Inverting Schema Mappings:
Bridging the Gap between Theory and Practice

Marcelo Arenas
PUC Chile
marenas@ing.puc.cl

Jorge Pérez
PUC Chile
jperez@ing.puc.cl

Juan Reutter
PUC Chile
jlreutte@puc.cl

Cristian Riveros
R&M Tech
criveros@rmt.cl

ABSTRACT

The inversion of schema mappings has been identified as one of the fundamental operators for the development of a general framework for metadata management. In fact, during the last years three alternative notions of inversion for schema mappings have been proposed (Fagin-inverse [10], quasi-inverse [14] and maximum recovery [2]). However, the procedures that have been developed for computing these operators have some features that limit their practical applicability. First, these algorithms work in exponential time and produce inverse mappings of exponential size. Second, these algorithms express inverses in some mappings languages which include features that are difficult to use in practice. A typical example is the use of disjunction in the conclusion of the mapping rules, which makes the process of exchanging data much more complicated.

In this paper, we propose solutions for the two problems mentioned above. First, we provide a polynomial time algorithm that computes the three inverse operators mentioned above given a mapping specified by a set of tuple-generating dependencies (tgds). This algorithm uses an output mapping language that can express these three operators in a compact way and, in fact, can compute inverses for a much larger class of mappings. Unfortunately, it has already been proved that this type of mapping languages has to include some features that are difficult to use in practice and, hence, this is also the case for our output mapping language. Thus, as our second contribution, we propose a new and natural notion of inversion that overcomes this limitation. In particular, every mapping specified by a set of tgds admits an inverse under this new notion that can be expressed in a mapping language that slightly extends tgds, and that has the same good properties for data exchange as tgds. Finally, as our last contribution, we provide an algorithm for computing such inverses.

1. INTRODUCTION

A schema mapping is a specification that describes how

data from a source schema is to be mapped to a target schema. Schema mappings have proved to be essential for several data-interoperability tasks such as data exchange [11], data integration [21] and peer data management [6, 17]. The research on this area has mainly focused on performing these tasks, and has produced several applications that work with declarative specifications of schema mappings [15, 17]. However, as Bernstein pointed out in [3], many information-system problems involve not only the design and integration of complex application artifacts, but also their subsequent manipulation. Driven by this consideration, Bernstein proposed in [3] a general framework for managing schema mappings. In this framework, schema mappings are usually specified in a logical language, and high-level algebraic operators like match, merge and compose are used to manipulate them [3, 23, 24, 4].

One of the operators in Bernstein's framework is the *inverse* of a schema mapping, that has recently received considerable attention [10, 14, 4, 2]. Consider a mapping $\mathcal{M}$ from a schema $\mathbf{A}$ to a schema $\mathbf{B}$. Intuitively, an *inverse* of $\mathcal{M}$ is a new mapping that describes the *reverse* relationship from $\mathbf{B}$ to $\mathbf{A}$, and is semantically consistent with the relationship previously established by $\mathcal{M}$.

In practical scenarios, the inverse of a mapping can have several applications. In a data exchange context [11], if a mapping $\mathcal{M}$ is used to exchange data from a source to a target schema, an inverse of $\mathcal{M}$ can be used to exchange the data back to the source, thus *reverting* the application of $\mathcal{M}$. As a second application, consider a peer-data management system (PDMS) [6, 17]. In a PDMS, a peer can act as a data source, a mediator, or both, and the system relates peers by establishing mappings between the peers schemas. Mappings between peers are usually *directional*, and are used to reformulate queries. For example, if there is a mapping $\mathcal{M}$ from peer P_1 to peer P_2 and a query over P_2, a PDMS can use $\mathcal{M}$ to reformulate the query by using P_1 as a source. Hence, an inverse of $\mathcal{M}$ would allow the PDMS to reformulate a query over P_1 in terms of P_2, thus considering this time P_2 as a source. Another application is schema evolution, where the inverse together with the composition play a crucial role [4]. Consider a mapping $\mathcal{M}$ between schemas $\mathbf{A}$ and $\mathbf{B}$, and assume that schema $\mathbf{A}$ evolves into a schema $\mathbf{A}'$. This evolution can be expressed as a mapping $\mathcal{M}'$ between $\mathbf{A}$ and $\mathbf{A}'$. Thus, the relationship between the new schema $\mathbf{A}'$ and schema $\mathbf{B}$ can be obtained by inverting mapping $\mathcal{M}'$ and then composing the result with mapping $\mathcal{M}$.

All the previous work on inverting schema mappings have

been motivated by foundational issues [10, 14, 2], being one of the most delicate the definition of a good semantics for inversion. In fact, up to now little attention has been paid to the study of practical issues regarding inverting schema mappings.

In the data exchange scenario, the standard procedure used to exchange data with a mapping is based on the *chase* procedure [11]. More precisely, given a mapping $\mathcal{M}$ and a source database I, a *canonical* translation of I according to $\mathcal{M}$ is computed by *chasing* I with the set of dependencies defining $\mathcal{M}$ [11]. Thus, when computing an inverse of $\mathcal{M}$, it would be desirable from a practical point of view to obtain a mapping $\mathcal{M}'$ where the chase procedure can be used to exchange data.

Closely related with the above issue, there is a *representation* issue. For example, in a PDMS schema mappings are usually expressed in terms of *tuple-generating dependencies* (tgds), or, equivalently, *global-and-local-as-view* constraints (GLAV). Hence, it would be desirable that the inverse of a schema mapping could also be expressed as a set of tgds or GLAV constraints, thus maintaining mappings in a manageable setting.

Orthogonal to the previous issues, the efficiency of the algorithms used for computing inverses is also crucial. If one wants to use the inverse operator in practice, research on the feasibility of implementing these algorithms must be carried out.

In this paper, we study the practical issues mentioned above. But before showing into detail our contributions, we take a closer look at the inverse notions proposed in the literature, and the features of these notions that limit their practical applicability.

The first proposal of an inverse operator, that we call here Fagin-inverse, was proposed by Fagin in [10]. Arguably, this notion is too strong for practical purposes, as most of the schema mappings used in practice do not admit a Fagin-inverse. As shown by Fagin et al. in their subsequent work [14], the fact that some mappings do not admit a Fagin-inverse should not lead to the conclusion that for these mappings no data can be recovered. In fact, Fagin et al. responded to the existence issue by proposing in [14] the notion of quasi-inverse of a schema mapping. Under this new notion, numerous schema mappings that do not have Fagin-inverses possess natural and useful quasi-inverses [14]. Nevertheless, there are still simple mappings specified by tgds that have no quasi-inverse. And not only that, there exists an additional issue about the type of mappings needed to specify quasi-inverses. In [14], the authors provide an algorithm to compute quasi-inverses of mappings given by tgds whenever such an inverse exists. The output of this algorithm is a set of dependencies similar to tgds, but that include disjunctions in the conclusions of the dependencies. Moreover, they proved that disjunctions are unavoidable in order to represent quasi-inverses. Notice that this type of mappings are difficult to use in the data exchange context. In particular, it is not clear whether the standard chase procedure could be used to produce a single canonical target database in this case, thus making the process of exchanging data and answering queries much more complicated. [1]

---

[1] Some extensions of the chase procedure to deal with disjunctive dependencies have been proposed in the literature [7]. Nevertheless, these extensions have not been developed for the data exchange tasks mentioned here.

In [2], Arenas et al. proposed another notion of inverse for schema mappings, which was called *maximum recovery*. One of the main properties of this new notion is that every mapping specified by a set of tgds has a maximum recovery. Since maximum recoveries always exist for the most common mappings, from a practical point of view this notion is an improvement over the notions of Fagin-inverse and quasi-inverse. Nevertheless, as shown in [2], the maximum recovery of a mapping specified by a set of tgds cannot always be represented as a set of tgds. In fact, the authors proved in [2] that disjunctions are unavoidable to represent maximum recoveries and, hence, one cannot hope to always obtain a representation of a maximum recovery with good properties for data exchange.

Regarding the efficiency issue mentioned above, the proposed algorithms for computing the notions of Fagin-inverse, quasi-inverse and maximum recovery produce inverses of exponential size for the case of mappings specified by tgds. In fact, it was proved in the extended version of [2] that for the mapping languages considered in [10, 14, 2], there exists a mapping given by a set of tgds such that every maximum recovery of this mapping is of exponential size.

1.1 Contributions

In this paper, we revisit the problem of inverting schema mappings, motivated by some of the practical limitations of the previous notions of inversion. In particular, we propose solutions for all the limitations mentioned above. More precisely, the following are our main contributions.

A query language-based notion of inverse. One of our conceptual contributions is the proposal of a new and natural notion of inversion for schema mappings. Assume that a mapping $\mathcal{M}$ is used to exchange data from source to target. Then we measure the amount of exchanged data that can be recovered back in the source according to a class of queries. Intuitively, our new notion of inverse focuses on recovering the maximum amount of information with respect to a class $\mathcal{C}$ of queries, which gives rise to the notion of $\mathcal{C}$-maximum recovery.

We study the connection of the notion of $\mathcal{C}$-maximum recovery with the previously proposed notions of inverse. In particular, we show that $\mathcal{C}$-maximum recoveries capture these notions for different choices of the class $\mathcal{C}$ of queries. And more importantly, by focusing on the class of conjunctive queries, we show that the notion of $\mathcal{C}$-maximum recovery can be used to overcome some of the practical limitations mentioned above.

