Laconic Schema Mappings: Computing the Core with SQL Queries

Balder ten Cate
INRIA and ENS Cachan
balder.tencate@inria.fr

Laura Chiticariu
IBM Almaden
chiti@almaden.ibm.com

Phokion Kolaitis
UC Santa Cruz and
IBM Almaden
kolaitis@cs.ucsc.edu

Wang-Chiew Tan
UC Santa Cruz
wctan@cs.ucsc.edu

ABSTRACT

A schema mapping is a declarative specification of the relationship between instances of a source schema and a target schema. The data exchange (or data translation) problem asks: given an instance over the source schema, materialize an instance (or solution) over the target schema that satisfies the schema mapping. In general, a given source instance may have numerous different solutions. Among all the solutions, universal solutions and core universal solutions have been singled out and extensively studied. A universal solution is a most general one and also represents the entire space of solutions, while a core universal solution is the smallest universal solution and is unique up to isomorphism (hence, we can talk about the core).

The problem of designing efficient algorithms for computing the core has attracted considerable attention in recent years. In this paper, we present a method for directly computing the core by SQL queries, when schema mappings are specified by source-to-target tuple-generating dependencies (s-t tgds). Unlike prior methods that, given a source instance, first compute a target instance and then recursively minimize that instance to the core, our method avoids the construction of such intermediate instances. This is done by rewriting the schema mapping into a laconic schema mapping that is specified by first-order s-t tgds with a linear order in the active domain of the source instances. A laconic schema mapping has the property that a "direct translation" of the source instance according to the laconic schema mapping produces the core. Furthermore, a laconic schema mapping can be easily translated into SQL, hence it can be optimized and executed by a database system to produce the core. We also show that our results are optimal: the use of the linear order is inevitable and, in general, schema mappings with constraints over the target schema cannot be rewritten to a laconic schema mapping.

1. INTRODUCTION

A schema mapping specifies the relationship between instances of a source schema and a target schema. The data exchange (aka data translation) problem asks: given a source instance, transform it into a target instance so the schema mapping is satisfied. Such a target instance is called a *solution* for the given source instance. Data translation underlies numerous data inter-operability applications and has been the subject of research that dates back to more than thirty-years ago [14]. In the past, schema mappings were expressed procedurally, as a query that can be directly executed to compute a solution. In recent years, systems such as Clio [10] and HePToX [1] adopt a declarative logical formalism for specifying schema mappings.

More formally, a *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, and Σ is the specification of the relationship between $\mathbf{S}$ and $\mathbf{T}$. In recent data exchange literature, Σ is given by a finite set of *source-to-target tuple generating dependencies (s-t tgds)*, target tgds, and target *equality-generating dependencies (egds)*. Intuitively, the s-t tgds of a schema mapping dictate the existence of certain facts in a solution for a given source instance. The target tgds and target egds are constraints over the target schema $\mathbf{T}$ that further "shape" the facts dictated by the s-t tgds for a given source instance. Target tgds and target egds contain as special cases such important dependencies as inclusion dependencies and functional dependencies, respectively.

In general, a given source instance may have no solutions, since it may not be possible to materialize a solution that satisfies a target egd. On the other hand, a given source instance may have a multitude of solutions. Intuitively, this is so because, while the s-t tgds of a schema mapping dictate the existence of certain facts in a solution of a given source instance, they do not spell out what should not be in a solution for the given source instance. Furthermore, s-t tgds may not specify how certain attributes of a relation in the target schema should be populated with values from the given source instance. As a consequence, there are numerous ways to materialize these unknown values.

Prior research [5] has shown that, among all solutions of a given source instance, the *universal solutions* are the preferred ones because they are the most general and also encapsulate the entire space of solutions. Furthermore, it was shown in [7] that the *core* of a universal solution is the smallest universal solution and is unique up to isomorphism. Henceforth, we shall talk about *the core universal solution* or, simply, *the core*. In addition to being the smallest universal solution, the core possesses certain other good properties. Specifically, among all universal solutions, the core returns the most conservative answers on conjunctive queries with inequalities. In other words, the result of evaluating a conjunctive query with inequalities Q over the core is contained in the result of evaluating Q over any univeral solution. Furthermore, in a precise sense, the core is the solution that satisfies the most embedded dependencies.

