member of $\bar{f}$ is a function symbol; (b) each formula φ_i ($1 \leqslant i \leqslant n$) is a conjunction of relational atoms of the form $S(y_1, \ldots, y_k)$ and equality atoms of the form $t = t'$, where S is a k-ary relation symbol of $\mathbf{R}_1$ and $y_1, \ldots, y_k$ are (not necessarily distinct) variables in $\bar{x}_i$, and t, t' are terms built from $\bar{x}_i$ and $\bar{f}$; (c) each formula ψ_i ($1 \leqslant i \leqslant n$) is a conjunction of relational atomic formulas over $\mathbf{R}_2$ mentioning terms built from $\bar{x}_i$ and $\bar{f}$; and (d) each variable in $\bar{x}_i$ ($1 \leqslant i \leqslant n$) appears in some relational atom of φ_i. As was noted in [14,25], there is a subtlety in the semantics of SO-tgds, namely, the semantics of existentially quantified function symbols. In particular, in deciding whether (I, J) satisfies an SO-tgd λ, what should the domain and range of the functions instantiating the existentially quantified function symbols be? The obvious choice is to let the domain and range be $\mathrm{dom}(I) \cup \mathrm{dom}(J)$, but it is shown in [14,25] that this does not work properly. Instead, the solution in [14,25] is as follows. Let λ be an SO-tgd from a source schema $\mathbf{R}_1$ to a target schema $\mathbf{R}_2$. Then given an instance I of $\mathbf{R}_1$ and an instance J of $\mathbf{R}_2$, instance (I, J) is converted into a structure $(\mathbf{D}; I, J)$, which is just like (I, J) except that it has universe $\mathbf{D}$. The domain and range of the functions in λ is then taken to be $\mathbf{D}$. The intuition is that the universe contains the domain of (I, J) along with an infinite set of extra values. Then (I, J) is said to satisfy λ, denoted by $(I, J) \models \lambda$, if $(\mathbf{D}; I, J) \models \lambda$ under the standard notion of satisfaction in second-order logic (see, for example, [9]). It should be noticed that it is shown in [14] that in the case of SO-tgds, instead of taking the universe $\mathbf{D}$, one can take a "sufficiently large" finite universe.

Let $\mathbf{R}_1$ and $\mathbf{R}_2$ be schemas with no relation symbols in common and Σ a set of sentences over $\mathbf{R}_1$ and $\mathbf{R}_2$. Then we say that a mapping $\mathcal{M}$ from $\mathbf{R}_1$ to $\mathbf{R}_2$ is *specified* by Σ, denoted by $\mathcal{M} = (\mathbf{R}_1, \mathbf{R}_2, \Sigma)$, if for every instance I of $\mathbf{R}_1$ and instance J of $\mathbf{R}_2$, it holds that $(I, J) \in \mathcal{M}$ if and only if (I, J) satisfies the dependencies in Σ. Notice that SO-tgds are closed under conjunction (that is, if σ_1 and σ_2 are SO-tgds, then $\sigma_1 \wedge \sigma_2$ is logically equivalent to an SO-tgd). Thus, when specifying mappings, we talk about a mapping specified by an SO-tgd (instead of a set of SO-tgds).

We conclude this section by pointing out that every finite set Σ of st-tgds can be transformed into an equivalent SO-tgd by *Skolemizing* the existentially quantified variables in the conclusions of the dependencies in Σ. For example, a set Σ consisting of the following st-tgds:

$$A(x, y) \rightarrow \exists z \, \big(R(x, z) \wedge T(z, y)\big),$$
$$B(x, y) \rightarrow R(x, y)$$

is logically equivalent to the following SO-tgd:

$$\exists f \, \Big(\forall x \forall y \, \big(A(x, y) \rightarrow R\big(x, f(x, y)\big)\big) \wedge \forall x \forall y \, \big(A(x, y) \rightarrow T\big(f(x, y), y\big)\big) \wedge \forall x \forall y \, \big(B(x, y) \rightarrow R(x, y)\big)\Big). \tag{2}$$

3. Composition, inversion, and how the composition of st-tgds is not always invertible

Two of the most important schema mapping operators that have been considered in the literature are the composition and inversion. In this section, we first recall the definitions of these operators, and show a negative result regarding the combination of both operators for mappings given by st-tgds.

3.1. Composition

The composition of mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ is a mapping that intuitively has the same effect as the application of $\mathcal{M}_1$ and $\mathcal{M}_2$ one after the other. The formalization of mappings as just pairs of database instances allows for a clean definition of the semantics of the composition of mappings, based on the composition of binary relations [23,14]. Formally, given schemas $\mathbf{R}_1$, $\mathbf{R}_2$, and $\mathbf{R}_3$, and mappings $\mathcal{M}_1$ from $\mathbf{R}_1$ to $\mathbf{R}_2$ and $\mathcal{M}_2$ from $\mathbf{R}_2$ to $\mathbf{R}_3$, the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted by $\mathcal{M}_1 \circ \mathcal{M}_2$, is defined as:

$$\mathcal{M}_1 \circ \mathcal{M}_2 = \big\{(I, K) \in \mathrm{Inst}(\mathbf{R}_1) \times \mathrm{Inst}(\mathbf{R}_3) \;\big|\; \text{there exists } J \in \mathrm{Inst}(\mathbf{R}_2) \text{ such that } (I, J) \in \mathcal{M}_1 \text{ and } (J, K) \in \mathcal{M}_2\big\}.$$

Notice that this definition of composition ensures that for mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ is unique.

Once the notion of composition is clearly defined, the natural problem of how to specify the composition arises. This problem was throughly studied by Fagin et al. in [14]. In that paper, the authors first show a negative result regarding the composition of mappings specified by st-tgds, namely that there exist mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ both specified by st-tgds such that the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ cannot be specified in first-order logic [14]. On the positive side, Fagin et al. show in [14] that the language of SO-tgds is a well-behaved language regarding composition. In particular, they show that SO-tgds are *closed under composition*:

Theorem 1. *(See [14].) Let $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ be mappings specified by SO-tgds. Then the composition $\mathcal{M}_{12} \circ \mathcal{M}_{23}$ can also be specified by an SO-tgd.*

It is important to notice that Theorem 1 implies that the composition of a finite number of mappings specified by st-tgds can be defined by an SO-tgd, as every finite set of st-tgds can be expressed as an SO-tgd.

Theorem 2. *(See [14].) The composition of a finite number of mappings, each specified by a finite set of st-tgds, can be specified by an SO-tgd.*

In [14], Fagin et al. show several other desirable properties of SO-tgds, being one of the most important that SO-tgds are the minimal language capable of expressing the composition of st-tgds. In particular, they show that every mapping specified by an SO-tgd is equivalent to the composition of a finite number of mappings, each specified by a finite set of st-tgds. This result was later improved by Arenas et al. in [1], where the authors prove that every mapping specified by an SO-tgd is equivalent to the composition of two mappings specified by st-tgds.

3.2. Inversion

In contrast with the case of the composition of schema mappings, whose semantics can be defined in a clean way in terms of the composition of binary relations, the inverse operator for schema mappings has turned out to be far more difficult to define. In fact, there is not yet consensus on a semantics for this operator, so we review in this section some of the main alternatives that have been studied in the literature: Fagin-inverse [10], quasi-inverse [15], maximum recovery [5], and $\mathcal{C}$-maximum recovery [4]. Recall that we are mainly interested in computing inverses of st-mappings, that is, of mappings from a schema $\mathbf{S}$ to a schema $\mathbf{T}$ in which instances of $\mathbf{S}$ are assumed to contain only constant values while instances of $\mathbf{T}$ contain constant and nulls. Notice that the inverse of an st-mapping is a ts-mapping. (To see some recent notions of inverse that relax this assumption, we refer the reader to [16] and [2].)

Fagin [10] gave a first formal semantics for the inversion of schema mappings. Intuitively, Fagin's definition was based on the idea that any mapping composed with its inverse should be equal to the *identity*. Since we already have a clear notion of what composition is, it is only needed to define a meaningful notion of identity in the context of schema mappings. Fagin does so by introducing the following notion. Let $\mathbf{S}$ be a schema and consider the mapping $\overline{\mathrm{Id}}_{\mathbf{S}} = \{(I, J) \mid I, J \in \mathrm{Inst}(\mathbf{S})$ and $I \subseteq J\}$. In [10], Fagin made the case that $\overline{\mathrm{Id}}_{\mathbf{S}}$ is a natural identity mapping in the context of st-tgds (which was the main language considered in [10]). Thus, given a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$, a mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ is a *Fagin-inverse* of $\mathcal{M}$ if $\mathcal{M} \circ \mathcal{M}' = \overline{\mathrm{Id}}_{\mathbf{S}}$ [10].

It is observed in [15] that the notion of Fagin-inverse is very restricted as it is rare that a mapping possesses a Fagin-inverse. For this reason, Fagin et al. [15] introduced the notion of a quasi-inverse of a schema mapping. We do not introduce the formalization of this notion here and refer the reader to [15] for details. We only mention that the notion of quasi-inverse is a strict generalization of notion of Fagin-inverse, as if $\mathcal{M}'$ is a Fagin-inverse of a mapping $\mathcal{M}$, then $\mathcal{M}'$ is also a quasi-inverse of $\mathcal{M}$ [15]; and, on the other side, there are mappings specified by st-tgds that have quasi-inverses but do not have Fagin-inverses [15]. It was also shown in [15] that there exist simple mappings specified by st-tgds for which quasi-inverses do not exist.

In view of the aforementioned results, Arenas et al. introduced the notion of maximum recovery in [5]. Consider the mapping $\mathrm{Id}_{\mathbf{S}}$ given by $\mathrm{Id}_{\mathbf{S}} = \{(I, I) \mid I \in \mathrm{Inst}(\mathbf{S})\}$. Notice the difference between $\overline{\mathrm{Id}}_{\mathbf{S}}$ and $\mathrm{Id}_{\mathbf{S}}$; mapping $\mathrm{Id}_{\mathbf{S}}$ is the classical identity of binary relations. When trying to invert a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$, the ideal would be to find a mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ such that $\mathcal{M} \circ \mathcal{M}' = \mathrm{Id}_{\mathbf{S}}$. If such a mapping exists, we know that if we use $\mathcal{M}$ to exchange data, the application of $\mathcal{M}'$ gives as result exactly the initial source instance. Unfortunately, in most cases this ideal is impossible to reach. The intuition behind the notion of maximum recovery is that we want to find a mapping $\mathcal{M}'$ such that $\mathcal{M} \circ \mathcal{M}'$ is *as close as possible* to $\mathrm{Id}_{\mathbf{S}}$. The following definition formalizes this idea.

Definition 3. (See [5].) Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$. A mapping $\mathcal{M}'$ is a *recovery* of $\mathcal{M}$ if $\mathrm{Id}_{\mathbf{S}} \subseteq \mathcal{M} \circ \mathcal{M}'$. Moreover, mapping $\mathcal{M}'$ is a *maximum recovery* of $\mathcal{M}$, if $\mathcal{M}'$ is a recovery of $\mathcal{M}$ and for every other recovery $\mathcal{M}''$ of $\mathcal{M}$, it holds that $\mathcal{M} \circ \mathcal{M}' \subseteq \mathcal{M} \circ \mathcal{M}''$.

One of the main results in [5] is that every st-mapping specified by a finite set of st-tgds has a maximum recovery. The authors also prove the following result that states the relationship between Fagin-inverses, quasi-inverses and maximum recoveries. We first need to introduce some terminology. We say that a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ is *total* if for every $I \in \mathrm{Inst}(\mathbf{S})$, it holds that $\mathrm{Sol}_{\mathcal{M}}(I) \neq \emptyset$. Moreover, $\mathcal{M}$ is said to be *closed-down on the left* [10,27] if for every $(I, J) \in \mathcal{M}$ and $K \subseteq I$, we have that $(K, J) \in \mathcal{M}$. It should be noticed that every mapping specified by an SO-tgd is total and closed-down on the left (and, thus, every mapping specified by a finite set of st-tgds is also total and closed-down on the left).

Proposition 4. *(See [5].) Let $\mathcal{M}$ be a total mapping that is closed-down on the left.*

1. *If $\mathcal{M}$ has a Fagin-inverse, then $\mathcal{M}$ has a maximum recovery and every maximum recovery $\mathcal{M}'$ of $\mathcal{M}$ is also a Fagin-inverse of $\mathcal{M}$.*
2. *If $\mathcal{M}$ has a quasi-inverse, then $\mathcal{M}$ has a maximum recovery and every maximum recovery $\mathcal{M}'$ of $\mathcal{M}$ is also a quasi-inverse of $\mathcal{M}$.*

Finally, we also consider in this section the notion of $\mathcal{C}$-maximum recovery, which is a relaxation of the notion of maximum recovery w.r.t. a class of queries $\mathcal{C}$.

Definition 5. (See [4].) Let $\mathcal{M}$ be a mapping from a schema $\mathbf{S}$ to a schema $\mathbf{T}$, and $\mathcal{C}$ a class of queries. A mapping $\mathcal{M}'$ is a *$\mathcal{C}$-recovery* of $\mathcal{M}$ if for every query $Q \in \mathcal{C}$ over $\mathbf{S}$ and every instance $I \in \mathrm{Inst}(\mathbf{S})$:

$$\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) \subseteq Q(I).$$

Moreover, mapping $\mathcal{M}'$ is a *$\mathcal{C}$-maximum recovery* of $\mathcal{M}$, if $\mathcal{M}'$ is a $\mathcal{C}$-recovery of $\mathcal{M}$ and for every other $\mathcal{C}$-recovery $\mathcal{M}''$ of $\mathcal{M}$, it holds that:

$$\text{certain}_{\mathcal{M}\circ\mathcal{M}''}(Q, I) \subseteq \text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I),$$

for every query $Q \in \mathcal{C}$ over $\mathbf{S}$ and every instance $I \in \text{Inst}(\mathbf{S})$.