CQ-maximum recovery. Let CQ be the class of conjunctive queries. We show in this paper that the notion of CQ-maximum recovery has two desirable properties, namely that every mapping specified by a set of tgds admits an inverse under this new notion, and that this mapping can be expressed in a language that has the same good properties for data exchange as tgds. In particular, we provide an algorithm that given a mapping $\mathcal{M}$ specified by a set of tgds, produces a CQ-maximum recovery of $\mathcal{M}$ specified by a set of tgds extended with two features: inequalities and a built-in predicate $\mathbf{C}(\cdot)$ to differentiate constants from null values. These two features are included only in the premises of the dependencies, thus obtaining a language as good as tgds for data exchange purposes. We also prove that inequalities and predicate $\mathbf{C}(\cdot)$ are both essential to express CQ-maximum recoveries of mappings given by tgds.

A polynomial-time algorithm for computing inverses. As we mentioned before, an orthogonal issue regarding the notions of inverse proposed so far is the efficiency of the algorithms developed to compute them [10, 14, 2]. In this paper, we present the first polynomial time algorithm for computing inverses of schema mappings. In fact, our algorithm can be used to compute Fagin-inverses, quasi-inverses, and maximum recoveries of mappings given by tgds. And moreover, our algorithm works not only for mappings specified by tgds, but also for mappings specified in a much richer language that contains most of the schema mapping languages used in practice (for example, nested mappings [15]).

As we mentioned before, it has already been proved that a language capable of expressing Fagin-inverses, quasi-inverses, and maximum recoveries has to include some features that are difficult to use in practice. Thus, our gain in time complexity comes with the price of a stronger and less manageable mapping language for expressing inverses. In fact, our algorithm uses an output language that is similar to the language of second-order tgds proposed in [13], but with some extra features.

The organization of the paper corresponds with the previous list of contributions. We begin by defining the terminology used trough the paper.

2. BASIC NOTATION

Our study assumes that data is represented in the relational model. A *relational schema*, or just *schema*, is a finite set $\{R_1, \ldots, R_n\}$ of relation symbols. As usual in the data exchange literature, we consider database instances with two types of values: *constants* and *nulls*. If we refer to a schema **S** as a *source* schema, then we restrict all instances of **S** to consist only of constant values. On the other hand, we allow null values in the instances of any *target* schema **T**.

Schema mappings and solutions. Schema mappings are used to define a semantic relationship between two schemas. In this paper, 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 $\mathbf{R}_2$. Further, we say that J is a *solution for I under $\mathcal{M}$* if (I, J) is in $\mathcal{M}$.

As usual, we use the class of *tuple-generating dependencies* (tgds) to specify schema mappings [11]. A set Σ of tuple generating dependencies from a schema $\mathbf{R}_1$ to a schema $\mathbf{R}_2$ is a set of formulas of the form:

$$\varphi(\bar{x}) \rightarrow \exists \bar{y} \; \psi(\bar{x}, \bar{y})$$

where $\varphi(\bar{x})$ is a conjunction of relational atoms over $\mathbf{R}_1$, and $\psi(\bar{x}, \bar{y})$ is a conjunction of relational atoms over $\mathbf{R}_2$. For example, the following is a tuple-generating dependency:

$$R(x, y) \wedge S(y, z) \quad \rightarrow \quad \exists u \, T(x, z, u).$$

The semantics of tgds is defined as follows. We say that a pair (I, J) of instances satisfies a set Σ of tgds if it holds that, for every rule in this set of the form $\varphi(\bar{x}) \rightarrow \exists \bar{y} \, \psi(\bar{x}, \bar{y})$ and every tuple of elements $\bar{a}$ from I, if I satisfies $\varphi(\bar{a})$, then there exists a tuple $\bar{b}$ of elements from J such that J satisfies $\psi(\bar{a}, \bar{b})$.

Composition of mappings. The notion of composition has shown to be of fundamental importance in defining the notion of inverse of a schema mapping [10, 14, 2]. Let $\mathcal{M}_{12}$ be a schema mapping from $\mathbf{R}_1$ to $\mathbf{R}_2$, and $\mathcal{M}_{23}$ a schema mapping from $\mathbf{R}_2$ to $\mathbf{R}_3$. Then the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$, denoted by $\mathcal{M}_{12} \circ \mathcal{M}_{23}$, is defined as the standard composition of binary relations, that is, as the set of all pairs of instances (I, J) such that I is an instance of $\mathbf{R}_1$, J is an instance of $\mathbf{R}_3$, and there exists an instance K of $\mathbf{R}_2$ such that (I, K) belongs to $\mathcal{M}_{12}$, and (K, J) belongs to $\mathcal{M}_{23}$.

Query Answering. In this paper, we use CQ to denote the class of conjunctive queries and UCQ to denote the class of unions of conjunctive queries. Given a query Q and a database instance I, we denote by $Q(I)$ the evaluation of Q over I. Moreover, we use predicate $\mathbf{C}(\cdot)$ to differentiate between constants and nulls, that is, $\mathbf{C}(a)$ holds if and only if a is a constant value.

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 answers of Q for I under $\mathcal{M}$*, denoted by $\underline{\text{certain}}_{\mathcal{M}}(Q, I)$, is the set of tuples that belong to the evaluation of Q over every possible solution for I under $\mathcal{M}$, that is, $\bigcap \{Q(J) \mid J \text{ is a solution for } I \text{ under } \mathcal{M}\}$.

3. QUERY LANGUAGE-BASED INVERSES OF SCHEMA MAPPINGS

Intuitively, an inverse of a schema mapping $\mathcal{M}$ is a *reverse* mapping that *undoes* the application of $\mathcal{M}$. Any natural notion of inverse should capture the intuition that, if $\mathcal{M}$ describes how to exchange data from a source to a target schema, the inverse of $\mathcal{M}$ must describe how to *recover* the initial data back in the source (or, at least, part of it). Moreover, one should impose a soundness requirement; we would like to recover only *sound information*, that is, information that was already present before the exchange.

A natural question at this point is how one can formally define the idea of recovering sound information. In this paper, we give a formal definition of recovering sound information with respect to a query language, and use this notion to define our query-language based notion of inverse of a schema mapping.

Before giving any formal definition, let us present the intuition of our notions with one example.

EXAMPLE 3.1. Let **S** be a source schema composed by binary relations $R(\cdot, \cdot)$ and $S(\cdot, \cdot)$, **T** a target schema composed by a binary relation $T(\cdot, \cdot)$, and assume that schemas **S** and **T** are related by a mapping $\mathcal{M}$ given by the tgd:

$$R(x, y) \wedge S(y, z) \quad \rightarrow \quad T(x, z). \tag{1}$$

Thus, target relation T stores the join of source relations R and S. Consider now the *reverse* mapping $\mathcal{M}'$ relating **T** and **S** through the dependency:

$$T(x, y) \quad \rightarrow \quad \exists u \, R(x, u). \tag{2}$$

This dependency states that whenever an element is in the first component of relation T in the target database, it must also be in the first component of relation R in the source database. Thus, given the definition of mapping $\mathcal{M}$, one can intuitively conclude that mapping $\mathcal{M}'$ recovers sound information with respect to $\mathcal{M}$. The previous intuition can be formalized by considering the composition of $\mathcal{M}$ with

$\mathcal{M}'$, which represents the idea of using $\mathcal{M}'$ to bring back to the source the information that was exchanged by using mapping $\mathcal{M}$. It is important to notice that $\mathcal{M} \circ \mathcal{M}'$ is a *round-trip* mapping from **S** to **S** and, therefore, one can use queries over **S** to measure the amount of recovered information. In particular, one can claim in this example that $\mathcal{M}'$ recovers sound information with respect to $\mathcal{M}$ since for every source instance I and query Q over **S**, if a tuple $\bar{t}$ belongs to the certain answers of Q for I under $\mathcal{M} \circ \mathcal{M}'$, then $\bar{t}$ also belongs to the evaluation of Q over I.

Let us give an example of the previous discussion with a concrete scenario. Assume that I is a source database $\{R(1,2),\ R(3,4),\ S(2,5)\}$, and Q is a source conjunctive query $\exists y\, R(x,y)$. If we directly evaluate Q over I, we obtain the set of answers $Q(I) = \{1,3\}$. On the other hand, if we evaluate Q over the set of instances obtained by exchanging data from I through $\mathcal{M} \circ \mathcal{M}'$, then we obtain the set of answers $\{1\}$, which is contained in $Q(I)$. Thus, when computing the certain answers for I we obtain a subset of the direct evaluation of the query over I, that is

$$\underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) \subseteq Q(I). \qquad (3)$$

$\square$

In general, we say that a mapping $\mathcal{M}'$ is a Q-recovery of a mapping $\mathcal{M}$ whenever equation (3) holds for every source database I. For example, for the mappings $\mathcal{M}$ and $\mathcal{M}'$ defined by dependencies (1) and (2), respectively, we have that $\mathcal{M}'$ is a Q-recovery of $\mathcal{M}$ for query $Q(x) = \exists y\, R(x,y)$. But moreover, it can be shown that $\mathcal{M}'$ is a Q-recovery of $\mathcal{M}$ for every query Q. Indeed, this is the reason why we claim that $\mathcal{M}'$ recovers sound information with respect to $\mathcal{M}$ in this case. But in some cases one may be interested in retrieving sound information not for every possible query but for a class of queries of interest (for instance, for the class of conjunctive queries). This gives rise to the following notion of $\mathcal{C}$-recovery, where $\mathcal{C}$ is a class of queries.