Earlier Work on Computing the Core For schema mappings specified by s-t tgds, Clio and HePToX compute universal solutions by first compiling the schema mapping into a script in an ex-

VLDB '09, August 24-28, 2009, Lyon, France

ecutable language, such as SQL, which can then be executed on a given source instance to construct a universal solution for that instance. Such a framework for computing a universal solution has several advantages. In particular, this framework is able to push the computation process into off-the-shelf transformation engines, such as relational database management systems.

In general, the universal solution produced by the above method is not the core. In view of the desirable properties of the core (especially, being the smallest universal solution), one would like to have a method to compute the core using SQL queries. It should be noted that computing the core of an arbitrary database instance is an NP-hard problem [2]. It was shown, however, that for broad classes of schema mappings, there are polynomial-time algorithms for computing the core of universal solutions. Indeed, for schema mappings specified by s-t tgds and target egds (in particular, for schema mappings specified by s-t tgds), two different polynomial-time algorithms for computing the core of universal solutions were given in [7]; the first is a *greedy* algorithm, while the second is a *blocks* algorithm. Furthermore, for schema mappings specified by s-t tgds, target egds, and target tgds obeying the *weak acyclicity* condition, a polynomial-time algorithm for computing the core based on a sophisticated extension of the blocks algorithm was given in [9].

In contrast to the direct method of computing a universal solution using SQL queries, all the above algorithms for computing the core are recursive algorithms that rely on extra post-processing steps on an intermediate target instance. Specifically, they are based on the following generic methodology: first, compute an intermediate target instance for the given source instance; second, recursively minimize the intermediate target instance until the core is obtained.

In more concrete terms, for a schema mapping $\mathcal{M}$ specified by s-t tgds and target egds, the *greedy algorithm*, given a source instance I, first computes a target instance J that is universal for I. After this, the greedy algorithm will repeatedly remove tuples from J one at a time, as long as the s-t tgds of $\mathcal{M}$ are satisfied. When no more tuples can be removed, the resulting instance is the core of the universal solutions for I. The blocks algorithm for a schema mapping $\mathcal{M}$ specified by s-t tgds and target egds begins by computing a target instance J that is a universal solution for I with respect to the s-t tgds of $\mathcal{M}$. After this, the blocks algorithm computes a sequence of intermediate instances such that the next instance is both a proper subinstance and a homomorphic image of the preceding instance via a homomorphism that is the identity everywhere except for a *block* (a connected component) of nulls. When no proper subinstance of the current instance is the homomorphic image of the current instance via such a homomorphism, then the current instance is the core. Both the greedy algorithm and the blocks algorithm terminate after a number of iterations that, in general, depends on the size of the given source instance. Thus, both these algorithms are inherently recursive and do not give rise to a computation of the core via SQL queries.

Summary of Contributions In this paper, we address the following question: Can the core be computed using SQL queries? In other words, can one leverage off-the-shelf relational database systems and compute the core while adhering to the framework of systems such as Clio and HePToX?

This question was first addressed by one of the authors of this paper in [3], where it was shown that for schema mappings specified by a syntactically restricted (and rather limited) class of s-t tgds, the core can be computed using SQL queries (see Section 5.1). Here, we present a method that applies to every schema mapping specified by s-t tgds and makes it possible to compute the core using SQL queries. Unlike the aforementioned prior methods that, given a source instance, first compute a target instance

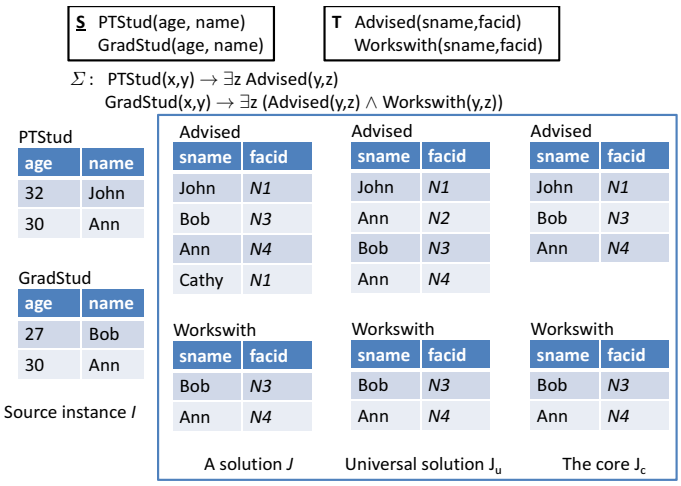

Figure 1: An example of a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, a source instance I, a solution J for I, a universal solution J_u for I, and the core J_c for I.