It is not difficult to show that if $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$, then $\mathcal{M}'$ is a $\mathcal{C}$-maximum recovery of $\mathcal{M}$ for every possible class of queries $\mathcal{C}$ [4]. In [4], the authors show several properties of $\mathcal{C}$-maximum recoveries when CQ is considered as the class of queries $\mathcal{C}$. In particular, it is proved in [4] that there exist mappings $\mathcal{M}$ and $\mathcal{M}'$ specified by st-tgds such that $\mathcal{M}'$ is a CQ-maximum recovery of $\mathcal{M}$ but $\mathcal{M}'$ is not a maximum recovery of $\mathcal{M}$, thus, showing that the notion of CQ-maximum recovery strictly generalizes the notion of maximum recovery.

3.3. The composition of st-tgds is not always invertible

Recall that every st-mapping specified by a finite set of st-tgds admits a maximum recovery [5]. Our next result shows that, unfortunately, SO-tgds are not appropriate for inversion, as there exist mappings specified by SO-tgds that do not admit maximum recoveries. More precisely, we show something stronger, namely that there exists a mapping specified by an SO-tgd that does not even admit a CQ-maximum recovery.

Theorem 6. *There exists a mapping $\mathcal{M}$ specified by an SO-tgd that has no* CQ*-maximum recovery.*

Proof. Let $\mathbf{S}$ be a source schema $\{A(\cdot), B(\cdot), C_1(\cdot), C_2(\cdot)\}$, $\mathbf{T}$ be target schema $\{R(\cdot,\cdot), S(\cdot)\}$, and $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ a mapping specified by the following SO-tgd λ:

$$\begin{aligned}\exists f \exists g\ \big[&\forall x\,\big(B(x) \wedge C_1(x) \wedge x = f(x) \to S(x)\big)\\ &\wedge \forall x\,\big(B(x) \wedge C_1(x) \to R\big(x, f(x)\big)\big)\\ &\wedge \forall x\,\big(B(x) \wedge C_2(x) \to R(x, x)\big)\\ &\wedge \forall x\,\big(A(x) \to R\big(x, g(x)\big)\big)\big].\end{aligned}$$

We show next that $\mathcal{M}$ does not have a CQ-maximum recovery. In order to obtain a contradiction, assume that $\mathcal{M}'$ is a CQ-maximum recovery of $\mathcal{M}$.

We construct two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ from $\mathbf{T}$ to $\mathbf{S}$ to derive a contradiction. Let $\mathcal{M}_1$ be the mapping from $\mathbf{T}$ to $\mathbf{S}$ given by the dependencies:

$$\exists y\,\big(R(x, y) \wedge x \neq y\big) \to B(x) \wedge C_1(x),$$
$$R(x, x) \wedge S(x) \to B(x) \wedge C_1(x),$$

and $\mathcal{M}_2$ the mapping from $\mathbf{T}$ to $\mathbf{S}$ given by the dependency:

$$R(x, x) \to B(x) \wedge C_2(x).$$

We next show that $\mathcal{M}_1$ and $\mathcal{M}_2$ are CQ-recoveries of $\mathcal{M}$. We begin by showing this property for $\mathcal{M}_1$. Let I be an arbitrary instance of $\mathbf{S}$, and $f^\star : \mathbf{D} \to \mathbf{D}$ a function such that for every element $a \in \text{dom}(I)$, it holds that $f^\star(a) \neq a$. Consider now the instance J of $\mathbf{T}$ constructed as follows:

$$\begin{aligned}J = &\big\{R\big(a, f^\star(a)\big) \bigm| B(a) \in I \text{ and } C_1(a) \in I\big\}\\ &\cup \big\{R(a, a) \bigm| B(a) \in I \text{ and } C_2(a) \in I\big\}\\ &\cup \big\{R(a, a) \bigm| A(a) \in I\big\}.\end{aligned}$$

It is not difficult to see that $(I, J) \models \lambda$ by using $f^\star$ as the interpretation for f and the identity function to interpret g. Thus we have that $J \in \text{Sol}_{\mathcal{M}}(I)$. Moreover, by the definition of J and $\mathcal{M}_1$ it holds that $I \in \text{Sol}_{\mathcal{M}_1}(J)$, thus we have that $I \in \text{Sol}_{\mathcal{M}\circ\mathcal{M}_1}(I)$. This last fact implies that $\text{certain}_{\mathcal{M}\circ\mathcal{M}_1}(Q, I) \subseteq Q(I)$ for every query Q, and, in particular, for every query Q in CQ. Thus we obtain that $\mathcal{M}_1$ is a CQ-recovery of $\mathcal{M}$.

For the case of $\mathcal{M}_2$, let I be an arbitrary instance of $\mathbf{S}$ and $f^\star$ a function as defined above. Consider now the instance J of $\mathbf{T}$ such that:

$$\begin{aligned}J = &\big\{R(a, a) \bigm| B(a) \in I \text{ and } C_2(a) \in I\big\}\\ &\cup \big\{R\big(a, f^\star(a)\big) \bigm| B(a) \in I \text{ and } C_1(a) \in I\big\}\\ &\cup \big\{R\big(a, f^\star(a)\big) \bigm| A(a) \in I\big\}.\end{aligned}$$

We have that $(I, J) \models \lambda$ (this time interpreting both f and g as $f^\star$), and following a similar argument as in the previous paragraph we can use J to show that $I \in \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}_2}(I)$. Thus we obtain that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}_2}(Q, I) \subseteq Q(I)$ for every query Q, and, in particular, for every query Q in CQ, implying that $\mathcal{M}_2$ is a CQ-recovery of $\mathcal{M}$.

Now given that $\mathcal{M}'$ is a CQ-maximum recovery of $\mathcal{M}$, and since $\mathcal{M}_1$ and $\mathcal{M}_2$ are CQ-recoveries of $\mathcal{M}$, we have that for every instance I of **S** and conjunctive query Q, it holds that:

$$\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}_1}(Q, I) \subseteq \mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) \subseteq Q(I), \tag{3}$$

$$\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}_2}(Q, I) \subseteq \mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) \subseteq Q(I). \tag{4}$$

We show next that (3) and (4) lead to a contradiction.

Consider the instance $I_1 = \{B(a), C_1(a)\}$. We next argue that by the definition of $\mathcal{M}$ and $\mathcal{M}_1$, we have that for every $K \in \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}_1}(I_1)$ it holds that $I_1 \subseteq K$. Assume that $(I_1, J) \models \lambda$. Thus, since $\mathrm{dom}(I_1) = \{a\}$ we have only two possibilities: either f is interpreted as a function $f^\star$ such that $f^\star(a) \neq a$, or f is interpreted as a function $f^\star$ such that $f^\star(a) = a$. In the first case we have that $R(a, f^\star(a)) \in J$, and in the second case we have that $R(a, a), S(a) \in J$ and thus, in both cases we have that every solution for J under $\mathcal{M}_1$ contains the facts $B(a)$ and $C_1(a)$. Moreover, it is not difficult to see that $I_1 \in \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}_1}(I_1)$. All the previous discussion implies that for the query Q_1 given by formula $B(x) \wedge C_1(x)$, it holds that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}_1}(Q_1, I_1) = Q_1(I_1) = \{a\}$. Thus, from (3) we obtain that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q_1, I_1) = \{a\}$. Therefore, for every instance $K \in \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_1)$ it holds that $B(a)$ and $C_1(a)$ are facts in K.

Consider now the instance $I_2 = \{B(a), C_2(a)\}$. Similarly as in the previous paragraph, we can show that for the query Q_2 given by $B(x) \wedge C_2(x)$, it holds that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}_2}(Q_2, I_2) = Q_2(I_2) = \{a\}$. Thus, from (4) we obtain that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q_2, I_2) = \{a\}$. Therefore, for every instance $K \in \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_2)$ it holds that $B(a)$ and $C_2(a)$ are facts in K.

Finally, consider instance $I_3 = \{A(a)\}$. Notice that an instance J is a solution of I_3 under $\mathcal{M}$ if and only if J contains a fact of the form $R(a, b)$ with b an arbitrary element. From this it is easy to conclude that $J \in \mathrm{Sol}_{\mathcal{M}}(I_1) \cup \mathrm{Sol}_{\mathcal{M}}(I_2)$, and thus $\mathrm{Sol}_{\mathcal{M}}(I_3) \subseteq \mathrm{Sol}_{\mathcal{M}}(I_1) \cup \mathrm{Sol}_{\mathcal{M}}(I_2)$. Moreover, $\mathrm{Sol}_{\mathcal{M}}(I_3) \subseteq \mathrm{Sol}_{\mathcal{M}}(I_1) \cup \mathrm{Sol}_{\mathcal{M}}(I_2)$ implies that $\mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_3) \subseteq \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_1) \cup \mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_2)$. We have shown that every instance in $\mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_1)$ contains the facts $B(a), C_1(a)$ and that every instance in $\mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_2)$ contains the facts $B(a), C_2(a)$, thus implying that every instance in $\mathrm{Sol}_{\mathcal{M} \circ \mathcal{M}'}(I_3)$ contains the fact $B(a)$. Consider the query Q given by $B(x)$. By the previous discussion, we have that $a \in \mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q, I_3)$ but $Q(I_3) = \emptyset$, and thus we have that $\mathrm{certain}_{\mathcal{M} \circ \mathcal{M}'}(Q, I_3) \not\subseteq Q(I_3)$. This contradicts the fact that $\mathcal{M}'$ is a CQ-maximum recovery of $\mathcal{M}$, completing the proof. □

Recall that if a mapping $\mathcal{M}$ has a maximum recovery, then $\mathcal{M}$ also has a CQ-maximum recovery. Thus, by the previous result, we obtain that, as opposed to the case of st-tgds, a mapping specified by an SO-tgd is not guaranteed to have a maximum recovery. Also notice that a mapping specified by an SO-tgd is total and closed-down on the left, thus, we have by Proposition 4 that a mapping specified by an SO-tgd is guaranteed to have neither a Fagin-inverse nor a quasi-inverse. Therefore, by considering the fact that every SO-tgd defines the composition of two mappings specified by st-tgds [14,1], we obtain a fundamental negative result for the composition operator.

Corollary 7. *There exist mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, each specified by a finite set of st-tgds, such that the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ has neither* CQ*-maximum recovery, nor maximum recovery, nor quasi-inverse, nor Fagin-inverse.*

4. Plain SO-tgds to the rescue: CQ-composition, inversion and good structural properties

The negative result in Section 3.3 suggests that to obtain a language that is closed under composition and has good properties regarding inversion, we have no choice but to relax the semantics of composition. In this section, we take this path by considering the notion of composition w.r.t. conjunctive queries, or just CQ-composition, introduced by Madhavan and Halevy in [22], and which predates the set-theoretic notion of composition presented in Section 3.1.

The notion of CQ-composition can be defined in terms of the concept of *conjunctive-query equivalence* of mappings that was implicitly introduced in [22], and generalized in [13] when studying optimization of schema mappings. Two mappings $\mathcal{M}$ and $\mathcal{M}'$ from a schema **S** to a schema **T** are said to be *equivalent w.r.t. conjunctive queries*, denoted by $\mathcal{M} \equiv_{\mathrm{CQ}} \mathcal{M}'$, if for every conjunctive query Q over **T** and every instance I of **S**, it holds that $\mathrm{certain}_{\mathcal{M}}(Q, I) = \mathrm{certain}_{\mathcal{M}'}(Q, I)$. Then a mapping $\mathcal{M}_3$ is said to be a CQ-composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ if $\mathcal{M}_3 \equiv_{\mathrm{CQ}} \mathcal{M}_1 \circ \mathcal{M}_2$.

Interestingly, in this section we show that the notion of CQ-composition is captured by a natural fragment of the language of SO-tgds, which also shares some fundamental structural properties with the language of st-tgds and has good properties regarding inversion. More specifically, we define in Section 4.1 the language of *plain* SO-tgds, and then we show in Section 4.2 that they are the right language for representing the CQ-composition of mappings given by st-tgds. Moreover, we prove in Section 4.2 that every mapping given by plain SO-tgds admits a maximum recovery, thus showing that this language is also appropriate for inversion. Finally, we present in Section 4.3 some of the structural properties of st-tgds that have been identified as of fundamental importance for data exchange [27], and we prove that these properties are also satisfied by the class of mappings defined by plain SO-tgds.

4.1. Plain SO-tgds

In order to define plain SO-tgds, we use the notion of plain term. Given a tuple $\bar{f}$ of function symbols and a tuple $\bar{x}$ of variables, a *plain term* built from $\bar{f}$ and $\bar{x}$, is either a variable x in $\bar{x}$, or a term of the form $f(u_1, \dots, u_k)$ where f is a function symbol in $\bar{f}$ and each u_i is a variable in $\bar{x}$ $(1 \leqslant i \leqslant k)$.

Definition 8. Given schemas $\mathbf{R}_1$ and $\mathbf{R}_2$ with no relation symbols in common, a *plain second-order tuple-generating dependency* (*plain SO-tgd*) *from* $\mathbf{R}_1$ *to* $\mathbf{R}_2$ is a formula of the form:

$$\exists \bar{f} \, \big(\forall \bar{x}_1 \, (\varphi_1 \to \psi_1) \wedge \cdots \wedge \forall \bar{x}_n \, (\varphi_n \to \psi_n)\big),$$

where:

1. each member of $\bar{f}$ is a function symbol,
2. each formula φ_i $(1 \leqslant i \leqslant n)$ is a conjunction of relational atoms of the form $S(y_1, \dots, y_k)$, where S is a k-ary relation symbol of $\mathbf{R}_1$ and $y_1, \dots, y_k$ are (not necessarily distinct) variables in $\bar{x}_i$,
3. each formula ψ_i $(1 \leqslant i \leqslant n)$ is a conjunction of relational atomic formulas over $\mathbf{R}_2$ mentioning plain terms built from $\bar{x}_i$ and $\bar{f}$, and
4. each variable in $\bar{x}_i$ $(1 \leqslant i \leqslant n)$ appears in some relational atom of φ_i.

That is, the language of plain SO-tgds is obtained from the language of SO-tgds by forbidding equality of terms and nesting of functions. For example, formula (2) is a plain SO-tgd, while the formula used in the proof of Theorem 6 is not a plain SO-tgd as it includes an equality of terms.