DEFINITION 3.2. *Let $\mathcal{C}$ be a class of queries, $\mathcal{M}$ a mapping from schema **S** to schema **T**, and $\mathcal{M}'$ a mapping from **T** to **S**. Then $\mathcal{M}'$ is a $\mathcal{C}$-recovery of $\mathcal{M}$ if for every query $Q \in \mathcal{C}$ over **S** and every instance I of **S**, it holds that:*

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

Being a $\mathcal{C}$-recovery is a sound but mild requirement. Thus, it is natural to ask whether one can compare mappings according to their ability to recover sound information, and then whether there is a natural way to define a notion of *best* possible recovery according to a given query language. It turns out that there is simple and natural way to do this, as we show in the following example.

EXAMPLE 3.3. Let $\mathcal{M}$ and $\mathcal{M}'$ be the mappings given by dependencies (1) and (2), respectively, and $\mathcal{M}''$ a mapping specified by dependency:

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

As we mentioned above, $\mathcal{M}'$ recovers sound information with respect to $\mathcal{M}$ as it is a Q-recovery of $\mathcal{M}$ for every query Q. Furthermore, it can be shown that this property also holds for mapping $\mathcal{M}''$. But not only that, it can also be shown that for every query Q and source instance I, it holds that:

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

For instance, if $I = \{R(1,2), R(3,4), S(2,5)\}$ and $Q(x,y)$ is conjunctive query $\exists z\, (R(x,z) \wedge S(z,y))$, then we have that $Q(I) = \{(1,5)\}$ and $\underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}''}(Q, I) = \{(1,5)\}$, while $\underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) = \emptyset$.

Hence, every tuple that is retrieved by posing query Q over the space of solutions for I under $\mathcal{M} \circ \mathcal{M}'$ is also retrieved by posing this query over the space of solutions for I under $\mathcal{M} \circ \mathcal{M}''$. Thus, we can claim that $\mathcal{M}''$ is *better* than $\mathcal{M}'$ recovering sound information with respect to $\mathcal{M}$. $\square$

The above discussion gives rise to a simple way to compare two Q-recoveries $\mathcal{M}'$ and $\mathcal{M}''$ of a mapping $\mathcal{M}$; a mapping $\mathcal{M}''$ recovers as much information as $\mathcal{M}'$ does for $\mathcal{M}$ under Q if for every source instance I, it holds that $\underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}'}(Q, I) \subseteq \underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}''}(Q, I)$. With this way of comparing inverse mappings, it is simple to define a notion of *best* possible recovery according to a query language.

DEFINITION 3.4. *Let $\mathcal{C}$ be a class of queries and $\mathcal{M}_1$ a $\mathcal{C}$-recovery of a mapping $\mathcal{M}$. Then $\mathcal{M}_1$ is a $\mathcal{C}$-maximum recovery of $\mathcal{M}$ if for every $\mathcal{C}$-recovery $\mathcal{M}_2$ of $\mathcal{M}$, it holds that:*

$$\underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}_2}(Q, I) \quad \subseteq \quad \underline{\text{certain}}_{\mathcal{M} \circ \mathcal{M}_1}(Q, I)$$

for every query Q in $\mathcal{C}$ and source database I.

That is, $\mathcal{M}_1$ is a $\mathcal{C}$-maximum recovery of a mapping $\mathcal{M}$, if by exchanging data from I through $\mathcal{M} \circ \mathcal{M}_1$, one can retrieve by using queries from $\mathcal{C}$ as much information as by exchanging data from I through $\mathcal{M} \circ \mathcal{M}_2$, for any other $\mathcal{C}$-recovery $\mathcal{M}_2$ of $\mathcal{M}$. For instance, for the mappings $\mathcal{M}$ and $\mathcal{M}''$ mentioned in Example 3.3, it holds that $\mathcal{M}''$ is an ALL-maximum recovery of $\mathcal{M}$, where ALL is the class of all queries.

It is important to notice that the choice of a query language makes a difference in the definition of $\mathcal{C}$-maximum recovery. For example, assume that $\mathcal{M}$ is mapping given by the following dependencies:

$$\begin{aligned} A(x) &\rightarrow D(x), \\ B(x) &\rightarrow D(x) \wedge E(x). \end{aligned}$$

In this case, it can be shown that the mapping $\mathcal{M}'$ given by the following dependencies is an ALL-maximum recovery of $\mathcal{M}$:

$$\begin{aligned} D(x) &\rightarrow A(x) \vee B(x), \\ E(x) &\rightarrow B(x). \end{aligned}$$

Unfortunately, the first dependency includes a disjunction on the conclusion, which makes the processes of exchanging data and computing certain answers much more complicated. On the other hand, if one is interested in retrieving information by using only conjunctive queries, then mapping $\mathcal{M}'$ as well as a mapping $\mathcal{M}''$ given by the following tgd are both CQ-maximum recoveries of $\mathcal{M}$:

$$E(x) \quad \rightarrow \quad B(x).$$

Thus, by focusing on certain query languages in the definition of $\mathcal{C}$-maximum recovery, one can avoid employing in mapping languages features that are difficult to use in practice. This observation motivates the search for a class of queries that gives rise to an inverse notion meeting the two requirements mentioned in the introduction, namely that every mapping specified by a set of tgds admits an inverse

under this new notion, and that this mapping can be expressed in a language that has the same good properties for data exchange as tgds. A natural starting point in this search is the class of conjunctive queries, as this class is widely used in practice and, in particular, has been extensively studied in the context of data exchange [11]. In fact, from the results in [2], it is straightforward to prove that every mapping specified by a set of tgds has a CQ-maximum recovery. Hence, it remains to show that CQ-maximum recoveries are expressible in a mapping language with good properties for data exchange. This is done in Section 4. But before going into the details of this result, we show in the next section that the general notion of $\mathcal{C}$-maximum recovery is of independent interest, as it can be used to characterize the previous notions of inverse proposed in the literature [10, 14, 2].

3.1 Comparison with previous notions

In the last years, three different notions of inverse have been proposed for schema mappings: Fagin-inverse [10], quasi-inverse [14] and maximum recovery [2]. The latter notion was motivated by the idea of defining an optimal way of retrieving sound information. In this sense, the notion of $\mathcal{C}$-maximum recovery is inspired by the concept of maximum recovery, as a $\mathcal{C}$-maximum recovery tries to find an optimal way to retrieve sound information according to the query language $\mathcal{C}$. In fact, it can be easily proved that if $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$, then $\mathcal{M}'$ is an ALL-maximum recovery of $\mathcal{M}$, where ALL is the class of all queries. The other two notions of inverse mentioned above are defined in a rather different way, and at a first glance they do not seem to be related with the query language approach proposed in this paper. However, they can be characterized in terms of the notion of $\mathcal{C}$-maximum recovery for some specific choices of the query language $\mathcal{C}$, as we show next. This result is of independent interest as it shows that the concept of $\mathcal{C}$-maximum recovery provides a unified framework for the notions of inversion proposed in the literature.

We start by considering the notion of Fagin-inverse proposed in [10]. Roughly speaking, Fagin's definition is based on the idea that a mapping composed with its inverse should be equal to the identity schema mapping. To define this notion, Fagin first defines an *identity mapping* $\overline{\mathrm{Id}}$ as $\{(I_1, I_2) \mid I_1, I_2 \text{ are source instances and } I_1 \subseteq I_2\}$, which is an appropriate identity for the mappings specified by tgds [10]. Then a mapping $\mathcal{M}'$ is said to be a *Fagin-inverse* of a mapping $\mathcal{M}$ if $\mathcal{M} \circ \mathcal{M}' = \overline{\mathrm{Id}}$. The following theorem states that whenever a Fagin-inverse exists, it coincides with the notion of $\mathrm{UCQ}^{\neq}$-maximum recovery, where $\mathrm{UCQ}^{\neq}$ is the class of unions of conjunctive queries with inequalities.

THEOREM 3.5. *Let $\mathcal{M}$ be a mapping specified by a set of tgds, and assume that $\mathcal{M}$ has a Fagin-inverse. Then the following statements are equivalent:*

- *$\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$.*
- *$\mathcal{M}'$ is a $\mathrm{UCQ}^{\neq}$-maximum recovery of $\mathcal{M}$.*

We continue by considering the notion of quasi-inverse [14]. The idea behind quasi-inverses is to relax the notion of Fagin-inverse by not differentiating between instances that are *data-exchange equivalent*. Two instances I_1, I_2, are data-exchange equivalent w.r.t. a mapping $\mathcal{M}$, denoted by $I_1 \sim_{\mathcal{M}} I_2$, if the space of solutions of I_1 under $\mathcal{M}$ coincides with the space of solutions of I_2 under $\mathcal{M}$ [14]. Given a mapping $\mathcal{M}_1$ from $\mathbf{S}$ to $\mathbf{S}$, mapping $\mathcal{M}_1[\sim_{\mathcal{M}}, \sim_{\mathcal{M}}]$ is defined as $\{(I_1, I_2) \mid \exists (I_1', I_2') : I_1 \sim_{\mathcal{M}} I_1', I_2 \sim_{\mathcal{M}} I_2' \text{ and } (I_1', I_2') \in \mathcal{M}_1\}$. Then $\mathcal{M}'$ is a *quasi-inverse* of $\mathcal{M}$ if $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}, \sim_{\mathcal{M}}] = \overline{\mathrm{Id}}[\sim_{\mathcal{M}}, \sim_{\mathcal{M}}]$.