and then recursively minimize that instance into a core, our method avoids the construction of such intermediate instances. Instead, we first rewrite a schema mapping specified by s-t tgds into a logically equivalent *laconic* schema mapping that is specified by first-order s-t tgds with a linear order in the active domain of the source instance. A laconic schema mapping has the property that a "direct translation" of the source instance according to the laconic schema mapping produces the core. Furthermore, a laconic schema mapping can be easily translated into SQL, hence it can be optimized and executed by a database system to produce the core. Our method of computing the core can be easily integrated into existing data exchange systems by adding a module that rewrites the schema mapping and slightly extending the existing SQL translation module to handle first-order s-t tgds with a linear order. We also show that our results are optimal; the linear order that may arise in the rewrite is necessary and our method cannot be extended to schema mappings that involve constraints over the target schema.

In [12], similar results were independently obtained for a restricted class of s-t tgds, and empirical data is provided showing that their method outperforms existing approaches to computing core universal solutions.

Paper Outline In the next section, we recall basic notions and properties of schema mappings. Section 3 explains how the canonical universal solution of a source instance with respect to a schema mapping specified by first-order s-t tgds can be obtained using SQL queries. In Section 4, we introduce the notion of laconic schema mappings, and present our algorithm for transforming a schema mapping specified by first-order s-t tgds into a logically equivalent laconic schema mapping specified by first-order s-t tgds, assuming a linear order on the active domain of the source instance. In Sections 5 and 6, we demonstrate the optimality of our algorithm.

2. BACKGROUND AND NOTATION

We present the necessary background and results related to the core; we also illustrate various concepts by means of an example.

Instances and homomorphisms We assume that there is an infinite set Const of constant values and an infinite set Vars of null values that is disjoint from Const. We further assume that we have a fixed linear order $<$ on the set Const of all constant values. We consider source instances to have values from Const and target in-

stances to have values from $\underline{\mathsf{Const}} \cup \underline{\mathsf{Vars}}$. We use $\mathrm{dom}(I)$ to denote the set of values that occur in facts in the instance I. A *homomorphism* $h : I \to J$, with I, J instances of the same schema, is a function $h : \underline{\mathsf{Const}} \cup \underline{\mathsf{Vars}} \to \underline{\mathsf{Const}} \cup \underline{\mathsf{Vars}}$ with $h(a) = a$ for all $a \in \underline{\mathsf{Const}}$, such that for all relations R and all tuples of (constant or null) values $(v_1, \ldots, v_n) \in R^I$, $(h(v_1), \ldots, h(v_n)) \in R^J$. Instances I, J are *homomorphically equivalent* if there are homomorphisms $h : I \to J$ and $h' : J \to I$. An *isomorphism* $h : I \cong J$ is a homomorphism that is a bijection between $\mathrm{dom}(I)$ and $\mathrm{dom}(J)$ and that preserves truth of atomic formulas in both directions. Intuitively, nulls act as placeholders for actual (constant) values, and a homomorphism from I to J captures the fact that J "contains more, or at least as much information" as I.

Query languages We will denote by CQ, UCQ, and FO the sets of conjunctive queries, unions of conjunctive queries, and first-order queries, respectively. The sets $\mathrm{CQ}^<$, $\mathrm{UCQ}^<$, and $\mathrm{FO}^<$ are defined similarly, except that the queries may refer to the linear order. Thus, unless indicated explicitly, it is assumed that queries do not refer to the linear order. For a query q and an instance I, we denote by $q(I)$ the answers of q in I; furthermore, we denote by $q(I)_\downarrow$ the ground answers of q, i.e., $q(I)_\downarrow = q(I) \cap \underline{\mathsf{Const}}^k$ for k the arity of q. In other words, $q(I)_\downarrow$ contains the tuples from $q(I)$ that consist entirely of constants.