As was noted in Section 2.3, given an SO-tgd σ from a schema $\mathbf{R}_1$ to a schema $\mathbf{R}_2$ and a pair (I, J) of instances of $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively, the satisfaction of σ by (I, J) is defined by considering the standard notion of satisfaction in second-order logic and a structure $(\mathbf{D}; I, J)$, which is just like (I, J) except that it has universe $\mathbf{D}$. That is, (I, J) is said to satisfy σ if $(\mathbf{D}; I, J) \models \sigma$ under the standard notion of satisfaction in second-order logic. The semantics of plain SO-tgds is inherited from the semantics of SO-tgds. But as opposed to the case of SO-tgds, it can be shown that for every plain SO-tgd λ from a schema $\mathbf{R}_1$ to a schema $\mathbf{R}_2$ and a pair (I, J) of instances of $\mathbf{R}_1$ and $\mathbf{R}_2$, respectively, $(\mathbf{D}; I, J) \models \lambda$ if and only if $(\text{dom}(I) \cup \text{dom}(J); I, J) \models \lambda$. Thus, in order to check whether a plain SO tgd λ is satisfied by a pair (I, J) of instances, one can assume that $\text{dom}(I) \cup \text{dom}(J)$ is the domain and range of the functions instantiating the existentially quantified function symbols in λ.

Finally, it should be noticed that, as for the case of SO-tgds, plain SO-tgds are closed under conjunction and, thus, we talk about a mapping specified by a plain SO-tgd (instead of a set of plain SO-tgds). Moreover, it is easy to see that every set of st-tgds is equivalent to a plain SO-tgd.

4.2. Plain SO-tgds: a language for CQ*-composition which is also invertible*

In this section, we show that the language of plain SO-tgds is the right language for representing the CQ-composition of mappings given by st-tgds, in the same sense that SO-tgds are the right language for representing the composition of mappings given by st-tgds, as shown in Section 3.1. Besides, we also show that in sharp contrast with the case of SO-tgds, every mapping specified by a plain SO-tgd admits a maximum recovery. In Section 5, we provide a polynomial-time algorithm for computing these maximum recoveries, thus showing that plain SO-tgds also have good properties regarding inversion.

We start by proving a useful lemma showing that although the language of plain SO-tgds is less expressive than the language of SO-tgds, in terms of CQ-equivalence they have the same expressive power.

Lemma 9. *For every SO-tgd λ, there exists a plain SO-tgd λ' such that $\lambda \equiv_{\text{CQ}} \lambda'$.*

Proof. We first recall a result that will considerably simplify the proof. Arenas et al. [1] proved that every SO-tgd λ is logically equivalent to an SO-tgd $\lambda^\star$ that does not have nesting of functions. That is, the language of SO-tgds is equivalent to the language obtained from plain SO-tgds by adding equality of plain terms [1]. Thus, we can assume that SO-tgds contain only plain terms.

We also make use of the chase procedure for SO-tgds that we introduce next. Given an SO-tgd λ from a schema $\mathbf{S}$ to a schema $\mathbf{T}$ of the form:

$$\exists \bar{f} \, \big(\forall \bar{x}_1 \, (\varphi_1 \to \psi_1) \wedge \cdots \wedge \forall \bar{x}_k \, (\varphi_n \to \psi_n)\big), \tag{5}$$

where λ only contains plain terms, the chase of the source instance I with λ, denoted by $\text{chase}_\lambda(I)$, is the instance J of $\mathbf{T}$ constructed as follows [14]. Fix an interpretation for the function symbols of $\bar{f}$ such that for every n-ary function symbol f in $\bar{f}$ and n-ary tuple $\bar{a}$ of elements in I, $f(\bar{a})$ is interpreted as a fresh null value. Notice that with this interpretation, the equality $f(\bar{a}) = g(\bar{b})$ holds if and only if f and g are the same function symbol and $\bar{a} = \bar{b}$. Thus we can denote a null of the form $f(\bar{a})$ with its own syntactic form (that is, simply as $f(\bar{a})$). We need an additional notation. Given a formula α that considers variables in a tuple $\bar{x}$ and plain terms built from $\bar{f}$ and $\bar{x}$, we denote by $\alpha[\bar{x} \to \bar{a}]$ the formula obtained by replacing every occurrence of a variable in $\bar{x}$ by the corresponding value in $\bar{a}$. We are now ready to introduce the process to construct instance J. For every $i \in \{1, \dots, n\}$ and every tuple $\bar{a}$ of elements in I such that the arity of $\bar{a}$ is the same as the arity of $\bar{x}_i$, if I satisfies the formula $\varphi_i[\bar{x}_i \to \bar{a}]$, then add all the atoms in $\psi_i[\bar{x}_i \to \bar{a}]$ to instance J. Fagin et al. [14] show that chase$_\lambda(I)$ is a universal solution for I under the mapping specified by λ.

We next show how to transform an SO-tgd λ into a plain SO-tgd λ' such that chase$_\lambda(I) =$ chase$_{\lambda'}(I)$. Let λ be an SO-tgd of the form (5). We first construct a set of formulas Σ' by repeating the following for every i such that $1 \leqslant i \leqslant n$. Start with φ'_i as φ_i and ψ'_i as ψ_i.

1. Replace every equality of the form $f(x_1, \dots, x_k) = f(y_1, \dots, y_k)$ in φ'_i by the conjunction of equalities $x_1 = y_1 \wedge \dots \wedge x_k = y_k$.
2. If there is an equality of the form $x = x'$ in φ'_i with x and x' variables, then eliminate the equality and replace in φ'_i and in ψ'_i every occurrence of x' by x.
3. Repeat the previous step until φ'_i has no equality of the form $x = x'$ with x and x' variables.

If after the process, φ'_i does not contain any equality between plain terms (the only equalities that may remain must mention different function symbols), then add formula $\forall \bar{x}'_i\ (\varphi'_i \to \psi'_i)$ to Σ', where $\bar{x}'_i$ is the tuple of remaining variables in φ'_i. Finally, we define λ' as the formula $\exists \bar{f}\ (\bigwedge \Sigma')$, where $(\bigwedge \Sigma')$ denotes the conjunction of all the formulas in Σ'. Notice that the formula λ' is a plain SO-tgd.

It is not difficult to see that for every instance I, it holds that chase$_\lambda(I) =$ chase$_{\lambda'}(I)$. Just notice that if I satisfies the formula $\varphi_i[\bar{x}_i \to \bar{a}]$, then (i) φ_i does not contain equalities of the form $f(\bar{u}) = g(\bar{v})$ with f and g distinct function symbols, (ii) if $f(x_1, \dots, x_k) = f(y_1, \dots, y_k)$ is an equality in φ_i, then the equalities $x_1 = y_1, \dots, x_k = y_k$ should hold for the assignment $\bar{x}_i \to \bar{a}$ and (iii) if $x = x'$ is an equality in φ_i, then the values a and a' that correspond to x and x' in the assignment $\bar{x}_i \to \bar{a}$ should be equal. This implies that if $\forall \bar{x}'_i\ (\varphi'_i \to \psi'_i)$ is in Σ', and $\bar{x}'_i \to \bar{a}'$ is the assignment obtained from the assignment $\bar{x}_i \to \bar{a}$ by considering only the variables in $\bar{x}'_i$, then I satisfies $\varphi'_i[\bar{x}'_i \to \bar{a}']$. Similarly, it can be shown that if $\forall \bar{x}'_i\ (\varphi'_i \to \psi'_i)$ is in Σ' and I satisfies $\varphi'_i[\bar{x}'_i \to \bar{a}']$, then I satisfies $\varphi_i[\bar{x}_i \to \bar{a}]$. The property follows from the observation that if I satisfies $\varphi_i[\bar{x}_i \to \bar{a}]$, then $\psi_i[\bar{x}_i \to \bar{a}]$ is equal to $\psi'_i[\bar{x}'_i \to \bar{a}']$.

To conclude the proof we use a result proved by Fagin et al. [13]. In [13], it is proved that if two mappings have the same universal solutions (up to homomorphisms) for every source instance, then the mappings are CQ equivalent. Thus, given that for every source instance I, it holds that chase$_\lambda(I)$ is a universal solution for I under the mapping specified by λ, and chase$_{\lambda'}(I)$ is a universal solution for I under the mapping specified by λ', we obtain that λ and λ' are CQ-equivalent since chase$_\lambda(I) =$ chase$_{\lambda'}(I)$. This concludes the proof of the lemma. □

As a corollary of Lemma 9 and the fact that SO-tgds are closed under composition (see Theorem 1), we obtain that the language of plain SO-tgds is closed under CQ-composition.

Theorem 10. *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be mappings specified by plain SO-tgds. Then a* CQ*-composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ can also be specified by a plain SO-tgd.*

In Theorem 5.4 in [14], it is shown that there exist schema mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, each specified by a finite set of st-tgds, such that the composition of $\mathcal{M}_1$ and $\mathcal{M}_2$ cannot be specified by any finite or infinite set of SO-tgds without equalities. Hence, the language of plain SO-tgds is not closed under the usual notion of composition, and we have to consider the notion of CQ-composition instead. In particular, as a corollary of Theorem 10, we obtain that the language of plain SO-tgds is appropriate to represent the CQ-composition of mappings given by st-tgds.

Corollary 11. *A* CQ*-composition of a finite number of mappings, each defined by a finite set of st-tgds, can be defined by a plain SO-tgd.*

Finally, the following property is a consequence of the fact that every plain SO-tgd is also an SO-tgd. More precisely, if a mapping $\mathcal{M}$ is specified by a plain SO-tgd, then from the results proved in [1] for SO-tgds, we have that $\mathcal{M}$ is logically equivalent to the composition of two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ specified by st-tgds. Thus, given that logical equivalence implies CQ-equivalence, we conclude that $\mathcal{M}$ is a CQ-composition of $\mathcal{M}_1$ and $\mathcal{M}_2$.

Corollary 12. *Let $\mathcal{M}$ be a mapping specified by a plain SO-tgd. Then there exist mappings $\mathcal{M}_1$ and $\mathcal{M}_2$, each specified by a finite set of st-tgds, such that $\mathcal{M}$ is a* CQ*-composition of $\mathcal{M}_1$ and $\mathcal{M}_2$.*

Theorem 10 together with Corollaries 11 and 12 show that plain SO-tgds are the right language for representing the CQ-composition of mappings given by st-tgds. We conclude this section by pointing out that by using some of the tools developed in [5] (see, for example, Theorem 3.12 in [5]), it is possible to prove that plain SO-tgds are also a good language for inversion.

Theorem 13. *Every mapping specified by a plain SO-tgd admits a maximum recovery.*

We omit here the proof of Theorem 13 based on the results from [5], as this theorem is also a corollary of Theorem 19 proved in Section 5, which shows that there exists a polynomial-time algorithm for computing the maximum recovery of a plain SO-tgd.

We end this section with a remark on computing the CQ-composition of mappings specified by st-tgds. Notice that the notion of CQ-composition is just a relaxation of the composition as defined in [14]. Thus, the same algorithm presented in [14] can be used to compute the CQ-composition of mappings specified by st-tgds. Nevertheless, the algorithm in [14] provides as output a not necessarily plain SO-tgd. One can combine the results in [14] and the construction in Lemma 9 to provide an algorithm for computing the CQ-composition that has a plain SO-tgd as output. Unfortunately, as we next discuss, the complexity of such algorithm would be high. The composition algorithm in [14] provides as output an SO-tgd which is of size exponential w.r.t. the size of the input mappings. Moreover, the construction in Lemma 9 is based on a result presented in [1] stating that every SO-tgd is logically equivalent to an SO-tgd without nesting of function symbols. In [1] the authors provide a procedure to show this result which, given an SO-tgd as input, computes a logically equivalent SO-tgd without nesting of function symbols which is of size exponential w.r.t. the size of the input. Thus, both procedures together plus the construction in Lemma 9 give rise to a doubly exponential time algorithm for computing a CQ-composition expressed as a plain SO-tgd. It is open whether one can device a single exponential time algorithm for this problem that has a plain SO-tgd as output.

4.3. Structural properties of plain SO-tgds

In this section, we study some structural properties of plain SO-tgds as a language for defining schema mappings. Borrowing the framework developed by ten Cate and Kolaitis [26,27], we make a case that plain SO-tgds can be considered as a good alternative for a mapping language which is to be used in practical applications, since, despite being more expressive than st-tgds, they satisfy most of the good properties that st-tgds enjoy. We also show how plain SO-tgds refine the picture introduced in [26,27], allowing us to answer an open question from the previous work.

Amongst all structural properties of mapping languages that have been studied in the literature, ten Cate and Kolaitis argue in [27] that three of the most fundamental ones for data exchange and data integration are *allowing for universal solutions*, *allowing for rewriting of conjunctive queries*, and *closure under target homomorphisms*. More precisely, given a mapping $\mathcal{M}$ from a schema **S** to a schema **T**, $\mathcal{M}$ is said to *admit universal solutions* if for every instance I of **S**, there exists a universal solution for I under $\mathcal{M}$, and $\mathcal{M}$ is said to *allow for conjunctive query rewriting* if for every union of conjunctive queries Q over **T**, there exists a union of conjunctive queries Q' over **S** such that $Q'(I) = \text{certain}_{\mathcal{M}}(Q, I)$ [27]. Moreover, $\mathcal{M}$ is said to be *closed under target homomorphisms* if whenever $(I, J) \in \mathcal{M}$ and there exists a homomorphism from J to an instance J' of **T**, then $(I, J') \in \mathcal{M}$ [27].

It has been shown that if a mapping is specified by a finite set of st-tgds, then it admits universal solutions, it allows for conjunctive query rewriting and it is closed under target homomorphisms [26]. On the other hand, if a mapping is specified by an SO-tgd, then it satisfies the first two conditions but it is not closed under target homomorphisms [14, 26]. Interestingly, we show in the following theorem that plain SO-tgds form a second-order language satisfying the three fundamental properties defined above.