THEOREM 3.6. *Let $\mathcal{M}$ be a mapping specified by a set of tgds, and assume that $\mathcal{M}$ has a quasi-inverse. Then there exists a subclass $\mathcal{C}$ of $\mathrm{UCQ}^{\neq}$ such that the following statements are equivalent:*

- *$\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$.*
- *$\mathcal{M}'$ is a $\mathcal{C}$-maximum recovery of $\mathcal{M}$.*

The class of queries $\mathcal{C}$ in the above theorem depends on the mapping $\mathcal{M}$, and can be effectively obtained from $\mathcal{M}$.

4. COMPUTING CQ-MAXIMUM RECOVERIES

In this section, we present an algorithm for computing CQ-maximum recoveries. Given a mapping specified by a set of tgds, our algorithm generates a CQ-maximum recovery specified by a set of tgds extended with inequalities and the predicate $\mathbf{C}(\cdot)$ in the premises of dependencies (recall that $\mathbf{C}(a)$ holds if a is a constant value). Although our algorithm does not produce a set of standard tgds, the output of our algorithm is a mapping that has the same good properties for data exchange as tgds. In particular, the set of dependencies is chaseable, that is, the standard chase procedure can be used to obtain a single canonical database. Furthermore, we show in Section 4.2 that inequalities and the predicate $\mathbf{C}(\cdot)$ are unavoidable in order to represent CQ-maximum recoveries of mappings given by tgds.

We begin this section by describing some of the tools needed in our algorithm. The first of such tools is *query rewriting*. Consider a mapping $\mathcal{M}$ between schemas $\mathbf{S}$ and $\mathbf{T}$ and a query Q over schema $\mathbf{T}$. We say that a query Q' over $\mathbf{S}$ is a *rewriting of Q over the source* if for every source database I, the set $Q'(I)$ is exactly the set of certain answers of Q over I with respect to $\mathcal{M}$, that is,

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

In our algorithm, we need to compute rewritings for conjunctive queries. It is known that given a mapping $\mathcal{M}$ specified by a set of tgds and a conjunctive query Q over the target schema, a rewriting of Q over the source always exists [1, 5]. Moreover, it can be shown that if Q is a conjunctive query, then a rewriting of Q over the source can always be expressed as a union of conjunctive queries with the equality predicates ($\mathrm{UCQ}^{=}$). As an example, consider a mapping given by the following tgds:

$$\begin{aligned} A(x, y) &\rightarrow P(x, y), \\ B(x) &\rightarrow P(x, x), \end{aligned}$$

and let Q be the target query $P(x, y)$. Then a rewriting of Q over the source is given by $A(x, y) \vee (B(x) \wedge x = y)$, which is a query in $\mathrm{UCQ}^{=}$. Notice that in this rewriting, we do need disjunction and the equality $x = y$.

The following observations are useful for our algorithm. Let Q be a query in $\mathrm{UCQ}^{=}$. Every equality of the form $x = y$ that occurs in Q is *safe* in the sense that x or y must occur in a relational atom in Q. Also, by using the appropriate variable substitutions, one can eliminate all equalities in

Q between two existentially quantified variables, or between a free variable and an existentially quantified variable. For example, the query $\exists u\,(P(x,y,u) \wedge y = u)$ is equivalent to $P(x,y,y)$, and the query $\exists u \exists v\,(P(x,u,v) \wedge u = v)$ is equivalent to $\exists u\,P(x,u,u)$. On the other hand, equalities between free variables cannot be simply eliminated, as in the above query $A(x,y) \vee (B(x) \wedge x = y)$. Thus, from now on, we assume that if an equality $x = y$ occurs in a query Q of the class $\mathrm{UCQ}^=$, then both x and y are free variables in Q.

A second tool that we use in our algorithm is the notion of *conjunctive-query equivalence* of mappings. Two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ are said to be equivalent with respect to conjunctive queries, denoted by $\mathcal{M}_1 \equiv_{\mathrm{CQ}} \mathcal{M}_2$, if for every conjunctive query Q, the set of certain answers of Q under $\mathcal{M}_1$ coincides with the set of certain answers of Q under $\mathcal{M}_2$. More precisely, we have that $\mathcal{M}_1 \equiv_{\mathrm{CQ}} \mathcal{M}_2$ if for every conjunctive query Q over the target and every source database I, it holds that $\underline{\mathrm{certain}}_{\mathcal{M}_1}(Q,I) = \underline{\mathrm{certain}}_{\mathcal{M}_2}(Q,I)$. The notion of conjunctive-query equivalence was introduced by Madhavan and Halevy [22] when studying the composition of schema mappings, and has also been used for schema mapping optimization in [12]. The following lemma relates the notions of CQ-maximum recovery and conjunctive-query equivalence, and is used in the formulation of our algorithm.

LEMMA 4.1. *Let $\mathcal{M}'$ be a* CQ-*maximum recovery of $\mathcal{M}$, and assume that $\mathcal{M}' \equiv_{\mathrm{CQ}} \mathcal{M}''$. Then $\mathcal{M}''$ is also a* CQ-*maximum recovery of $\mathcal{M}$.*

For the sake of readability, in this section we omit the existentially quantified variables in the consequents of tgds. Thus, we use notation $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$ for a tgd, where $\varphi(\bar{x})$ is a conjunction of relational atoms over $\mathbf{S}$ and $\psi(\bar{x})$ is a conjunctive query over $\mathbf{T}$ (that is, $\psi(\bar{x}) = \exists \bar{y}\,\alpha(\bar{x},\bar{y})$).

4.1 A CQ-maximum recovery algorithm

The basic strategy of our algorithm is as follows. Given a mapping $\mathcal{M}$ specified by a set of tgds, we start by computing a maximum recovery $\mathcal{M}_1$ of $\mathcal{M}$ by using the algorithm presented in [2]. Since the output mapping $\mathcal{M}_1$ is a maximum recovery of $\mathcal{M}$, we know by the results in Section 3 that $\mathcal{M}_1$ is also a CQ-maximum recovery of $\mathcal{M}$. However, the mapping $\mathcal{M}_1$ may have disjunctions and equalities in the conclusions of the dependencies specifying it. Thus, our algorithm performs a series of transformations on $\mathcal{M}_1$ that eliminate these disjunctions and equalities, while preserving conjunctive-query equivalence. To be more precise, we first use a procedure that eliminates the equalities from $\mathcal{M}_1$ to produce a mapping $\mathcal{M}_2$ that is also a maximum recovery of $\mathcal{M}$, and then we use some graph-theoretical techniques to produce a mapping $\mathcal{M}^\star$ from $\mathcal{M}_2$ such that $\mathcal{M}^\star \equiv_{\mathrm{CQ}} \mathcal{M}_2$ and the dependencies specifying $\mathcal{M}^\star$ do not include disjunctions in the conclusions. By using Lemma 4.1, we conclude that the resulting mapping $\mathcal{M}^\star$ is also a CQ-maximum recovery of $\mathcal{M}$.

Computing a maximum recovery

We start by presenting the algorithm for computing maximum recoveries of mappings given by tgds developed in the full version of [2]. The algorithm relies on query rewriting in order to compute a maximum recovery. In particular, we assume that given a set Σ of tgds and a conjunctive query $Q(\bar{x})$ over the target schema of Σ, REWRITE$(\Sigma, Q(\bar{x}))$ produces a query in $\mathrm{UCQ}^=$ that is a rewriting of Q over the source.

The study of conjunctive query rewriting has produced several techniques that can be used in the implementation of REWRITE [8, 26, 16]. Therefore, we use the process of query rewriting as a *black box* in our algorithm.

Algorithm: MAXIMUMRECOVERY(Σ) [2]
Input: A set Σ of tgds
Output: A set Σ' that defines a maximum recovery of Σ

1. Let Σ' be empty.
2. For every tgd $\varphi(\bar{x}) \rightarrow \psi(\bar{x})$ in Σ do the following:

    2.1. Let $\alpha(\bar{x})$ be the output of REWRITE$(\Sigma, \psi(\bar{x}))$.

    2.2. Add dependency $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \rightarrow \alpha(\bar{x})$ to Σ'.

3. Return the set Σ'. □

Notice that predicate $\mathbf{C}(\cdot)$ is used to differentiate constant from nulls in the reverse mapping produced by the algorithm MAXIMUMRECOVERY. In fact, this predicate has been used to ensure that only constant values are returned back to the source database. It has been shown that $\mathbf{C}(\cdot)$ is essential in expressing maximum recoveries of mappings given by tgds [2] (as well as in Fagin-inverses and quasi-inverses [14]). We show later that predicate $\mathbf{C}$ is also essential in order to express CQ-maximum recoveries. Another important issue is the form of the conclusion of the dependencies in the output of algorithm MAXIMUMRECOVERY. Notice that formula $\alpha(\bar{x})$ constructed in Step 2.1 is a union of conjunctive queries with equalities, since it is obtained by rewriting the conjunctive query $\psi(\bar{x})$. It follows then that the mapping generated by the algorithm is specified by dependencies with disjunctions and equalities in their conclusions, which have to be eliminated in order to obtain an inverse mapping with the same good properties for data exchange as tgds. We show how to do this in the remaining of this section.

Eliminating equalities

We now proceed to eliminate the equalities that occur in the conclusions of the dependencies generated by algorithm MAXIMUMRECOVERY. The elimination algorithm replaces these equalities by inequalities in the premises of the dependencies. As an example, consider the following dependency.

$$P(x,y) \quad \rightarrow \quad A(x,y) \vee (B(x) \wedge x = y).$$

It is not difficult to see that the above dependency is logically equivalent to the following set of dependencies:

$$P(x,y) \wedge x \neq y \quad \rightarrow \quad A(x,y),$$
$$P(x,x) \quad \rightarrow \quad A(x,x) \vee B(x).$$

To formally describe this process, we need to introduce some terminology. Let $\bar{x} = (x_1, \ldots, x_n)$ be a tuple of distinct variables, and assume that π is a partition of the variables in $\bar{x}$. Then $\pi(x_i)$ is the set of variable in $\bar{x}$ that are assigned to the same class by π. For example, if $\bar{x} = (x_1, x_2, x_3)$ and π is the partition $\{\{x_1\}, \{x_2, x_3\}\}$, then $\pi(x_3) = \{x_2, x_3\}$. Let $f_\pi : \{x_1, \ldots, x_n\} \rightarrow \{x_1, \ldots, x_n\}$ be a function such that $f_\pi(x_i) = x_j$ if j is the minimum index over all the indexes of the variables in $\pi(x_i)$. That is, f_π is a function that selects a unique representative from every class in π. For example, if $\bar{x} = (x_1, x_2, x_3, x_4, x_5)$ and π is the partition $\{\{x_1, x_4\}, \{x_2, x_5\}, \{x_3\}\}$, then $f_\pi(x_1) = x_1$, $f_\pi(x_2) = x_2$, $f_\pi(x_3) = x_3$, $f_\pi(x_4) = x_1$, $f_\pi(x_5) = x_2$. Moreover, given a formula $\alpha(\bar{x})$, we denote by $\alpha(f_\pi(\bar{x}))$ the formula obtained

from $\alpha(\bar{x})$ by replacing every variable x_i by $f_\pi(x_i)$, and we denote by δ_π a formula obtained by taking the conjunction of the inequalities $f_\pi(x_i) \neq f_\pi(x_j)$ whenever $f_\pi(x_i)$ and $f_\pi(x_j)$ are distinct variables. For instance, in the above example we have that δ_π is the formula $x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_2 \neq x_3$. Finally, given a conjunction of equalities α and a conjunction of inequalities β, we say α is *consistent* with β if there is an assignment of values to the variables in α and β that satisfies all the equalities and inequalities in these formulas. For example, $x_1 = x_2$ is consistent with $x_1 \neq x_3$, while $x_1 = x_2 \wedge x_2 = x_3$ is not consistent with $x_1 \neq x_3$. It should be noticed that this consistency problem can be solved in polynomial time.

Algorithm: ELIMINATEEQUALITIES(Σ')

Input: A set Σ' of dependencies of the form $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \to \alpha(\bar{x})$, with $\psi(\bar{x})$ a conjunction of relational atoms and $\alpha(\bar{x})$ a query in UCQ$^=$.

Output: A set Σ'' of dependencies of the form $\rho(\bar{y}) \wedge \mathbf{C}(\bar{y}) \wedge \delta(\bar{y}) \to \gamma(\bar{y})$, with $\rho(\bar{y})$ a conjunction of relational atoms, $\delta(\bar{y})$ a conjunction of inequalities and $\gamma(\bar{y})$ a union of conjunctive queries (without equalities).

1. Let Σ'' be empty.

2. For every dependency σ in Σ' of the form $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \to \alpha(\bar{x})$, and for every partition π of $\bar{x}$ do the following:

   – Let $\alpha(\bar{x}) = \beta_1(\bar{x}) \vee \cdots \vee \beta_k(\bar{x})$.
   – Construct a formula γ from $\alpha(f_\pi(\bar{x}))$ as follows. For every $i \in \{1, \ldots, k\}$:

     – If the equalities in $\beta_i(f_\pi(\bar{x}))$ are consistent with δ_π, then drop the equalities in $\beta_i(f_\pi(\bar{x}))$ and add the resulting formula as a disjunct in γ.

   – If γ has at least one disjunct, then add to Σ'' the dependency $\psi(f_\pi(\bar{x})) \wedge \mathbf{C}(f_\pi(\bar{x})) \wedge \delta_\pi \to \gamma$.

3. Return the set Σ''.   □

For example, assume that the following dependency is in Σ':

$$A(x_1, x_2, x_3) \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \wedge \mathbf{C}(x_3) \to$$
$$[\, P(x_1, x_2) \wedge R(x_1, x_1) \wedge x_2 = x_3 \,] \vee$$
$$[\, \exists y \,(P(x_1, y) \wedge R(x_2, x_3)) \,] \vee$$
$$[\, P(x_1, x_2) \wedge R(x_2, x_3) \wedge x_1 = x_3 \,],$$

and let π be the partition $\{\{x_1\}, \{x_2, x_3\}\}$. Then we have that $f_\pi(x_1) = x_1$, $f_\pi(x_2) = f_\pi(x_3) = x_2$ and δ_π is the formula $x_1 \neq x_2$. Next we show now how formula γ is constructed from this partition in the Step 2 of the algorithm. In the first disjunct, after replacing variables according to f_π we obtain the formula $P(x_1, x_2) \wedge R(x_1, x_1) \wedge x_2 = x_2$. Thus, given that $x_2 = x_2$ is consistent with δ_π, we add $P(x_1, x_2) \wedge R(x_1, x_1)$ as a disjunct in γ. Similarly for the second disjunct, we replace variables according to f_π to obtain formula $\exists y \,(P(x_1, y) \wedge R(x_2, x_2))$, which is added as a disjunct in γ. Finally, for the third disjunct, by applying f_π we obtain $P(x_1, x_2) \wedge R(x_2, x_2) \wedge x_1 = x_2$. But equality $x_1 = x_2$ is not consistent with δ_π and, hence, no further disjunct is added to γ. Notice that γ contains at least one disjunct, thus, we add to Σ'' dependency:

$$A(x_1, x_2, x_2) \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \wedge x_1 \neq x_2 \to$$
$$[P(x_1, x_2) \wedge R(x_1, x_1)] \vee [\exists y \,(P(x_1, y) \wedge R(x_2, x_2))]. \quad (4)$$

The following lemma states a key property of the algorithm ELIMINATEEQUALITIES, namely that the transformations in the algorithm preserve the property of being a maximum recovery.

LEMMA 4.2. *Let Σ be a set of tgds, Σ' the output of* MAXIMUMRECOVERY(Σ)*, and Σ'' the output of* ELIMINA-TEEQUALITIES(Σ')*. Then the mapping specified by Σ'' is a maximum recovery of the mapping specified by Σ.*

Eliminating disjunctions

Let Σ'' be the set obtained by the successive application of algorithms MAXIMUMRECOVERY and ELIMINATEEQUAL-ITIES to a set of tgds. Then, every dependency in Σ'' is of the form:

$$\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \wedge \delta(\bar{x}) \quad \to \quad \beta_1(\bar{x}) \vee \cdots \vee \beta_k(\bar{x}),$$

where $\psi(\bar{x})$ and each $\beta_i(\bar{x})$ are conjunctive queries, and $\delta(\bar{x})$ is a conjunction of inequalities $x \neq x'$ for every pair of distinct variables x, x' in $\bar{x}$. In the last step of our algorithm, we eliminate the disjunctions in the conclusions of the dependencies. Next we describe the machinery used to obtain a set $\Sigma^\star$ that is CQ-equivalent to Σ'', but such that the conclusions of the dependencies of $\Sigma^\star$ are conjunctive queries.

Let Q_1 and Q_2 be two n-ary conjunctive queries, and assume that $\bar{x}$ is the tuple of free variables of Q_1 and Q_2. The *product* of Q_1 and Q_2, denoted by $Q_1 \times Q_2$, is defined as a k-ary conjunctive query (with $k \leq n$) constructed as follows. Let $f(\cdot, \cdot)$ be a one-to-one function from pairs of variables to variables such that: (1) $f(x, x) = x$ for every variable x in $\bar{x}$, and (2) $f(y, z)$ is a fresh variable (mentioned neither in Q_1 nor in Q_2) in any other case. Then for every pair of atoms $R(y_1, \ldots, y_m)$ in Q_1 and $R(z_1, \ldots, z_m)$ in Q_2, the atom $R(f(y_1, z_1), \ldots, f(y_m, z_m))$ is included as a conjunct in the query $Q_1 \times Q_2$. Furthermore, the set of free variables of $Q_1 \times Q_2$ is the set of variables from $\bar{x}$ that are mentioned in $Q_1 \times Q_2$. For example, consider conjunctive queries:

$$Q_1(x_1, x_2) = P(x_1, x_2) \wedge R(x_1, x_1),$$
$$Q_2(x_1, x_2) = \exists y \,(P(x_1, y) \wedge R(x_2, x_2)).$$

Then we have that $Q_1 \times Q_2$ is the conjunctive query:

$$(Q_1 \times Q_2)(x_1) = \exists z_1 \exists z_2 \,(P(x_1, z_1) \wedge R(z_2, z_2)).$$

In this case, we have used a function f such that $f(x_1, x_1) = x_1$, $f(x_2, y) = z_1$, and $f(x_1, x_2) = z_2$. As shown in the example, the free variables of $Q_1 \times Q_2$ do not necessarily coincide with the free variables of Q_1 and Q_2. Notice that the product of two queries may be empty. For example, if $Q_1 = \exists y_1 \exists y_2 P(y_1, y_2)$ and $Q_2 = \exists z_1 R(z_1, z_1)$, then $Q_1 \times Q_2$ is empty.