Theorem 14. *The language of plain SO-tgds satisfies the following properties*:

(1) *Every mapping specified by a plain SO-tgd is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting.*
(2) *There exists a plain SO-tgd that is not expressible in first-order logic.*

Proof. (1) Let $\mathcal{M}$ be a mapping defined by a plain SO-tgd λ from a schema **S** to a schema **T**. From the fact that every mapping specified by an SO-tgd admits universal solutions and allows for conjunctive query rewriting, we conclude that these two properties hold for $\mathcal{M}$. Thus, it only remains to show that $\mathcal{M}$ is closed under target homomorphisms.

Assume that λ is of the form $\exists \bar{f}\ (\forall \bar{x}_1\ (\varphi_1 \to \psi_1) \wedge \cdots \wedge \forall \bar{x}_n\ (\varphi_n \to \psi_n))$. Moreover, assume that I is an instance of **S** and J, J' are instances of **T** such that $(I, J) \models \lambda$ and there exists a homomorphism h from J to J'. Next we prove that $(I, J') \models \lambda$. As for the case of the proof of Lemma 9, given a formula α mentioning variables from $\bar{x}$ and plain terms built from $\bar{x}$ and $\bar{f}$, and a tuple of elements $\bar{a}$, we use $\alpha[\bar{x} \to \bar{a}]$ to denote the formula obtained from α by replacing every variable in $\bar{x}$ according to the assignment $\bar{x} \to \bar{a}$.

Given that $(I, J) \models \lambda$, we have that there exists an interpretation $\bar{f}^{(I,J)}$ over (I, J) for the function symbols in $\bar{f}$ such that for every $i \in \{1, \ldots, n\}$, if I satisfies the formula $\varphi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J)}$, then J satisfies formula $\psi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J)}$. Consider now the interpretation $\bar{f}^{(I,J')}$ of $\bar{f}$ over (I, J') defined as follows. For every function symbol f in $\bar{f}$ and tuple of elements $\bar{b}$ from (I, J), let $f^{(I,J')}(\bar{b}) = h(f^{(I,J)}(\bar{b}))$, if $f^{(I,J)}(\bar{b})$ belongs to the domain of J, or otherwise $f^{(I,J')}(\bar{b}) = \bot$, where $\bot$ is any arbitrary element from (I, J'). Next we show that (I, J') satisfies λ with interpretation $\bar{f}^{(I,J')}$.

Assume that I satisfies $\varphi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J')}$, for $i \in \{1, \dots, n\}$. Given that λ is a plain SO-tgd, we have that φ_i does not mention any function symbol and, hence, I satisfies $\varphi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J)}$. Then we know that $J \models \psi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J)}$. Let $R(\bar{a}')$ be an arbitrary conjunct in $\psi_i[\bar{x}_i \to \bar{a}]$ with function symbols interpreted according to $\bar{f}^{(I,J)}$. Then we know that $J \models R(\bar{a}')$. Thus, given that h is a homomorphism from J to J', we have that $J' \models R(h(\bar{a}'))$.

Furthermore, since $J \models \psi_i[\bar{x}_i \to \bar{a}]$, it follows that the interpretation $\bar{f}^{(I,J')}$ is defined so that each conjunct of form $R(\bar{x}_i)$ in ψ_i corresponds to $R(h(\bar{a}'))$ in $\psi_i[\bar{x}_i \to \bar{a}]$, when function symbols are interpreted according to $\bar{f}^{(I,J')}$. Hence, $J' \models \psi_i[\bar{x}_i \to \bar{a}]$ with interpretation $\bar{f}^{(I,J')}$, which was to be shown.

(2) Consider a plain SO-tgd λ from $\mathbf{R}_1 = \{E(\cdot,\cdot)\}$ to $\mathbf{R}_2 = \{T(\cdot,\cdot)\}$ defined as:

$$\exists f \, \big(\forall x \forall y \; E(x, y) \to T\big(f(x), f(y)\big)\big).$$

Next we show that λ is not expressible by a first-order logic sentence over the vocabulary $\mathbf{R}_1 \cup \mathbf{R}_2$. For the sake of contradiction, assume that λ is expressible in first-order logic, and let Φ be a first-order sentence over $\mathbf{R}_1 \cup \mathbf{R}_2$ that is logically equivalent to λ. Let u and v be two variables not mentioned in Φ and consider the sentence Φ' obtained from Φ by replacing every relational atom $T(x, y)$ by formula $(x = u \wedge y = v) \vee (x = v \wedge y = u)$. Now let Ψ be the first-order sentence over $\mathbf{R}_1$ defined as:

$$\exists u \exists v \, \big(u \neq v \wedge \Phi'\big).$$

It is not difficult to see that $I \models \Psi$ if and only if for the instance J of $\mathbf{R}_2$ such that $T^J = \{(1, 2), (2, 1)\}$, it holds that $(I, J) \models \Phi$. Thus, given that Φ is logically equivalent to λ, we obtain that $I \models \Psi$ if and only if there exists an interpretation $f^\star$ for the function symbol f such that for every element c in I, it holds that $f^\star(c)$ is either 1 or 2, and for every tuple $(a, b) \in E^I$, it holds that $f^\star(a) \neq f^\star(b)$. Therefore, we have that $I \models \Psi$ if and only if the graph represented by I is 2-colorable. This leads to a contradiction since 2-colorability is not expressible in first-order logic [21]. □

Notice that by using a similar argument as the one used in the above proof, we can obtain a contradiction by using 3-colorability instead of 2-colorability. Thus, by a result by Dawar [8], we obtain that λ cannot even be defined in the finite-variable infinitary logic $\mathcal{L}^{\omega}_{\infty\omega}$, which is strictly more expressive than first-order logic (see [21] for a definition of $\mathcal{L}^{\omega}_{\infty\omega}$).

It is important to notice that ten Cate and Kolaitis [26,27] were interested in characterizing schema-mapping languages in terms of the structural properties that they satisfy. In particular, they were interested in a language capable of specifying all the mappings that are closed under target homomorphisms, admit universal solutions, and allow for conjunctive query rewriting. The language of nested st-tgds is a fragment of first-order logic introduced by Fuxman et al. in [19]. This language is strictly more expressive than st-tgds, and still satisfies the three structural properties just mentioned, so it is considered in [26,27] as a plausible candidate to exactly characterize the class of mappings that are closed under target homomorphisms, admit universal solutions, and allow for conjunctive query rewriting. In fact, ten Cate and Kolaitis posed the following question in [26,27]:

Question 15. *(See [26,27].) Is it the case that a schema mapping is definable by a set of nested st-tgds if and only if it is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting*?

For our purposes, it is not important to formally define the language of nested st-tgds (we refer the reader to [19,26] for details), it is just important to mention that the language of nested st-tgds is strictly less expressive than the language of plain SO-tgds. In particular, by the results in [19,26] it is easy to conclude that every set of nested st-tgds can be specified using plain SO-tgds and, moreover, Theorem 14 (2) shows that there are plain SO-tgds that are not equivalent to any finite set of nested st-tgds (as the latter are a fragment of first-order logic). Thus, Theorem 14 gives a negative answer to Question 15 as every mapping specified by a plain SO-tgd is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting. It is worth mentioning that it is an open problem whether the language of plain SO-tgds exactly characterizes the class of mappings that satisfy the aforementioned structural properties. In fact, the following questions are still open.

Questions 16.

- *Is it the case that a schema mapping is definable by a set of nested st-tgds if and only if it is definable in first-order logic, is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting*?
- *Is it the case that a schema mapping is definable by a plain SO-tgd if and only if it is closed under target homomorphisms, admits universal solutions, and allows for conjunctive query rewriting*?

We conclude this section by going a step forward in our understanding of plain SO-tgds. More precisely, we show in the first place that these dependencies correspond exactly to the fragment of SO-tgds defining mappings that are closed under target homomorphisms.

Proposition 17. *If $\mathcal{M}$ is a mapping defined by an SO-tgd λ and $\mathcal{M}$ is closed under target homomorphisms, then λ is logically equivalent to a plain SO-tgd.*

Proof. Assume that λ is an SO-tgd from a schema $\mathbf{S}$ to a schema $\mathbf{T}$. From the proof of Lemma 9, we know that there exists a plain SO-tgd λ' such that $\text{chase}_\lambda(I) = \text{chase}_{\lambda'}(I)$ for every instance I of $\mathbf{S}$. Given that $\mathcal{M}$ is closed under target homomorphisms and for every instance I of $\mathbf{S}$, $\text{chase}_\lambda(I)$ is a universal solution for I under $\mathcal{M}$, we have that for every instance I of $\mathbf{S}$ and every instance J of $\mathbf{T}$: $(I, J) \models \lambda$ if and only if there is a homomorphism from $\text{chase}_\lambda(I)$ to J. In the same way, we can conclude that for every instance I of $\mathbf{S}$ and every instance J of $\mathbf{T}$: $(I, J) \models \lambda'$ if and only if there is a homomorphism from $\text{chase}_{\lambda'}(I)$ to J (notice that the mapping specified by λ' is closed under target homomorphisms as λ' is a plain SO-tgd). Thus, given that $\text{chase}_\lambda(I) = \text{chase}_{\lambda'}(I)$ for every instance I of $\mathbf{S}$, we conclude that for every instance I of $\mathbf{S}$ and every instance J of $\mathbf{T}$: $(I, J) \models \lambda$ if and only if $(I, J) \models \lambda'$. Therefore, we have that λ and λ' are logically equivalent, which was to be shown. □

In the second place, we point out that the language of plain SO-tgds satisfies a property that is arguably the most fundamental property satisfied by the class of mappings given by st-tgds, and which also help us to show that an important problem associated to this class of dependencies is decidable. More precisely, given a mapping $\mathcal{M}$ from a schema $\mathbf{S}$ to a schema $\mathbf{T}$, we say that the spaces of solutions under $\mathcal{M}$ are *characterized by universal solutions* if for every instance I of $\mathbf{S}$ and every universal solution J for I under $\mathcal{M}$, it holds that K is a solution for I under $\mathcal{M}$ if and only if there exists a homomorphism from J to K.[1] This property is satisfied by every mapping specified by a finite set of st-tgds. Since plain SO-tgds, just as st-tgds, are closed under target homomorphisms and admit universal solutions, every mapping defined by a plain SO-tgd also satisfies this fundamental property. Surprisingly, this fact can be used to obtain an algorithm for deciding whether a plain SO-tgd is equivalent to a finite set of st-tgds, which is an important problem in the area of schema mapping optimization [13].

Proposition 18. *There is an algorithm that, given a plain SO-tgd λ and a finite set Σ of st-tgds, decides whether λ is logically equivalent to Σ.*

Proof. Assume that λ is a plain SO-tgd from a schema $\mathbf{S}$ to a schema $\mathbf{T}$, and that Σ is a finite set of st-tgds from $\mathbf{S}$ to $\mathbf{T}$. Then let $\mathcal{M}_1 = (\mathbf{S}, \mathbf{T}, \lambda)$ and $\mathcal{M}_2 = (\mathbf{S}, \mathbf{T}, \Sigma)$. Given that $\mathcal{M}_1$, $\mathcal{M}_2$ admit universal solutions and are closed under target homomorphisms, we have from the results in [13] that λ is logically equivalent to Σ if and only if $\mathcal{M}_1 \equiv_{\text{CQ}} \mathcal{M}_2$. Thus, all that needs to be shown is the existence of an algorithm to decide whether $\mathcal{M}_1$ is CQ-equivalent to $\mathcal{M}_2$. But this result has been recently established by Fagin and Kolaitis [11] for the more general case where $\mathcal{M}_1$ is specified by an SO-tgd and $\mathcal{M}_2$ is specified by a finite set of st-tgds. This concludes the proof of the proposition. □

It is important to notice that the previous result is in sharp contrast with the situation for the language of SO-tgds. In fact, an inspection of the proof of Theorem 1 in [18] reveals that the problem of verifying, given an SO-tgd λ and a finite set Σ of st-tgds, whether λ is logically equivalent to Σ is undecidable.

5. Inverting plain SO-tgds in polynomial time

We show in this section that every st-mapping specified by a plain SO-tgd has a maximum recovery. More specifically, we present in this section a polynomial time algorithm that, given a plain SO-tgd, returns a maximum recovery that is expressed in a language that extends plain SO-tgds with some extra features. Notice that every mapping specified by st-tgds can also be specified by a plain SO-tgd. Thus, our algorithm can be used to compute maximum recoveries of st-tgds in polynomial time. Unfortunately, the gain in time complexity comes with the price of a stronger and less manageable mapping language used to express maximum recoveries.

We start by giving some of the intuition behind the algorithm. Consider the following plain SO-tgd:

$$\exists f \exists g \, \big(\forall x \forall y \forall z \, \big(R(x, y, z) \rightarrow T\big(x, f(y), f(y), g(x, z)\big)\big)\big). \tag{6}$$

When exchanging data with an SO-tgd like (6), the standard assumption is that every application of a function symbol generates a fresh value [14]. For example, consider a source instance $\{R(1, 2, 3)\}$. When we exchange data with (6), we obtain a canonical target instance $\{T(1, a, a, b)\}$, where $a = f(2)$, $b = g(1, 3)$, and $a \neq b$. The intuition behind our algorithm is to produce a reverse mapping that focuses on this canonical target instance to recover as much source data as possible.

[1] Notice that the definition of universal solution only guarantees that if K is a solution for I under $\mathcal{M}$, then there is a homomorphism from J to K.