The definition of the product of queries is motivated by the standard notion of *Cartesian product* of graphs. In fact, if Q_1 and Q_2 are Boolean queries constructed by using a single binary relation $E(\cdot, \cdot)$, then the product $Q_1 \times Q_2$ exactly resembles the graph-theoretical Cartesian product [18]. Moreover, the product $Q_1 \times Q_2$ is a new query that has homomorphisms to both Q_1 and Q_2, and such that for every other query $Q^\star$ that has a homomorphism to Q_1 and Q_2, it holds that $Q^\star$ has a homomorphism to $Q_1 \times Q_2$. In this sense, $Q_1 \times Q_2$ is the "closest" query to both Q_1 and Q_2. Thus, the product of queries is useful in preserving the information that can be retrieved by using conjunctive queries

and, as such, it plays a key role in the following algorithm that eliminates disjunctions.

Algorithm: ELIMINATEDISJUNCTIONS(Σ'')

Input: A set Σ'' of dependencies of the form $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \wedge \delta(\bar{x}) \rightarrow \beta_1(\bar{x}) \vee \cdots \vee \beta_k(\bar{x})$, with $\psi(\bar{x})$ a conjunction of relational atoms, $\delta(\bar{x})$ a conjunction of inequalities $x \neq x'$ for every pair of distinct variables x, x' in $\bar{x}$, and each $\beta_i(\bar{x})$ a conjunctive query.

Output: A conjunctive-query equivalent set $\Sigma^\star$ of dependencies of the form $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \wedge \delta(\bar{x}) \rightarrow \gamma(\bar{x})$, with $\psi(\bar{x})$ and $\delta(\bar{x})$ as above, and $\gamma(\bar{x})$ a conjunctive query.

1. Let $\Sigma^\star$ be empty.

2. For every dependency of the form $\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \wedge \delta(\bar{x}) \rightarrow \beta_1(\bar{x}) \vee \cdots \vee \beta_k(\bar{x})$ in Σ'' do the following:

   – if $\beta_1(\bar{x}) \times \cdots \times \beta_k(\bar{x})$ is not empty, then add the following dependency to $\Sigma^\star$:

   $$\psi(\bar{x}) \wedge \mathbf{C}(\bar{x}) \wedge \delta(\bar{x}) \rightarrow \beta_1(\bar{x}) \times \cdots \times \beta_k(\bar{x}).$$

3. Return the set $\Sigma^\star$.   □

For example, assume that dependency (4) is in Σ''. Then since $[\, P(x_1, x_2) \wedge R(x_1, x_1) \,] \times [\, \exists y\, (P(x_1, y) \wedge R(x_2, x_2)) \,]$ is the query $\exists z_1 \exists z_2\, (P(x_1, z_1) \wedge R(z_2, z_2))$, the following dependency is included in $\Sigma^\star$:

$$A(x_1, x_2, x_2) \wedge \mathbf{C}(x_1) \wedge \mathbf{C}(x_2) \times x_1 \neq x_2 \;\rightarrow$$
$$\exists z_1 \exists z_2\, (P(x_1, z_1) \wedge R(z_2, z_2)). \quad (5)$$

The crucial property of algorithm ELIMINATEDISJUNCTIONS is stated in the following lemma.

LEMMA 4.3. *Let $\Sigma^\star$ be the set of dependencies obtained as output of* ELIMINATEDISJUNCTIONS(Σ'')*. Then the mapping specified by Σ'' is conjunctive-query equivalent to the mapping specified by $\Sigma^\star$.*

Let us give some intuition of why this result holds. Assume that J is the instance $\{A(1, 2, 2)\}$. Then every solution of J under the dependency (4) necessarily contains the tuples $P(1, 2), R(1, 1)$ (corresponding to the first disjunct) or the tuples $P(1, u), R(2, 2)$ for some value u (corresponding to the second disjunct). Thus, the *conjunctive* information that all the solutions of J share is that: the value 1 appears in the first component of a tuple of relation P, and some element appears in both components of the same tuple of relation R. If we now consider the space of solutions for J under dependency (5), we obtain that every solution contains the tuples $P(1, u), R(v, v)$ for some values u and v. That is, the conjunctive information shared by all the solutions of J under dependency (4) is exactly the same as under dependency (5). In fact, (as stated in Lemma 4.3), it can be formally proved that if $\mathcal{M}_1$ and $\mathcal{M}_2$ are the mappings specified by dependencies (4) and (5), respectively, then $\mathcal{M}_1 \equiv_{\mathrm{CQ}} \mathcal{M}_2$, that is, for every instance J and conjunctive query Q over the target of these mappings, it holds that $\underline{\mathrm{certain}}_{\mathcal{M}_1}(Q, J) = \underline{\mathrm{certain}}_{\mathcal{M}_2}(Q, J)$.

Putting it all together

The following is the complete version of the algorithm that computes CQ-maximum recoveries expressed in a language with good properties for data exchange.

Algorithm: CQ-MAXIMUMRECOVERY(Σ)

Input: A set Σ of tgds

Output: A set $\Sigma^\star$ of tgds with inequalities and predicate $\mathbf{C}$ in their premises, that defines a CQ-maximum recovery of Σ.

1. Let Σ' be the output of MAXIMUMRECOVERY(Σ).

2. Let Σ'' be the output of ELIMINATEEQUALITIES(Σ').

3. Let $\Sigma^\star$ be the output of ELIMINATEDISJUNCTIONS(Σ'').

4. Return $\Sigma^\star$.   □

The following theorem states the correctness of the above algorithm. The theorem is a direct consequence of Lemmas 4.1, 4.2, and 4.3.

THEOREM 4.4. *Let Σ be a set of tgds and $\Sigma^\star$ the output of algorithm* CQ-MAXIMUMRECOVERY(Σ)*. Then the mapping specified by $\Sigma^\star$ is a CQ-maximum recovery of the mapping specified by Σ.*

4.2 The language of CQ-maximum recoveries

At this point, a natural question is to which extent the extra features of the rules in the set $\Sigma^\star$ returned by algorithm CQ-MAXIMUMRECOVERY are really needed. Theorem 4.5 states that inequalities and predicate $\mathbf{C}$ in the premises of the output dependencies are both needed if we want to express CQ-maximum recoveries of mappings given by tgds.

THEOREM 4.5. *Let $\mathcal{M}$ be a mapping specified by a set of tgds. Then the following hold.*

(1) $\mathcal{M}$ has a CQ-maximum recovery specified by a set of tgds with inequalities and predicate $\mathbf{C}$ in their premises.

(2) Statement (1) is not necessarily true if we disallow either inequalities or predicate $\mathbf{C}$ in the premises of dependencies.

Part (1) of the theorem follows directly from the properties of the algorithm CQ-MAXIMUMRECOVERY. In [14], Fagin et al. prove that inequalities and predicate $\mathbf{C}$ are needed to express Fagin-inverses of tgds. It should be pointed out that part (2) of the above theorem cannot be obtained as a corollary of the results in [14], as the notion of CQ-maximum recovery is strictly weaker than the notion of Fagin-inverse. In fact, there exist mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ such that: $\mathcal{M}_1$ is specified by a set of tgds and has a Fagin-inverse, $\mathcal{M}_2$ is a CQ-maximum recovery of $\mathcal{M}_1$, but $\mathcal{M}_2$ is not a Fagin-inverse of $\mathcal{M}_1$.

5. COMPUTING INVERSES IN POLYNOMIAL TIME

As we mentioned before, during the last years people have proposed several notions of inversion of schema mappings [10, 14, 2]. One of the limitations of all these proposals is that they have only provided algorithms for computing inverses that work in exponential time and produce inverse mappings of exponential size. In this section, we overcome this limitation by providing the first polynomial time algorithm for the three notions of inversion proposed in [10, 14, 2], and also for the notion of CQ-maximum recovery proposed in this paper.

The algorithm proposed in this section uses an intermediate mapping language that has some form of second-order quantification, and which was inspired by the language proposed in [13] for expressing the composition of two schema mappings. In particular, to compute an inverse of a mapping given by a set Σ of tgds, the algorithm first translate in linear time Σ into a specification Σ' written in this second-order language, and then it computes an inverse for Σ'. In fact, the algorithm is capable of computing in polynomial time an inverse for an arbitrary mapping specified in this second-order language. This result is particularly interesting as there exist mapping languages that are used in practice and have not been taken into account in the previous work on inversion, and for which an inverse can be computed by translating them into this second-order language. For example, this is the case for nested mappings [15], which extend tgds with several desirable features and are used in Clio [19], the IBM data exchange tool. Next we introduce our second-order language, and then we present our polynomial time algorithm for computing inverses.

5.1 Plain SO-tgds

The limitations of tgds as a mapping language for data exchange has been recognized in some studies [13, 5]. In fact, the following example from [13] shows a practical case in which tgds are not powerful enough to capture the desired semantics of data exchange.

EXAMPLE 5.1. Consider the following schemas: A source schema $\mathbf{S}$ consisting of one binary relation $\mathtt{Takes}$, that associates a student name with a course she/he is taking, and a target schema $\mathbf{T}$ consisting of a binary relation $\mathtt{Enrollment}$, that associates a student id with a course that she/he is taking. Intuitively, when translating data from $\mathbf{S}$ to $\mathbf{T}$, one would like to replace the name n of a student by a student id i_n, and then for each course c that is taken by n, one would like to include the tuple (i_n, c) in the table $\mathtt{Enrollment}$. Unfortunately, as it is shown in [13], it is not possible to express this relationship by using a set of tgds. In particular, a tgd of the form:

$$\mathtt{Takes}(n, c) \quad \rightarrow \quad \exists y\, \mathtt{Enrollment}(y, c) \qquad (6)$$

does not express the desired relationship as it may associate a distinct student id y for each tuple (n, c) in $\mathtt{Takes}$ and, thus, it may create several identifiers for the same student name. □

One way to overcome the limitation of tgds in the previous example is by adding a way to associate to each student name n a unique identifier i_n. In fact, this can be done by including a function f that associate with each student name n an id $f(n)$:

$$\mathtt{Takes}(n, c) \quad \rightarrow \quad \mathtt{Enrollment}(f(n), c). \qquad (7)$$

We note that this rule expresses the desired relationship in Example 5.1, as for every course c taken by a student with name n, the tuple $(f(n), c)$ is in the table $\mathtt{Enrollment}$.