Thus, in order to invert a dependency like (6), we consider three unary functions f_1, g_1 and g_2. The idea is that f_1 represents the inverse of function f, while (g_1, g_2) represents the inverse of g. Notice that since g has two arguments, we need to use two functions to represent its inverse. Thus, considering the above example, the intended meaning of the functions is $f_1(a) = 2$, $g_1(b) = 1$, and $g_2(b) = 3$. With this in mind, we can represent an inverse of the plain SO-tgd (6) with a sentence of the form:

$$\exists f_1 \exists g_1 \exists g_2 \big(\forall u \forall v \forall w \ \big(T(u, v, v, w) \wedge \mathbf{C}(u) \rightarrow R\big(u, f_1(v), g_2(w)\big) \wedge u = g_1(w)\big)\big), \tag{7}$$

where $\mathbf{C}(\cdot)$ denotes a built-in unary predicate such that $\mathbf{C}(a)$ holds if and only if a is a constant, that is, $a \in \mathbf{C}$. Notice that, if we use dependency (7) to exchange data back from instance $\{T(1, a, a, b)\}$, we obtain an instance $\{R(1, f_1(a), g_2(b))\}$. The equality $u = g_1(w)$ has been added in order to ensure the correct interpretation of g_1 as the inverse function of g. In the example, the equality ensures that $g_1(b)$ is 1. Notice also that dependency (7) uses predicate $\mathbf{C}(\cdot)$ since we know from (6) that u must be a constant value coming from the original instance. Predicate $\mathbf{C}(\cdot)$ is usually required in order to express inverses of mappings when nulls are only allowed in the target [10,15,5].

In order to obtain a correct algorithm we need another technicality. We have mentioned that when exchanging data with SO-tgds, we assume that every application of a function produces a fresh value. In the above example, we have that value a is the result of applying f to 2, thus, we know that value a cannot be obtained with any other function. In particular, a cannot be obtained as an application of function g. Thus, when exchanging data back we should ensure that at most one inverse function is applied to every possible target value. Similarly, we know that every constant value in the target instance that came from the source instance cannot be obtained as the result of applying any function. In the above example, value 1 in the target came directly from the source and is not obtained by applying f or g. Thus, we also need to ensure that no inverse function is applied to a value that came from the source instance. We ensure all this by using an additional unary function $f_\star$. In the above example, whenever we apply function f_1 to some value v, we require that $f_1(v) = f_\star(v)$ and that $g_1(v) \neq f_\star(v)$. Similarly, whenever we apply function g_1 to some value w, we require that $g_1(w) = f_\star(w)$ and that $f_1(w) \neq f_\star(w)$. Thus, for example, if we apply f_1 to value a, we require that $f_1(a) = f_\star(a)$ and that $g_1(a) \neq f_\star(a)$. Notice that this forbids the application of g_1 to a, since, if that were the case, we would require that $g_1(a) = f_\star(a)$, which contradicts the previous requirement $g_1(a) \neq f_\star(a)$. Moreover, for every value u that is copied from target to source, we require that $f_1(u) \neq f_\star(u)$ and $g_1(u) \neq f_\star(u)$, ensuring that neither f_1 nor g_1 is applied to u. Our algorithm adds these equalities and inequalities as conjuncts in the conclusions of dependencies. Considering the example, our algorithm adds

$$\begin{aligned}\Phi(u, v, w) \equiv\ & f_1(v) = f_\star(v) \ \wedge \ g_1(v) \neq f_\star(v) \\ & \wedge \ g_1(w) = f_\star(w) \ \wedge \ f_1(w) \neq f_\star(w) \\ & \wedge \ f_1(u) \neq f_\star(u) \ \wedge \ g_1(u) \neq f_\star(u)\end{aligned}$$

to the right-hand side of the implication of dependency (7), and also adds the existential quantification over function $f_\star$. The final SO sentence that specifies a maximum recovery of the plain SO-tgd (6) is:

$$\exists f_\star \exists f_1 \exists g_1 \exists g_2 \big[\forall u \forall v \forall w \ \big(T(u, v, v, w) \wedge \mathbf{C}(u) \rightarrow R\big(u, f_1(v), g_2(w)\big) \wedge u = g_1(w) \wedge \Phi(u, v, w)\big)\big].$$

Before presenting our algorithm, we make some observations. Although we have assumed in the above explanation that every application of a function generates a fresh value, we remark that this assumption has only been used as a guide in the design of our algorithm. In fact, it is shown in Theorem 19 that the algorithm presented in this section produces maximum recoveries for the general case, where no assumption about the function symbols is made.

In the following sections, we formalize our algorithm to compute maximum recoveries of plain SO-tgds.

5.1. Auxiliary procedures

We start by fixing some notation. Given a plain SO-tgd λ, we denote by F_λ the set of function symbols that occur in λ. We also consider a set of function symbols F'_λ constructed as follows. For every n-ary function symbol f in F_λ, the set F'_λ contains n unary function symbols $f_1, \ldots, f_n$. Additionally, F'_λ contains a unary function symbol $f_\star$. For example, for plain SO-tgd (6), $F_\lambda = \{f, g\}$ and $F'_\lambda = \{f_1, g_1, g_2, f_\star\}$.

We now describe procedures CreateTuple, EnsureInv, and Safe. These procedures are the building blocks of the algorithm that computes a maximum recovery of a plain SO-tgd. Procedure CreateTuple($\bar{t}$) receives as input a tuple $\bar{t} = (t_1, \ldots, t_n)$ of plain terms. Then it builds an n-tuple of variables $\bar{u} = (u_1, \ldots, u_n)$ such that, if $t_i = t_j$ then u_i and u_j are the same variable, and they are distinct variables otherwise. For example, consider the right-hand side of the implication of dependency (6). In the argument of relation T we have the tuple of terms $\bar{t} = (x, f(y), f(y), g(x, z))$. In this case, we have that procedure CreateTuple($\bar{t}$) returns a tuple of the form (u, v, v, w). Notice that we have used this tuple as the argument of T in the left-hand side of the implication of dependency (7). Tuple $\bar{u}$ created with CreateTuple is used as an input in the following two procedures.

We now formalize procedure EnsureInv that outputs a formula that guarantees the correct use of the inverse function symbols.

Procedure: ENSUREINV($\lambda, \bar{u}, \bar{s}$).
Input: A plain SO-tgd λ, an n-tuple $\bar{u} = (u_1, \dots, u_n)$ of (not necessarily distinct) variables, and an n-tuple $\bar{s} = (s_1, \dots, s_n)$ of plain terms built from F_λ and a tuple of variables $\bar{y}$.
Output: A formula Q_e consisting of conjunctions of equalities between terms built from F'_λ, $\bar{u}$, and $\bar{y}$.

```
1. Construct formula $Q_e$ as follows. For every $i \in \{1, \dots, n\}$ do the following:
   – If $s_i$ is a variable $y$ of $\bar{y}$, then add equality $u_i = y$ as a conjunct in $Q_e$.
   – If $s_i$ is a term of the form $f(y_1, \dots, y_k)$, then add the conjunction of equalities

         $f_1(u_i) = y_1 \wedge \cdots \wedge f_k(u_i) = y_k$

     to $Q_e$, where $f_1, \dots, f_k$ are the $k$ unary functions in $F'_\lambda$ associated with $f$.
2. Return $Q_e$. □
```

As an example, let λ be the dependency (6), $\bar{s} = (x, f(y), f(y), g(x, z))$ the tuple of terms in the right-hand side of the implication of λ, and $\bar{u} = (u, v, v, w)$. When running the procedure ENSUREINV($\lambda, \bar{u}, \bar{s}$), we have that $u_4 = w$ and $s_4 = g(x, z)$. Thus, in the loop of Step 1, the conjunction $g_1(w) = x \wedge g_2(w) = z$ is added to formula Q_e. The final output of the procedure is in this case:

$$u = x \ \wedge \ f_1(v) = y \ \wedge \ g_1(w) = x \ \wedge \ g_2(w) = z. \tag{8}$$

Finally, we describe procedure SAFE, which is used to guarantee that a value in the target cannot be generated by two distinct functions.

Procedure: SAFE($\lambda, \bar{u}, \bar{s}$)
Input: A plain SO-tgd λ, an n-tuple $\bar{u} = (u_1, \dots, u_n)$ of (not necessarily distinct) variables, and an n-tuple $\bar{s} = (s_1, \dots, s_n)$ of plain terms built from F_λ and a tuple of variables $\bar{y}$.
Output: A formula Q_s consisting of equalities and inequalities between terms built from F'_λ and $\bar{u}$.

```
1. Construct formula $Q_s$ as follows. For every $i \in \{1, \dots, n\}$ do the following:
   – If $s_i$ is a term of the form $f(y_1, \dots, y_k)$, then add the following conjuncts to $Q_s$:
     – The equality $f_\star(u_i) = f_1(u_i)$.
     – The inequality $f_\star(u_i) \neq g_1(u_i)$, for every function symbol $g$ in $F_\lambda$ different from $f$.
   – If $s_i$ is a variable, then for every function symbol $f$ in $F_\lambda$, add the inequality $f_\star(u_i) \neq f_1(u_i)$ as a conjunct to $Q_s$.[2]
2. Return $Q_s$. □
```

Considering λ as the dependency (6), $\bar{s}$ the tuple of terms $(x, f(y), f(y), g(x, z))$ and $\bar{u} = (u, v, v, w)$, the algorithm SAFE($\lambda, \bar{u}, \bar{s}$) returns:

$$f_1(u) \neq f_\star(u) \ \wedge \ g_1(u) \neq f_\star(u) \ \wedge \ g_1(w) = f_\star(w) \ \wedge \ f_1(w) \neq f_\star(w) \ \wedge \ f_1(v) = f_\star(v) \ \wedge \ g_1(v) \neq f_\star(v). \tag{9}$$

It is important to notice that all the procedures presented in this section work in polynomial time with respect to the size of their inputs. Next we describe in detail our polynomial-time algorithm to compute a maximum recovery of a mapping specified by a plain SO-tgd.

5.2. Building the maximum recovery

The algorithm for inverting plain SO-tgds resembles the algorithm proposed in [5] to compute a maximum recovery of a mapping given by a set of full st-tgds. Recall that a full st-tgd is an st-tgd that does not include any existential quantifier in its conclusion. As an example, the following is a full st-tgd:

$$\forall x \forall y \left(P(x, y) \rightarrow S(x) \wedge S(y)\right).$$

For mappings specified by this type of dependencies, an algorithm was given in [5] to compute maximum recoveries in polynomial time. Informally, this algorithm works in two steps (see [5] for details). In the first step, the algorithm rewrites each dependency into a new set of full st-tgds where each rule contains a single atom in the head. This is a standard procedure in order to present a full st-tgd as a *GAV mapping* [12,20]. For the full dependency above, the new set of rules is the following:

[2] This technical step is new with respect to the algorithm given in [4], but notice that the output language remains the same. The inequalities added in this step are needed in the proof of correctness of the algorithm given in Theorem 19.

$$\forall x \forall y \, \big(P(x, y) \to S(x)\big),$$
$$\forall x \forall y \, \big(P(x, y) \to S(y)\big).$$

Notice that the new set of dependencies is logically equivalent to the previous set of full dependencies. In the second step, the algorithm considers every atom $R(\bar{x})$ that is in the conclusion of a dependency and adds to the final output a dependency of the form $\forall \bar{x} \; (R(\bar{x}) \to \alpha(\bar{x}))$ where $\alpha(\bar{x})$ is a *rewriting of* $R(\bar{x})$ *over the source* [5,27] (see Section 4.3 for a formalization of this notion). For our previous example, a rewriting of $S(x)$ over the source is $\exists y_1 \; P(x, y_1) \; \vee \; \exists y_2 \; P(y_2, x)$. Thus, a maximum recovery for our example is:

$$\forall x \, \big(S(x) \to \exists y_1 \; P(x, y_1) \vee \exists y_2 \; P(y_2, x)\big).$$

The algorithm to compute maximum recoveries for plain SO-tgds works in a similar way. In the first part of the algorithm, we normalize the set of rules exactly like in the algorithm for full dependencies. Notice that for plain SO-tgds the function terms allow us to separate the conclusion of each dependency without altering the semantics of the mapping. Then in the second part, we define for each atom in the conclusion of a dependency a new rule that intuitively contains as disjuncts all the possible ways this atom could have been generated. The conclusion of this new rule is similar to a rewriting of the atom over the source, but it further considers the technicalities presented at the beginning of this section.

Before formally presenting our algorithm for computing maximum recoveries of plain SO-tgds, we need to introduce some additional notation. Let $\bar{t}$ and $\bar{s}$ be n-tuples of plain terms. Then we say that $\bar{t}$ *is subsumed by* $\bar{s}$ (or $\bar{s}$ *subsumes* $\bar{t}$) if, whenever the i-th component of $\bar{t}$ contains a variable, the i-th component of $\bar{s}$ also contains a variable. Notice that if $\bar{s}$ subsumes $\bar{t}$, then whenever the i-th component of $\bar{s}$ contains a non-atomic term, the i-th component of $\bar{t}$ also contains a non-atomic term. For example, the tuple of terms $(x, f(y), f(y), g(x, z))$ is subsumed by $(u, v, h(u), h(v))$.

The following algorithm computes a maximum recovery of a mapping specified by a plain SO-tgd in polynomial time.

```
Algorithm: PolySOInverse($\mathcal{M}$)
Input: An st-mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \lambda)$ with $\lambda$ a plain SO-tgd of the form $\exists \bar{f} \; (\sigma_1 \wedge \cdots \wedge \sigma_n)$.
Output: A ts-mapping $\mathcal{M}' = (\mathbf{T}, \mathbf{S}, \lambda')$ that specifies a maximum recovery of $\mathcal{M}$ such that $\lambda'$ is a sentence in SO.

 1. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, and $\Sigma'$ be empty.
 2. Normalize $\Sigma$ as follows. For every $i \in \{1, \ldots, n\}$ do:
    – If $\sigma_i$ is of the form $\forall \bar{x} \; (\varphi(\bar{x}) \to R_1(\bar{t}_1) \wedge \cdots \wedge R_\ell(\bar{t}_\ell))$, then replace $\sigma_i$ by $\ell$ dependencies $\forall \bar{x}_1 \; (\varphi_1(\bar{x}_1) \to R_1(\bar{t}_1)), \ldots,$
      $\forall \bar{x}_\ell \; (\varphi_\ell(\bar{x}_\ell) \to R_\ell(\bar{t}_\ell))$ such that for every $1 \leqslant i \leqslant \ell$:
      – $\bar{x}_i$ is exactly the tuple of variables shared by $\bar{x}$ and $\bar{t}_i$,
      – $\varphi_i(\bar{x}_i)$ is the formula obtained from $\varphi(\bar{x})$ by existentially quantifying the variables of $\bar{x}$ not mentioned in $\bar{x}_i$.
 3. For every $\sigma$ of the form $\forall \bar{x} \; (\varphi(\bar{x}) \to R(\bar{t}))$ in the normalized set $\Sigma$, where $\bar{t} = (t_1, \ldots, t_m)$ is a tuple of plain terms, do
    the following:
    (a) Let $\bar{u} = \text{CreateTuple}(\bar{t})$.
    (b) Let $prem_\sigma(\bar{u})$ be a formula defined as the conjunction of the atom $R(\bar{u})$ and the formulas $\mathbf{C}(u_i)$ for every $i$ such
        that $t_i$ is a variable.
    (c) Create a set of formulas $\Gamma_\sigma$ as follows. For every dependency $\forall \bar{y} \; (\psi(\bar{y}) \to R(\bar{s}))$ in $\Sigma$ such that $\bar{s}$ subsumes $\bar{t}$, do
        the following:
        – Let $Q_e = \text{EnsureInv}(\lambda, \bar{u}, \bar{s})$.
        – Let $Q_s = \text{Safe}(\lambda, \bar{u}, \bar{s})$.
        – Add to $\Gamma_\sigma$ the formula $\exists \bar{y} \; (\psi(\bar{y}) \wedge Q_e \wedge Q_s)$.
    (d) Add to $\Sigma'$ the dependency:

            $\forall \bar{u} \; \big(prem_\sigma(\bar{u}) \to \gamma_\sigma(\bar{u})\big)$

        where $\gamma_\sigma(\bar{u})$ is the disjunction of the formulas in $\Gamma_\sigma$.
 4. Let $\lambda' = \exists \bar{f}' \; (\bigwedge \Sigma')$, where $\bar{f}'$ is a tuple containing the function symbols in $F'_\lambda$. Return $\mathcal{M}' = (\mathbf{T}, \mathbf{S}, \lambda')$. □
```

As an example of the execution of the algorithm, let λ be the plain SO-tgd (6), and assume that $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \lambda)$ is the input of algorithm PolySOInverse. In Step 3 of algorithm PolySOInverse($\mathcal{M}$) we have to consider a single dependency $\sigma = R(x, y, z) \to T(x, f(y), f(y), g(x, z))$. Let $\bar{t}$ be the tuple of terms $(x, f(y), f(y), g(x, z))$. Recall that CreateTuple($\bar{t}$) is a tuple of the form (u, v, v, w) and, thus,

$$prem_\sigma(\bar{u}) = T(u, v, v, w) \wedge \mathbf{C}(u)$$

is built in Step 3.b. Notice that $\mathbf{C}(u)$ has been added since the first component of $\bar{t}$ is the variable x. Then in Step 3.c, we need to consider just dependency σ. Notice that $\bar{t}$ subsumes itself and, hence, formula

$$\exists x \exists y \exists z \, \big(R(x, y, z) \wedge Q_e \wedge Q_s\big)$$

is added to the set Γ_σ, where Q_e is the formula (8) and Q_s is the formula (9). Notice that $F'_\lambda = \{f_\star, f_1, g_1, g_2\}$, and thus, the following formula λ' is created in the last step of the algorithm:

$$\exists f_\star \exists f_1 \exists g_1 \exists g_2 \left[\forall u \forall v \forall w \left(T(u,v,v,w) \wedge \mathbf{C}(u) \rightarrow \exists x \exists y \exists z \left(R(x,y,z) \wedge u = x \wedge f_1(v) = y \right.\right.\right.$$
$$\left.\left.\left.\wedge\, g_1(w) = x \wedge g_2(w) = z \wedge Q_s\right)\right)\right]. \tag{10}$$

Notice that the existentially quantified variables can be eliminated from dependency (10). Thus, replacing formula Q_s and eliminating the existential quantification in the right-hand side of the implication, we obtain that dependency (10) is equivalent to:

$$\begin{aligned}\exists f_\star \exists f_1 \exists g_1 \exists g_2 \big[\forall u \forall v \forall w \big(T(u,v,v,w) \wedge \mathbf{C}(u) \rightarrow{} & R\big(u, f_1(v), g_2(w)\big) \wedge u = g_1(w)\\ & \wedge f_1(v) = f_\star(v) \wedge g_1(v) \neq f_\star(v)\\ & \wedge g_1(w) = f_\star(w) \wedge f_1(w) \neq f_\star(w)\\ & \wedge f_1(u) \neq f_\star(u) \wedge g_1(u) \neq f_\star(u)\big)\big],\end{aligned}$$

which specifies a maximum recovery of $\mathcal{M}$.

It is important to point out that the output of PolySOInverse is a second-order sentence that extends plain SO-tgds by using predicate $\mathbf{C}(\cdot)$ in the premise of the dependencies, and disjunctions, equalities and inequalities over plain terms, in the conclusions of the dependencies. Thus, the gain in time complexity when computing maximum recoveries of plain SO-tgds comes with the price of a stronger and less manageable mapping language.

We now prove the correctness of PolySOInverse.

Theorem 19. *Let $\mathcal{M}$ be an st-mapping specified by a plain SO-tgd λ. Then* PolySOInverse($\mathcal{M}$) *computes a mapping $\mathcal{M}'$ specified by an SO sentence such that $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$. Moreover, algorithm* PolySOInverse *works in polynomial time.*

Proof. Let λ be a plain SO-tgd of the form $\exists \bar{f}\ (\sigma_1 \wedge \cdots \wedge \sigma_k)$, and $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \lambda)$. Let Σ be the set $\{\sigma_1, \ldots, \sigma_k\}$. Notice that after Step 2 of the algorithm, we have that every σ in the (normalized) set Σ, is a formula of the form $\forall \bar{x}\ (\varphi(\bar{x}) \rightarrow R(\bar{t}(\bar{x})))$ where:

- $\varphi(\bar{x})$ is a CQ formula over $\mathbf{S}$ with $\bar{x}$ as tuple of free variables,
- R is an n-ary relation symbol in $\mathbf{T}$,
- $\bar{t}(\bar{x})$ is an n-tuple of plain terms constructed by using functions from $\bar{f}$ and variables from $\bar{x}$, and
- $\bar{x}$ is exactly the tuple of (distinct) variables that $\varphi(\bar{x})$ and $\bar{t}(\bar{x})$ share.

Notice that in the above notation we have made explicit the variables mentioned in the tuple of terms $\bar{t}(\bar{x})$. Additionally, we assume that all the formulas in Σ have pair-wise disjoint sets of variables. It is straightforward to see that the formula $\exists \bar{f}\ \bigwedge \Sigma$ obtained after the normalization step is logically equivalent to the original plain SO-tgd provided as input for the algorithm.

Before continuing with the proof, recall that F_λ is the set of function symbols in $\bar{f}$. Also recall that F'_λ is the set of function symbols constructed as follows. For every n-ary function symbol f in $\bar{f}$, the set F'_λ contains n unary function symbols $f_1, \ldots, f_n$. Additionally, F'_λ contains a new unary function symbol $f_\star$. In the rest of the proof we assume that $\bar{f}'$ is a tuple of function symbols containing exactly the function symbols mentioned in F'_λ.

The rest of the proof is divided in three parts. We first introduce some auxiliary notation regarding the procedures CreateTuple, EnsureInv and Safe, to simplify the exposition of the proof. Second, we prove, using our new notation, that the mapping $\mathcal{M}'$ obtained as the output of PolySOInverse($\mathcal{M}$) is a recovery of $\mathcal{M}$. Finally, we prove that $\mathcal{M}'$ is actually a maximum recovery of $\mathcal{M}$.

(I) Auxiliary notation.

We introduce some notation regarding the procedures CreateTuple, EnsureInv and Safe. First, given a tuple of terms $\bar{t}(\bar{x}) = (t_1(\bar{x}), \ldots, t_n(\bar{x}))$ we use $\bar{u}_{\bar{t}(\bar{x})}$ to denote the output of CreateTuple($\bar{t}(x)$). That is, $\bar{u}_{\bar{t}(\bar{x})}$ is an n-tuple of variables $(u_1, \ldots, u_n)$ such that, for $i \neq j$, if the i-th component of $\bar{t}(\bar{x})$ is equal to the j-th component of $\bar{t}(\bar{x})$, then u_i and u_j are the same variable, and they are different variables otherwise. Consider for example the tuple of terms $\bar{t}(x_1, x_2) = (x_1, f(x_1), f(x_1), x_1, x_2)$. In this case we have that $t_1(x_1,x_2) = t_4(x_1,x_2) = x_1$, $t_2(x_1,x_2) = t_3(x_1,x_2) = f(x_1)$, and $t_5(x_1,x_2) = x_2$, and then $\bar{u}_{\bar{t}(x_1,x_2)}$ is a tuple of the form $(u_1, u_2, u_2, u_1, u_3)$. Second, given an n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of not necessarily distinct variables, and an n-tuple $\bar{s}(\bar{y}) = (s_1(\bar{y}), \ldots, s_n(\bar{y}))$ of plain terms, we denote by $\nu(\bar{u}, \bar{s}(\bar{y}))$ the formula obtained as output of procedure EnsureInv ($\lambda, \bar{u}, \bar{s}(\bar{y})$). Thus, we have that for every i with $1 \leqslant i \leqslant n$ it holds that:

- If $s_i(\bar{y})$ is a variable y_m of $\bar{y}$, then $\nu(\bar{u}, \bar{s}(\bar{y}))$ contains the equality $u_i = y_m$ as a conjunction.

- If $s_i(\bar{y})$ is a non-atomic term $f(y_{m_1}, \ldots, y_{m_k})$, with f a k-ary function symbol of $\bar{f}$ and every y_{m_j} a variable in $\bar{y}$, then $\nu(\bar{u}, \bar{s}(\bar{y}))$ contains the conjunction of equalities $f_1(u_i) = y_{m_1} \wedge \cdots \wedge f_k(u_i) = y_{m_k}$, where $f_1, \ldots, f_k$ are the k unary functions in $\bar{f}'$ associated with f.

For example, let $\bar{u} = (u_1, u_2, u_1)$ and $\bar{s}(y_1, y_2, y_3) = (y_1, f(y_3, y_1, y_2), g(y_2))$. Notice that $s_1(y_1, y_2, y_3) = y_1$, $s_2(y_1, y_2, y_3) = f(y_3, y_1, y_2)$ and $s_3(y_1, y_2, y_3) = g(y_2)$. In this case we have that $\nu(\bar{u}, \bar{s}(y_1, y_2, y_3))$ is the formula

$$u_1 = y_1 \ \wedge\ f_1(u_2) = y_3 \ \wedge\ f_2(u_2) = y_1 \ \wedge\ f_3(u_2) = y_2 \ \wedge\ g_1(u_1) = y_2.$$

Third, we denote by $\omega(\bar{u}, \bar{s}(\bar{y}))$ the formula obtained as the output of Safe$(\lambda, \bar{u}, \bar{s}(\bar{y}))$. That is, we have that for every i with $1 \leqslant i \leqslant n$, if $s_i(\bar{y})$ is a term of the form $f(y_{m_1}, \ldots, y_{m_k})$ with f a k-ary function symbol of $\bar{f}$ and every y_{m_j} a variable in $\bar{y}$, then $\omega(\bar{u}, \bar{s}(\bar{y}))$ contains as conjuncts:

- the equality $f_\star(u_i) = f_1(u_i)$, and
- the inequality $f_\star(u_i) \neq g_1(u_i)$, for every function symbol g in $\bar{f}$ different from f,

and if $s_i(\bar{y})$ is a variable, then $\omega(\bar{u}, \bar{s}(\bar{y}))$ contains as conjuncts the inequalities $f_\star(u_i) \neq f_1(u_i)$ for every f in $\bar{f}$. We have introduced this notation only to make explicit the variables used in the tuple of terms $\bar{t}(\bar{x})$ and $\bar{s}(\bar{y})$ in the inputs of CreateTuple, EnsureInv, and Safe.

Finally, given a dependency σ in Σ of the form $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$, in this proof we denote by $\mathcal{C}_{R(\bar{t}(\bar{x}))}$ the set Γ_σ constructed in Step 3.c. That is, for every dependency of the form $\psi(\bar{y}) \to R(\bar{s}(\bar{y}))$ in Σ such that $\bar{s}(\bar{y})$ subsumes $\bar{t}(\bar{x})$, then $\mathcal{C}_{R(\bar{t}(x))}$ includes the formula

$$\exists \bar{y}\ \big(\psi(\bar{y}) \wedge \nu\big(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})\big) \wedge \omega\big(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})\big)\big).$$

Notice that in the above formula we are using our new notation for the outputs of CreateTuple$(\bar{t}(x))$, EnsureInv$(\lambda, \bar{u}, \bar{s}(\bar{y}))$, and Safe$(\lambda, \bar{u}, \bar{s}(\bar{y}))$.

With our new notation we have that the set Σ' constructed in PolySOInverse contains, for every $\sigma \in \Sigma$ of the form $\forall \bar{x}\ (\varphi(\bar{x}) \to R(\bar{t}(\bar{x})))$, a dependency σ' such that

- the premise of σ' is composed of the atom $R(\bar{u}_{\bar{t}(\bar{x})})$ and formulas $\mathbf{C}(u_i)$ for every i such that $t_i(\bar{x})$ is a variable x of $\bar{x}$, and
- the conclusion of σ' is the disjunction of all the formulas in $\mathcal{C}_{R(\bar{t}(\bar{x}))}$.

We are now ready to continue with the proof. For simplicity, in what follows we omit the universal quantification in the formulas in Σ and Σ'. That is, if σ is a formula in Σ of the form $\forall \bar{x}\ (\varphi(\bar{x}) \to R(\bar{t}(\bar{x})))$, we just write $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$ to denote σ. Let $\mathcal{M}$ be the st-mapping specified by the formula $\lambda = \exists \bar{f}\ \bigwedge \Sigma$, and $\mathcal{M}'$ the ts-mapping specified by the formula $\lambda' = \exists \bar{f}'\ \bigwedge \Sigma'$ constructed in the last step of algorithm PolySOInverse. We need to show that $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$.