The possibility of using function symbols in tgds was identified in [13] as a way to overcome the limitations of tgds for expressing the composition of schema mappings. In this section, we take advantage of this idea, and introduce a mapping language that also uses function symbols (called *plain SO-tgds*), and which is extensively used in our inversion algorithm. But before introducing this language, we need to define the notion of *plain term*. Given a tuple $\bar{x}$ of variables and a tuple $\bar{f}$ of function symbols, a plain term built from $\bar{x}$ and $\bar{f}$ is either a variable x in $\bar{x}$, or a term of the form $f(x_1, \ldots, x_k)$ where each variable x_i is in $\bar{x}$ and f is in $\bar{f}$.

We now define the notion of plain SO-tgd. Given schemas $\mathbf{S}$ and $\mathbf{T}$ with no relation symbols in common, a *plain second-order tgd from $\mathbf{S}$ to $\mathbf{T}$* (plain SO-tgd) is a formula of the form:

$$\exists \bar{f}\, \big[\, \forall \bar{x}_1(\varphi_1 \rightarrow \psi_1) \,\wedge\, \cdots \,\wedge\, \forall \bar{x}_n(\varphi_n \rightarrow \psi_n)\,\big], \qquad (8)$$

where (a) each member of $\bar{f}$ is a function symbol, (b) each formula φ_i is a conjunction of formulas of the form $S(y_1, \ldots, y_k)$, where S is a relation symbol of $\mathbf{S}$ and $y_1, \ldots, y_k$ are (not necessarily distinct) variables in $\bar{x}_i$, and (c) each formula ψ_i is a conjunction of formulas of the form $T(t_1, \ldots, t_\ell)$, where T is a relation symbol of $\mathbf{T}$ and $t_1, \ldots, t_\ell$ are plain terms built from $\bar{x}_i$ and $\bar{f}$. For the sake of readability, we usually omit the second-order quantifier $\exists \bar{f}$ and the first-order quantifiers $\forall \bar{x}_1, \ldots, \forall \bar{x}_n$ from a plain SO-tgd of the form (8), as they are clear from the context. Thus, for example, rule (7) is a plain SO-tgd, as well as the following rule:

$$S(x, y) \quad \rightarrow \quad T(x, f(x, y)) \wedge U(g(x), y).$$

The semantics of plain SO-tgds is defined as follows. Assume that I is a source instance and J is a target instance. An interpretation of a k-ary function f in (I, J) is a function that maps every tuple $(a_1, \ldots, a_k)$ of elements in I to an element in J.

EXAMPLE 5.2. Assume that instance $I = \{\mathtt{Takes}(n_1, c_1),$ $\mathtt{Takes}(n_1, c_2), \mathtt{Takes}(n_2, c_1)\}$ and $J = \{\mathtt{Enrollment}(id_1, c_1),$ $\mathtt{Enrollment}(id_1, c_2), \mathtt{Enrollment}(id_2, c_1)\}$. Then a possible interpretation in (I, J) of a unary function f is $f(n_1) = id_1$, $f(n_2) = id_2$, $f(c_1) = c_1$ and $f(c_2) = c_2$. □

Then we say that a pair (I, J) of instances satisfies a plain SO-tgd of the form (8) if there exists an interpretation in (I, J) of each function in $\bar{f}$ such that (I, J) satisfies each dependency $\forall \bar{x}_i(\varphi_i \rightarrow \psi_i)$ with this interpretation. For example, if I and J are the instances shown in Example 5.2, then (I, J) satisfies plain SO-tgd (7) as (I, J) satisfies this dependency with the interpretation for f given in Example 5.2.

It is not difficult to see that every set of tgds can be transformed in linear time into a plain SO-tgd. For example, tgd (6) is equivalent to the following plain SO-tgd:

$$\mathtt{Takes}(n, c) \quad \rightarrow \quad \mathtt{Enrollment}(f(n, c), c).$$

Notice that the above dependency says that for every tuple (n, c) in the table $\mathtt{Takes}$, there exists a value $f(n, c)$ in the target such that the tuple $(f(n, c), c)$ is in the table $\mathtt{Enrollment}$, which corresponds to the intended semantics of tgd (6). Moreover, every nested mapping [15] can be translated in polynomial time into a plain SO-tgd. Thus, as the algorithm presented in this section is capable of inverting in polynomial time schema mappings given by plain SO-tgds, it can be used to efficiently compute inverses for the mappings most commonly used in practice [11, 15]. But not only that, as the language of plain SO-tgds is strictly more expressive than nested mappings, our algorithm can also be used to compute inverses for other mappings of practical interest.

We conclude this section by noticing that the language of plain SO-tgds is similar to the second-order language proposed in [13]. However, the language proposed in [13] is too

powerful in the sense that there exist mappings specified in this language that are not invertible under any of the inverse notions proposed in the data exchange literature [10, 14, 2]. On the contrary, every mapping specified by plain SO-tgds admits a maximum recovery which can be computed by using the inversion algorithm proposed in this section.

5.2 A polynomial time inversion algorithm

As we have mentioned before, there are three preponderant notions of inverse for schema mappings: Fagin-inverse [10], quasi-inverse [14] and maximum recovery [2]. All the algorithms for computing these three notions work in exponential time and produce inverses of exponential size. In fact, it was proved in the extended version of [2] that for the mapping languages considered in all those papers, there exists a mapping given by a set of tgds such that every maximum recovery of this mapping is of exponential size. In this section, we overcome these limitations and present a polynomial time algorithm that computes maximum recoveries of mappings given by sets of tgds, and uses a different mapping language from those considered in [10, 14, 2] to express inverses. It should be noticed that this algorithm can also be used to compute Fagin-inverses and quasi-inverses for sets of tgds, as from the results in [2], we know that every mapping $\mathcal{M}$ given by a set of tgds admits a maximum recovery, and if $\mathcal{M}$ admits a Fagin-inverse (quasi-inverse), then every maximum recovery of $\mathcal{M}$ is also a Fagin-inverse (quasi-inverse) of $\mathcal{M}$. Interestingly, this algorithm can also be used to deal with the notion of CQ-maximum recovery introduced in this paper, as every maximum recovery is also a CQ-maximum recovery.

More precisely, in this section we present a polynomial time algorithm that, given a set of plain SO-tgds, returns a maximum recovery that is expressed in a language that extends plain SO-tgds with some extra features. Since every set of tgds can be transformed in linear time into a plain SO-tgd, we obtained our desired result.

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

$$R(x, y, z) \quad \rightarrow \quad T(x, f(y), f(y), g(x, z)). \qquad (9)$$

When exchanging data with an SO dependency like (9), the standard assumption is that every application of a function symbol generates a fresh value [13]. For example, consider a source instance $\{R(1, 2, 3)\}$. When we exchange data with (9), 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. Thus, in order to invert a dependency like (9), we consider three unary functions f_1, g_1 and g_2. The idea is that f_1 represents the inverse of 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 (9) with a dependency of the form:

$$T(u, v, v, w) \rightarrow R(u, f_1(v), g_2(w)) \wedge u = g_1(w). \quad (10)$$

Notice that, if we use dependency (10) 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.

In order to obtain a correct algorithm we need another technicality, that we describe next. We have mentioned that when exchanging data with SO dependencies, 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. We do that 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 $g_1(a) \neq f_\star(a)$. Our algorithm adds these equalities and inequalities as conjuncts in the conclusions of dependencies. Considering the example, our algorithm adds

$$f_1(v) = f_\star(v) \ \wedge \ g_1(v) \neq f_\star(v) \ \wedge$$
$$g_1(w) = f_\star(w) \ \wedge \ f_1(w) \neq f_\star(w)$$

to the conclusion of dependency (10).

Before presenting our algorithm, we make some observations. First, notice that plain SO-tgds are closed under conjunction and, thus, a set of plain SO-tgds is equivalent to a single plain SO-tgd. For this reason, we assume that our algorithm has as input a single plain SO-tgd $\lambda = \exists \bar{f}(\sigma_1 \wedge \cdots \wedge \sigma_2)$. Second, 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 5.3 that the algorithm presented in this section produces inverses for the general case, where no assumption about the function symbols is made.

Preliminary 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 (9), $F_\lambda = \{f, g\}$ and $F'_\lambda = \{f_1, g_1, g_2, f_\star\}$.

We describe the procedures CREATETUPLE, ENSUREINV, and SAFE. These procedures are the building blocks of the algorithm that computes an inverse 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 conclusion of dependency (9). 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 premise of dependency (10).

Tuple $\bar{u}$ created with CREATETUPLE is used as an input in the following two procedures. We now formalize the procedure to obtain 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 $\bar{u} = (u_1, \ldots, u_n)$ of (not necessarily distinct) variables, and an n-tuple of plain terms $\bar{s}$ built from F_λ and a tuple of $\bar{y}$ variables.

Output: A formula Q_e consisting of conjunctions of equalities between terms built from F'_λ and $\bar{u}, \bar{y}$.