(II) $\mathcal{M}'$ is a recovery of $\mathcal{M}$.

We now show that $\mathcal{M}'$ is a recovery of $\mathcal{M}$, that is, we show that $(I, I) \in \mathcal{M} \circ \mathcal{M}'$ for every source instance I. Let I be a source instance, and J the result of chasing I with λ. Recall that J is constructed as follows. For every σ in Σ of the form $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$ and for every tuple $\bar{a}$ of constant values such that $I \models \varphi(\bar{a})$, we include in J the tuple $R(\bar{t}(\bar{a}))$. Notice that in this procedure, every ground term is viewed as a distinct value (for example, $f(a)$ and $g(a)$ are considered to be distinct values), and every ground non-atomic term is considered to be a null value. We claim that $(J, I) \in \mathcal{M}'$ which proves that $(I, I) \in \mathcal{M} \circ \mathcal{M}'$. In order to show that $(J, I) \models \lambda'$, we need to prove that there exists an *interpretation* for the functions of $\bar{f}'$ such that $(J, I) \models \bigwedge \Sigma'$. For every k-ary function f in $\bar{f}$ consider the interpretation of f_i with $1 \leqslant i \leqslant k$ as follows:

- for every k-tuple $(a_1, \ldots, a_k)$, we have that $f_i(f(a_1, \ldots, a_k)) = a_i$,
- f_i is an arbitrary *fresh* value in every other case.

That is f_i is interpreted as the *projection* of f over component i. Additionally, we interpret function $f_\star$ as follows. For every k-ary function f in $\bar{f}$ we let $f_\star(f(a_1, \ldots, a_k)) = a_1$, and $f_\star$ is an arbitrary fresh value in every other case. Notice that this interpretation is well defined since every ground term produced when chasing I, is viewed as a distinct value. We show now that with this interpretation, it holds that $(J, I) \models \bigwedge \Sigma'$.

Let σ' be a formula in Σ'. We need to show that (J, I) satisfies σ'. Assume that σ' was created from a formula in Σ of the form $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$. Assume that R is an n-ary relation symbol. Then σ' is of the form $R(\bar{u}_{\bar{t}(\bar{x})}) \wedge \mathbf{C}(\bar{u}') \to \alpha(\bar{u}_{\bar{t}(\bar{x})})$ with $\bar{u}_{\bar{t}(\bar{x})}$ an n-tuple of (not necessarily distinct) variables, $\bar{u}' \subseteq \bar{u}_{\bar{t}(\bar{x})}$, and $\alpha(\bar{u}_{\bar{t}(\bar{x})})$ the disjunction of the formulas in $\mathcal{C}_{R(\bar{t}(\bar{x}))}$. Now, suppose that there exists an n-tuple $\bar{b}$ of ground terms such that $J \models R(\bar{b}) \wedge \mathbf{C}(\bar{b}')$ (with $\bar{b}'$ the corresponding assignment to $\bar{u}'$ that derives from the assignment of $\bar{b}$ to $\bar{u}_{\bar{t}(\bar{x})}$). We must show that $I \models \alpha(\bar{b})$. Since J is the result of chasing I with λ, we know that there exists a formula $\psi(\bar{y}) \to R(\bar{s}(\bar{y}))$ that is used to generate $R(\bar{b})$ in J. Then there exists a tuple $\bar{a}$ of constants such that $I \models \psi(\bar{a})$ and $\bar{s}(\bar{a}) = \bar{b}$. Now, given that $\mathbf{C}(\bar{b}')$ holds and $\bar{s}(\bar{a}) = \bar{b}$, we conclude that $\bar{s}(\bar{y})$ subsumes $\bar{t}(\bar{x})$. Consequently, the formula

$$\beta(\bar{u}_{\bar{t}(\bar{x})}) = \exists \bar{y}\ \big(\psi(\bar{y}) \wedge \nu\big(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})\big) \wedge \omega\big(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})\big)\big)$$

belongs to $\mathcal{C}_{R(\bar{t}(\bar{x}))}$, and then, it is a disjunct in $\alpha(\bar{u}_{\bar{t}(\bar{x})})$. We claim that $I \models \beta(\bar{b}) = \beta(\bar{s}(\bar{a}))$ and then $I \models \alpha(\bar{b})$. Notice that $I \models \psi(\bar{a})$ and by the interpretation of the functions of $\bar{f}'$ it is straightforward to see that $\nu(\bar{s}(\bar{a}), \bar{s}(\bar{a}))$ and $\omega(\bar{s}(\bar{a}), \bar{s}(\bar{a}))$ hold. Then we have that $I \models \beta(\bar{b})$ with the chosen interpretation for the functions in $\bar{f}'$ by using tuple $\bar{a}$ as the witness for the tuple $\bar{y}$ of existentially quantified variables, and then $I \models \alpha(\bar{b})$. We have shown that, with the chosen interpretation for the functions in $\bar{f}'$, it holds that (J, I) satisfies every formula in Σ', and then (J, I) satisfies $\bigwedge \Sigma'$, which was to be shown.

(III) $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$.

We have shown that $\mathcal{M}'$ is a recovery of $\mathcal{M}$. To complete the proof that $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$, it is enough to show that, if $(I_1, I_2) \in \mathcal{M} \circ \mathcal{M}'$ then $\mathrm{Sol}_{\mathcal{M}}(I_2) \subseteq \mathrm{Sol}_{\mathcal{M}}(I_1)$ (see Proposition 3.8 in [5]). To simplify the exposition we introduce some notation. A *ground plain term* is a term of the form $f(a_1, \ldots, a_k)$ where $a_1, \ldots, a_k$ are values from some domain. Let $\bar{p}$ be a tuple of ground plain terms constructed by using function symbols from a tuple $\bar{g}$, and let $\bar{g}^0$ be an interpretation for the function symbols in $\bar{g}$. We write $\bar{p}[\bar{g} \mapsto \bar{g}^0]$ to denote the tuple obtained by replacing every ground plain term using a function symbol in $\bar{g}$ by its corresponding interpretation in $\bar{g}^0$. Similarly, if γ is a conjunction of atoms mentioning ground plain terms, we write $\gamma[\bar{g} \mapsto \bar{g}^0]$ to denote the conjunction obtained by replacing every ground plain term by its corresponding interpretation in $\bar{g}^0$. Abusing of the notation, for a first-order formula α that mentions plain terms constructed from function symbols in $\bar{g}$ we write $I \models \alpha[\bar{g} \mapsto \bar{g}^0]$ to denote that I satisfies α with the interpretation $\bar{g}^0$ for the function symbols in $\bar{g}$.

To continue with the proof, let I_1 and I_2 be source instances such that $(I_1, I_2) \in \mathcal{M} \circ \mathcal{M}'$. In order to prove that $\mathrm{Sol}_{\mathcal{M}}(I_2) \subseteq \mathrm{Sol}_{\mathcal{M}}(I_1)$ assume that $(I_2, J^\star) \in \mathcal{M}$. We need to show that $(I_1, J^\star) \in \mathcal{M}$. Since $(I_1, I_2) \in \mathcal{M} \circ \mathcal{M}'$, there exists a target instance J such that $(I_1, J) \in \mathcal{M}$ and $(J, I_2) \in \mathcal{M}'$. Thus, we have that $(I_1, J) \models \exists \bar{f}\ \bigwedge \Sigma$, and $(J, I_2) \models \exists \bar{f}'\ \bigwedge \Sigma'$. Then we know that there exist an interpretation $\bar{f}^{(I_1,J)}$ for the functions in $\bar{f}$, and an interpretation $\bar{f}'^{(J,I_2)}$ for the functions in $\bar{f}'$, such that $(I_1, J) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_1,J)}]$ and $(J, I_2) \models (\bigwedge \Sigma')[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$. Moreover, since $(I_2, J^\star) \in \mathcal{M}$ we know that there exists an interpretation $\bar{f}^{(I_2,J^\star)}$ for the functions in $\bar{f}$, such that $(I_2, J^\star) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$. We need to show that there exists an interpretation $\bar{f}^{(I_1,J^\star)}$ for $\bar{f}$, such that $(I_1, J^\star) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$. We describe now how to construct $\bar{f}^{(I_1,J^\star)}$ from $\bar{f}^{(I_1,J)}$, $\bar{f}'^{(J,I_2)}$ and $\bar{f}^{(I_2,J^\star)}$. Let f be a k-ary function symbol in $\bar{f}$, and let $\bar{a}$ be a k-tuple of constant values mentioned in I_1. Define $f^{(I_1,J^\star)}(\bar{a})$ as follows:

- Assume that there exists a unique function symbol g in $\bar{f}$, such that its associated function g_1 in $\bar{f}'$ satisfies:

$$f_\star^{(J,I_2)}\big(f^{(I_1,J)}(\bar{a})\big) = g_1^{(J,I_2)}\big(f^{(I_1,J)}(\bar{a})\big). \tag{11}$$

  Then, we let

$$f^{(I_1,J^\star)}(\bar{a}) = g^{(I_2,J^\star)}\big(g_1^{(J,I_2)}\big(f^{(I_1,J)}(\bar{a})\big), \ldots, g_k^{(J,I_2)}\big(f^{(I_1,J)}(\bar{a})\big)\big).$$

- Otherwise, if there is no function symbol in $\bar{f}$ satisfying equality (11), or there is more than one function symbol in $\bar{f}$ satisfying (11), then $f^{(I_1,J^\star)}(\bar{a}) = f^{(I_1,J)}(\bar{a})$.

We show next that, with $\bar{f}^{(I_1,J^\star)}$ as defined above, it holds that $(I_1, J^\star) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$.

Let σ be a formula in Σ of the form $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$, with R an n-ary relation symbol, and assume that $I_1 \models \varphi(\bar{a})$ for some tuple $\bar{a}$ of constant values. We need to show that $J^\star \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$. Now, since $I_1 \models \varphi(\bar{a})$ and $(I_1, J) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_1,J)}]$, we know that $J \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J)}]$. By construction of Σ', there exists a formula σ' in Σ' of the form

$$R(\bar{u}_{\bar{t}(\bar{x})}) \wedge \mathbf{C}\big(\bar{u}'\big) \to \alpha(\bar{u}_{\bar{t}(\bar{x})}),$$

that has been constructed from σ, with $\alpha(\bar{u}_{\bar{t}(\bar{x})})$ the disjunction of the formulas in $\mathcal{C}_{R(\bar{t}(\bar{x}))}$. Notice that, by the construction of $\bar{u}_{\bar{t}(\bar{x})}$, we obtain that J satisfies $R(\bar{u}_{\bar{t}(\bar{x})})$ with the assignment $\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}]$ to $\bar{u}_{\bar{t}(\bar{x})}$. Let $\bar{a}'$ be the corresponding assignment to $\bar{u}'$ that derives from the assignment of $\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}]$ to $\bar{u}_{\bar{t}(\bar{x})}$. By the construction of σ' and since $\bar{a}$ is a tuple of constant values, it is straightforward to see that $\mathbf{C}(\bar{a}')$ holds. Then we have that $J \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J)}] \wedge \mathbf{C}(\bar{a}')$. Moreover, since $(J, I_2) \models (\bigwedge \Sigma')[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$, we obtain that $I_2 \models \alpha(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}])[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$. From this last fact we conclude that there exist a disjunct $\beta(\bar{u}_{\bar{t}(\bar{x})})$ of $\alpha(\bar{u}_{\bar{t}(\bar{x})})$ of the form $\exists \bar{y}\ (\psi(\bar{y}) \wedge \nu(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})) \wedge \omega(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y})))$, and a tuple $\bar{b}$ of constant values such that

$$I_2 \models \psi(\bar{b}) \wedge \big(\nu\big(\bar{t}(\bar{a})\big[\bar{f} \mapsto \bar{f}^{(I_1,J)}\big], \bar{s}(\bar{b})\big) \wedge \omega\big(\bar{t}(\bar{a})\big[\bar{f} \mapsto \bar{f}^{(I_1,J)}\big], \bar{s}(\bar{b})\big)\big)\big[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}\big].$$

By the construction of the formula $\alpha(\bar{u}_{\bar{t}(\bar{x})})$, we know that $\beta(\bar{u}_{\bar{t}(\bar{x})})$ belongs to the set $\mathcal{C}_{R(\bar{t}(\bar{x}))}$. Thus, there exists a formula $\psi(\bar{y}) \to R(\bar{s}(\bar{y}))$ in Σ such that $\bar{s}(\bar{y})$ subsumes $\bar{t}(\bar{x})$. Notice that $I_2 \models \psi(\bar{b})$, and then since $(I_2, J^\star) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$, we know that $J^\star \models R(\bar{s}(\bar{b}))[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$. We show next that $R(\bar{s}(\bar{b}))[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$ is equal to $R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$, and then we obtain that $J^\star \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$, which is exactly what we want to prove.

Let $\bar{u}_{\bar{t}(\bar{x})} = (u_1, \ldots, u_n)$, $\bar{t}(\bar{x}) = (t_1(\bar{x}), \ldots, t_n(\bar{x}))$, and $\bar{s}(\bar{y}) = (s_1(\bar{y}), \ldots, s_n(\bar{y}))$. We show now that, for every i such that $1 \leqslant i \leqslant n$, it holds that $t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}] = s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$. First, assume that $s_i(\bar{y})$ is a variable y_m from $\bar{y}$, and b_m the (constant) value that corresponds to y_m in the assignment of $\bar{b}$ to $\bar{y}$. Notice that since $s_i(\bar{b})$ is the constant value b_m, then $s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}] = s_i(\bar{b}) = b_m$. Now, since $\bar{s}(\bar{y})$ subsumes $\bar{t}(\bar{x})$ and $s_i(\bar{y})$ is a variable, it holds that $t_i(\bar{x})$ is either a variable x_r of $\bar{x}$ or a non-atomic term $f(x_{r_1}, \ldots, x_{r_\ell})$, with f an ℓ-ary function symbol in $\bar{f}$ and $(x_{r_1}, \ldots, x_{r_\ell})$ an ℓ-tuple of variables from $\bar{x}$. Assume first that $t_i(\bar{x})$ is the variable x_r and let a_r be the value that corresponds to x_r in the assignment of $\bar{a}$ to $\bar{x}$. Notice that formula $\nu(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$ contains the equality $u_i = y_m$. Then since $\nu(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}], \bar{s}(\bar{b}))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$ holds, we obtain that $a_r = b_m$. Notice that since a_r is a constant value the interpretation of the function symbols in $\bar{f}$ (or in $\bar{f}'$) does not affect it. In particular, we have that $a_r = t_i(\bar{a}) = t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$ and thus, since $a_r = b_m$ we obtain that $t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}] = s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$, which was to be shown. Assume now that $t_i(\bar{x})$ is a term $f(x_{r_1}, \ldots, x_{r_\ell})$, with f in $\bar{f}$ and $(x_{r_1}, \ldots, x_{r_\ell})$ a tuple of variables from $\bar{x}$. Given that $\nu(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$ contains then equality $u_i = y_m$, and since $\nu(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}], \bar{s}(\bar{b}))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$ holds, we have that $f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell}) = b_m$ (where $(a_{r_1}, \ldots, a_{r_\ell})$ is the assignment to variables $(x_{r_1}, \ldots, x_{r_\ell})$ that derives from the assignment of $\bar{a}$ to variables $\bar{x}$). Moreover, since $s_i(\bar{y})$ is a variable, we know that formula $\omega(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$ contains the inequalities $f_\star(u_i) \neq g_1(u_i)$, for every function symbol g in $\bar{f}$. Thus, since formula $\omega(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}], \bar{s}(\bar{b}))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$ holds, we obtain that

$$f_\star^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big) \neq g_1^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big), \tag{12}$$

for every function symbol g in $\bar{f}$. Notice that from (12) and the construction of functions $\bar{f}^{(I_1,J^*)}$, we obtain that $f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell}) = f^{(I_1,J^*)}(a_{r_1}, \ldots, a_{r_\ell})$. Thus, since $f^{(I_1,J^*)}(a_{r_1}, \ldots, a_{r_\ell}) = f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell}) = b_m = s_i(\bar{b})$, we obtain that $t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}] = s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$, which was to be shown.

Now suppose that $\bar{s}_i(\bar{y})$ is a non-atomic term $g(y_{m_1}, \ldots, y_{m_k})$, with g a k-ary function symbol in $\bar{f}$ and $(y_{m_1}, \ldots, y_{m_k})$ a k-tuple of variables from $\bar{y}$. Then since $\bar{s}(\bar{y})$ subsumes $\bar{t}(\bar{x})$, it holds that $t_i(\bar{x})$ is a non-atomic term $f(x_{r_1}, \ldots, x_{r_\ell})$, with f an ℓ-ary function symbol in $\bar{f}$ and $(x_{r_1}, \ldots, x_{r_\ell})$ an ℓ-tuple of variables from $\bar{x}$. Notice that formula $\nu(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$ contains, for every j such that $1 \leqslant j \leqslant k$, the equality $y_{m_j} = g_j(u_i)$ as a conjunction, with g_j a unary function symbol in $\bar{f}'$. We also know that formula $\nu(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}], \bar{s}(\bar{b}))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$ holds. Then for every j such that $1 \leqslant j \leqslant k$, we have that $b_{m_j} = (g_j(t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}]))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$. Thus, since we are assuming that $t_i(\bar{x}) = f(x_{r_1}, \ldots, x_{r_\ell})$, we know that the following equalities hold:

$$\begin{aligned} b_{m_1} &= g_1^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big), \\ &\vdots \\ b_{m_k} &= g_k^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big). \end{aligned} \tag{13}$$

Now, focus on the formula $\omega(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$. Since $\bar{s}_i(\bar{y}) = g(y_{m_1}, \ldots, y_{m_k})$, we know that $\omega(\bar{u}_{\bar{t}(\bar{x})}, \bar{s}(\bar{y}))$ contains the equality $f_\star(u_i) = g_1(u_i)$, and the inequalities $f_\star(u_i) \neq h_1(u_i)$, for every h in $\bar{f}$ different from g. Then since we know that formula $\omega(\bar{t}(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J)}], \bar{s}(\bar{b}))[\bar{f}' \mapsto \bar{f}'^{(J,I_2)}]$ holds, we obtain that

$$f_\star^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big) = g_1^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big), \tag{14}$$

and for every h in $\bar{f}$ different from g,

$$f_\star^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big) \neq h_1^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big).$$

Notice then that g is the unique function in $\bar{f}$ that satisfies (14). Then by the construction of $\bar{f}^{(I_1,J^\star)}$ we know that

$$f^{(I_1,J^\star)}(a_{r_1}, \ldots, a_{r_\ell}) = g^{(I_2,J^\star)}\big(g_1^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big), \ldots, g_k^{(J,I_2)}\big(f^{(I_1,J)}(a_{r_1}, \ldots, a_{r_\ell})\big)\big).$$

By replacing the equalities in (13) in this last expression we obtain that

$$f^{(I_1,J^\star)}(a_{r_1}, \ldots, a_{r_\ell}) = g^{(I_2,J^\star)}(b_{m_1}, \ldots, b_{m_k}).$$

Notice $s_i(\bar{b}) = g(b_{m_1}, \ldots, b_{m_k})$, and $t_i(\bar{a}) = f(a_{r_1}, \ldots, a_{r_\ell})$, thus we have that $t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}] = s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$, which was to be shown.

We have shown that, for every i such that $1 \leqslant i \leqslant n$, it holds that $t_i(\bar{a})[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}] = s_i(\bar{b})[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$. Thus we have that $R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$ is equal to $R(\bar{s}(\bar{b}))[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$. Then since $J^\star \models R(\bar{s}(\bar{b}))[\bar{f} \mapsto \bar{f}^{(I_2,J^\star)}]$ we know that $J^\star \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1,J^\star)}]$. What we have proved is that, for every formula $\varphi(\bar{x}) \to R(\bar{t}(\bar{x}))$ in Σ, if $I_1 \models \varphi(\bar{a})$, then

$J^\star \models R(\bar{t}(\bar{a}))[\bar{f} \mapsto \bar{f}^{(I_1, J^\star)}]$. Thus we have that $(I_1, J^\star) \models (\bigwedge \Sigma)[\bar{f} \mapsto \bar{f}^{(I_1, J^\star)}]$, and then $(I_1, J^\star) \models \exists \bar{f} \ \bigwedge \Sigma$. This concludes the proof of the theorem. □

We conclude this section with two important remarks about the algorithm presented above. Recall that a mapping $\mathcal{M}$ from a schema $\mathbf{R}_1$ to a schema $\mathbf{R}_2$ is total if $\text{dom}(\mathcal{M}) = \text{Inst}(\mathbf{R}_1)$, and it is closed-down on the left if whenever $(I, J) \in \mathcal{M}$ and $I' \subseteq I$, it holds that $(I', J) \in \mathcal{M}$. Given that plain SO-tgds are total and closed down on the left, we obtain from Proposition 4 the following corollary.

Corollary 20. *Let $\mathcal{M}$ be an st-mapping specified by a plain SO-tgd λ. If $\mathcal{M}$ has a Fagin-inverse (quasi-inverse), then algorithm* PolySOInv($\mathcal{M}$) *computes in polynomial time a Fagin-inverse (quasi-inverse) of $\mathcal{M}$.*

Moreover, given that every set of st-tgds can be transformed into an equivalent plain SO-tgd in linear time, our algorithm can be used to compute Fagin-inverses, quasi-inverses, and maximum recoveries for st-mappings specified by sets of st-tgds. This is the first polynomial-time algorithm capable of doing this.

6. Concluding remarks

We have studied the language of plain SO-tgds, its structural properties, and its properties regarding composition and inversion. We have argued that plain SO-tgds are a good alternative as a mapping language for data exchange and data integration, since plain SO-tgds retain the most important structural properties of st-tgds, and enjoy some interesting new properties regarding composition and inversion.

In fact, just as st-tgds, plain SO-tgds admit universal solutions, allow for conjunctive query rewriting, and are closed under target homomorphisms, which have been identified in [27] as three of the most fundamental structural properties for data exchange and data integration. Furthermore, we have shown that plain SO-tgds are a good alternative if one wants to apply composition and inversion operators. Most notably, we have proved that plain SO-tgds are closed under CQ-composition, the CQ-composition of any number of mappings given by st-tgds can be defined with a plain SO-tgd, any plain SO-tgd defines the CQ-composition of two mappings specified by st-tgds, and plain SO-tgds always have a maximum recovery. These results show that plain SO-tgds are the right language for capturing the CQ-composition of st-tgds and, more importantly, they show how an inverse operator can be applied to a sequence of CQ-compositions of mappings given by either plain SO-tgds, or st-tgds.

With the practical applicability of inversion in mind, we also presented a polynomial-time algorithm for computing the inverse of plain SO-tgds (and thus, also the inverse of st-tgds) under all of the most important notions of inversion that have been proposed in the classical data exchange setting. This is the first efficient algorithm capable of doing this.

Many questions remain open. For example, it would be interesting to see whether plain SO-tgds capture all mappings that admit universal solutions, allow for conjunctive query rewriting and are closed under target homomorphisms. In the search for a mapping language that is closed under both inversion and composition operators, we would like to study whether all the features introduced in our algorithm are really needed to specify the maximum recovery of mappings given by plain SO-tgds. Furthermore, we want to investigate the closure properties of maximum recoveries of mappings specified by plain SO-tgds. In particular, study if the inversion algorithm can be refined, or the inversion semantics be relaxed, in order to ensure that the inverse of plain SO-tgds can always be expressed as a plain SO-tgd.

Acknowledgments

M. Arenas was supported by Fondecyt grants #1090565 and #1110287, J. Pérez by Fondecyt grant #11110404 and VID grant U-Inicia 11/04 Universidad de Chile, and J. Reutter by EPSRC grant G049165 and FET-Open Project FoX grant 233599.

References

[1] M. Arenas, R. Fagin, A. Nash, Composition with target constraints, in: ICDT, 2010, pp. 129–142.
[2] M. Arenas, J. Pérez, J.L. Reutter, Data exchange beyond complete data, in: PODS, 2011, pp. 83–94.
[3] M. Arenas, J. Pérez, J.L. Reutter, C. Riveros, Composition and inversion of schema mappings, SIGMOD Rec. 38 (3) (2009) 17–28.
[4] M. Arenas, J. Pérez, J.L. Reutter, C. Riveros, Inverting schema mappings: bridging the gap between theory and practice, PVLDB 2 (1) (2009) 1018–1029.
[5] M. Arenas, J. Pérez, C. Riveros, The recovery of a schema mapping: bringing exchanged data back, ACM Trans. Database Syst. 34 (4) (2009).
[6] P. Bernstein, Applying model management to classical meta data problems, in: CIDR, 2003.
[7] P. Bernstein, S. Melnik, Model management 2.0: manipulating richer mappings, in: SIGMOD, 2007, pp. 1–12.
[8] A. Dawar, A restricted second order logic for finite structures, Inform. and Comput. 143 (2) (1998) 154–174.
[9] H.B. Enderton, A Mathematical Introduction to Logic, second edition, Academic Press, 2011.
[10] R. Fagin, Inverting schema mappings, ACM Trans. Database Syst. 32 (4) (2007).
[11] R. Fagin, P.G. Kolaitis, Local transformations and conjunctive-query equivalence, in: PODS, 2012, pp. 179–190.
[12] R. Fagin, P.G. Kolaitis, R.J. Miller, L. Popa, Data exchange: semantics and query answering, Theoret. Comput. Sci. 336 (1) (2005) 89–124.
[13] R. Fagin, P.G. Kolaitis, A. Nash, L. Popa, Towards a theory of schema-mapping optimization, in: PODS, 2008, pp. 33–42.
[14] R. Fagin, P.G. Kolaitis, L. Popa, W.-C. Tan, Composing schema mappings: second-order dependencies to the rescue, ACM Trans. Database Syst. 30 (4) (2005) 994–1055.

[15] R. Fagin, P.G. Kolaitis, L. Popa, W.C. Tan, Quasi-inverses of schema mappings, ACM Trans. Database Syst. 33 (2) (2008).
[16] R. Fagin, P.G. Kolaitis, L. Popa, W.C. Tan, Reverse data exchange: coping with nulls, ACM Trans. Database Syst. 36 (2) (2011) 11.
[17] R. Fagin, A. Nash, The structure of inverses in schema mappings, J. ACM 57 (6) (2010) 31.
[18] I. Feinerer, R. Pichler, E. Sallinger, V. Savenkov, On the undecidability of the equivalence of second-order tuple generating dependencies, in: AMW, 2011.
[19] A. Fuxman, M.A. Hernández, C.T.H. Ho, R.J. Miller, P. Papotti, L. Popa, Nested mappings: schema mapping reloaded, in: VLDB, 2006, pp. 67–78.
[20] M. Lenzerini, Data integration: a theoretical perspective, in: PODS, 2002, pp. 233–246.
[21] L. Libkin, Elements of Finite Model Theory, 1st edition, Springer-Verlag, 2004.
[22] J. Madhavan, A.Y. Halevy, Composing mappings among data sources, in: VLDB, 2003, pp. 572–583.
[23] S. Melnik, Generic Model Management: Concepts and Algorithms, Lecture Notes in Comput. Sci., vol. 2967, Springer, 2004.
[24] S. Melnik, P.A. Bernstein, A.Y. Halevy, E. Rahm, Supporting executable mappings in model management, in: SIGMOD, 2005, pp. 167–178.
[25] A. Nash, P.A. Bernstein, S. Melnik, Composition of mappings given by embedded dependencies, in: PODS, 2005, pp. 172–183.
[26] B. ten Cate, P.G. Kolaitis, Structural characterizations of schema-mapping languages, in: ICDT, 2009, pp. 63–72.
[27] B. ten Cate, P.G. Kolaitis, Structural characterizations of schema-mapping languages, Commun. ACM 53 (1) (2010) 101–110.