1. Let $\bar{s} = (s_1, \ldots, s_n)$.
2. Construct formula Q_e as follows. For every $i \in \{1, \ldots, n\}$ do the following:

   - If s_i is a variable y, then add equality $u_i = y$ as a conjunct in Q_e.
   - If s_i is a term of the form $f(y_1, \ldots, y_k)$, 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 $f_1, \ldots, f_k$ are the k unary functions in F'_λ associated with f.

3. Return Q_e. □

As an example, let λ be the dependency (9), $\bar{s} = (x, f(y), f(y), g(x, z))$ the tuple of terms in the conclusion 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 2, the conjunction $g_1(w) = x \wedge g_2(w) = z$ is added to formula Q_e. The final output of the algorithm in this case is:

$$u = x \wedge f_1(v) = y \wedge g_1(w) = x \wedge g_2(w) = z. \quad (11)$$

We need to describe one more procedure, which guarantees that it could not be the case that a value in the target was generated by two distinct functions.

Procedure: SAFE$(\lambda, \bar{u}, \bar{s})$

Input: A plain SO-tgd λ, an n-tuple $\bar{u} = (u_1, \ldots, u_n)$ of not necessarily distinct variables, and an n-tuple of plain terms $\bar{s}$ 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. Let $\bar{s} = (s_1, \ldots, s_n)$.
2. Construct formula Q_s as follows. For every $i \in \{1, \ldots, n\}$ do the following:

   - If s_i is a term of the form $f(y_1, \ldots, y_k)$, 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_λ different from f.

3. Return Q_s. □

Considering λ as the dependency (9), $\bar{s}$ 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:

$$f_1(v) = f_\star(v) \wedge g_1(v) \neq f_\star(v) \wedge$$
$$g_1(w) = f_\star(w) \wedge f_1(w) \neq f_\star(w). \quad (12)$$

Notice that all the procedures presented so far work in polynomial time with respect to the size of their inputs.

Building the inverse

We need some additional notation before presenting the algorithm for computing inverses of plain SO-tgds. 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 plain SO-tgd λ in polynomial time. As a consequence of the results in [2] and the discussion in Section 3.1, the algorithm can also be used to compute Fagin-inverses, quasi-inverses, as well as CQ-maximum recoveries.

Algorithm: POLYSOINVERSE(λ)

Input: A plain SO-tgd $\lambda = \exists \bar{f}(\sigma_1 \wedge \cdots \wedge \sigma_n)$.

Output: An SO dependency λ' that specifies a maximum recovery of λ.

1. Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, and Σ' be empty.
2. (*Normalize Σ*) For every $i \in \{1, \ldots, n\}$ do:

   - If σ_i is of the form $\varphi(\bar{x}) \rightarrow R_1(\bar{t}_1) \wedge \ldots \wedge R_\ell(\bar{t}_\ell)$, then replace σ_i by ℓ dependencies $\varphi(\bar{x}) \rightarrow R_1(\bar{t}_1), \ldots, \varphi(\bar{x}) \rightarrow R_\ell(\bar{t}_\ell)$.

3. (*Construct Σ'*) For every σ of the form $\varphi(\bar{x}) \rightarrow R(\bar{t})$ in the normalized set Σ, where $\bar{t} = (t_1, \ldots, t_m)$ is a tuple of plain terms, do the following:

   a. Let $\bar{u} = $ 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 Γ_σ as follows. For every dependency $\psi(\bar{y}) \rightarrow R(\bar{s})$ in Σ such that $\bar{s}$ subsumes $\bar{t}$, do the following:

      - Let $Q_e = $ ENSUREINV$(\lambda, \bar{u}, \bar{s})$.
      - Let $Q_s = $ SAFE$(\lambda, \bar{u}, \bar{s})$.
      - Add to Γ_σ the formula $\exists \bar{y} \left(\psi(\bar{y}) \wedge Q_e \wedge Q_s \right)$

   d. Add to Σ' the dependency:

   $$prem_\sigma(\bar{u}) \rightarrow \gamma_\sigma(\bar{u})$$

   where $\gamma_\sigma(\bar{u})$ is the disjunction of the formulas in Γ_σ.

4. Return the SO dependency $\lambda' = \exists \bar{f}' \left(\bigwedge \Sigma' \right)$, where $\bar{f}'$ is a tuple containing the function symbols in F'_λ. □

Let λ be the plain SO-tgd (9). In Step 3 of algorithm POLYSOINVERSE(λ) we have to consider a single dependency $\sigma = R(x, y, z) \rightarrow T(x, f(y), f(y), g(x, z))$. 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

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

is added to the set Γ_σ, where Q_e is the formula (11) and Q_p is the formula (12). Finally, the formula:

$$T(u,v,v,w) \wedge \mathbf{C}(u) \;\rightarrow$$
$$\exists x \exists y \exists z \left(R(x,y,z) \wedge u = x \;\wedge\; f_1(v) = y \;\wedge \right.$$
$$\left. g_1(w) = x \;\wedge\; g_2(w) = z \wedge Q_p \right) \quad (13)$$

is the output of the algorithm. Notice that dependency (13) is equivalent to:

$$T(u,v,v,w) \wedge \mathbf{C}(u) \rightarrow$$
$$R(u, f_1(v), g_2(w)) \wedge g_1(w) = u \;\wedge\; Q_p,$$

which specifies a maximum recovery of λ.

THEOREM 5.3. *Let $\mathcal{M}$ be a mapping specified by a plain SO-tgd λ. Algorithm* POLYSOINVERSE(λ) *computes in polynomial time an SO dependency λ' that specifies a maximum recovery of $\mathcal{M}$.*

From the results in [2], we obtain the following corollary.

COROLLARY 5.4. *Let $\mathcal{M}$ be a mapping specified by a plain SO-tgd λ. If $\mathcal{M}$ has a Fagin-inverse (quasi-inverse), then algorithm* POLYSOINV(λ) *computes in polynomial time a Fagin-inverse (quasi-inverse) of $\mathcal{M}$.*

It is important to notice that since every set of tgds can be transformed into a plain SO-tgd in linear time, our algorithm can be used to compute Fagin-inverses, quasi-inverses, and maximum recoveries for sets of tgds. This is the first polynomial time algorithm capable of doing this. However, the gain in time complexity comes with the price of a stronger and less manageable mapping language for expressing inverses.

6. CONCLUDING REMARKS

In this paper, we have revisited the problem of inverting schema mappings paying special attention to some practical concerns. We consider two orthogonal limitations of the previous approaches, namely, the language needed to represent inverses and the efficiency of the algorithms to compute them. We have proposed solutions to both of these problems. In particular, one of our main contributions is the development of a polynomial time algorithm to compute all the notions of inverses of schema mappings proposed in the literature. This is the first efficient algorithm capable of computing these notions.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. *PODS*, pp. 229–240, 2004.

[2] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: bringing exchanged data back. *PODS*, pp. 13-22, 2008.

[3] P. Bernstein. Applying model management to classical meta data problems. *CIDR*, 2003.

[4] P. Bernstein, S. Melnik. Model management 2.0: manipulating richer mappings. *SIGMOD*, pp 1-12, 2007.

[5] B. ten Cate, P. Kolaitis. Structural Characterizations of Schema-Mapping Languages. To be in *ICDT*, 2009.

[6] G. Giacomo, D. Lembo, M. Lenzerini, R. Rosati. On reconciling data exchange, data integration, and peer data management. *PODS*, pp. 133-142, 2007.

[7] A. Deutsch and V. Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. *DBPL*, pp. 21–39, 2001.

[8] O. Duschka, M. Genesereth. Answering Recursive Queries Using Views. *PODS*, pp. 109-116, 1997

[9] R. Fagin. Horn clauses and database dependencies. *JACM*, 29(4):952–985, 1982.

[10] R. Fagin. Inverting schema mappings. *TODS*, 32(4), 2007.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

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

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

[14] R. Fagin, P. Kolaitis, L. Popa, and W.-C. Tan. Quasi-inverses of schema mappings. *PODS*, pp. 123-132, 2007.

[15] A. Fuxman, M. Hernández, H. Ho, R. Miller, P. Papotti, L. Popa Nested Mappings: Schema Mapping Reloaded *VLDB*, pp. 67-78, 2006

[16] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.* 10(4): 270-294 (2001)

[17] A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, I. Tatarinov. The Piazza Peer Data Management System. *IEEE TKDE* 16(7):787-798 (2004)

[18] P. Hell and J. Nesetril. Graphs and Homomorphisms. Oxford University Press, 2004.

[19] M. A. Hernández, H. Ho, L. Popa, A. Fuxman, R. J. Miller, T. Fukuda and P. Papotti. Creating Nested Mappings *ICDE*, pp. 1487-1488, 2007.

[20] T. Imielinski and W. Lipski Jr. Incomplete information in relational databases. *JACM*, 31(4):761-791, 1984.

[21] M. Lenzerini. Data Integration: A Theoretical Perspective.. *PODS*, pp. 233–246, 2002.

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

[23] S. Melnik. Generic model management: concepts and algorithms. Volume 2967 of *LNCS*, Springer, 2004.

[24] S. Melnik, P. Bernstein, A. Y. Halevy, and E. Rahm. Supporting executable mappings in model management. *SIGMOD*, pp. 167–178, 2005.

[25] A. Nash, P. Bernstein, S. Melnik. Composition of mappings given by embedded dependencies. *PODS* pp. 172–183, 2005.

[26] R. Pottinger, A. Y. Halevy. MiniCon: A scalable algorithm for answering queries using views. *VLDB J.* 10(2-3): 182-198 (2001)