Schema mappings, solutions, universal solutions A schema mapping is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S}$ and $\mathbf{T}$ are the disjoint source and target schemas respectively, and Σ is a finite set of sentences of some logical language over the schema $\mathbf{S} \cup \mathbf{T}$. From a semantic point of view, a schema mapping can be identified with the set of all pairs (I, J) such that I is a source instance, J is a target instance and (I, J) satisfies Σ (which we denote by $(I, J) \models \Sigma$). Two schema mappings, $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$, are *logically equivalent* if Σ and Σ' are logically equivalent, i.e., they are satisfied by the same pairs of instances. Given a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and a source instance I, a *solution* for I with respect to $\mathcal{M}$ is a target instance J such that (I, J) satisfies Σ. We denote the set of solutions for I with respect to $\mathcal{M}$ by $\mathsf{Sol}_\mathcal{M}(I)$, or simply $\mathsf{Sol}(I)$ when the schema mapping is clear from the context. A universal solution for a source instance I with respect to a schema mapping $\mathcal{M}$ is a solution $J \in \mathsf{Sol}_\mathcal{M}(I)$ such that, for every $J' \in \mathsf{Sol}_\mathcal{M}(I)$, there is a homomorphism from J to J'.

We will consider the following logical languages for specifying schema mappings. A *source-to-target tuple generating dependency* or, in short, an *s-t tgd*, is a first-order sentence of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \to \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y}))$, where $\phi(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$, and $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$, such that each variable in $\mathbf{x}$ occurs in $\phi(\mathbf{x})$. A *LAV s-t tgd* is a s-t tgd in which ϕ is a single atomic formula. A *full* s-t tgd is a s-t tgd in which there are no existentially quantified variables (i.e., $\mathbf{y}$ is the empty set). The class of *first-order s-t tgds (*FO s-t tgds*)* generalizes s-t tgds by allowing the antecedent $\phi(\mathbf{x})$ to be an arbitrary FO-formula over $\mathbf{S}$. The class of $\mathrm{FO}^<$ s-t tgds is defined similarly, allowing also comparisons of the form $x_i < x_j$ to be used in the antecedent. In what follows and in order to simplify notation, we will typically drop the outermost universal quantifiers when writing s-t tgds, FO s-t tgds, or $\mathrm{FO}^<$ s-t tgds.

It is common in data exchange to consider schema mappings specified using also target constraints in the form of *target tgds* and *target egds*. We will only discuss such target constraints in detail in Section 6 and therefore postpone the relevant definitions to that section.

Example 2.1 An example of a schema mapping is depicted in Figure 1. Both s-t tgds in that figure are LAV s-t tgds. Given the source instance I, three solutions (J, J_u, and J_c) for I are shown. The values N_1, ..., N_4 in the solutions are nulls from $\underline{\mathsf{Vars}}$. All other values are from $\underline{\mathsf{Const}}$. Even though J is a solution for I, the solution J contains an unnecessary tuple, namely, (Cathy, N_1). In other words, the result of removing (Cathy, N_1) from J, which is J_u, is still a solution. In fact, J_u is a universal solution. Intuitively, J_u is the most general solution because it does not make unnecessary assumptions about the existence of other tuples or values in place of the nulls. The solution J_c is also a universal solution (we shall explain what is meant by "the core" shortly). There is a homomorphism $h : J_u \to J$, where $h(v) = v$ for every $v \in \mathrm{dom}(J_u)$. There is also a homomorphism $h' : J_u \to J_c$, where $h'(N_2) = N_4$ and $h(v) = v$ for every $v \in \mathrm{dom}(J_u)$ and $v \neq N_2$. Clearly, since $J_c \subseteq J_u$, there is also a homomorphism from J_c to J_u. □

2.1 The Core

As stated in the Introduction, a source instance may have a multitude of solutions. Among all solutions, the universal solutions have been singled out as the preferred ones because they are the most general ones (i.e., for a given source instance I, a universal solution for I has a homomorphism into any solution for I). Among the universal solutions, the *core universal solution* plays a special role. A target instance J is said to be a *core* if there is no proper subinstance $J' \subseteq J$ and homomorphism $h : J \to J'$. An equivalent definition in terms of retractions is as follows: A subinstance $J' \subseteq J$ is called a *retract* of J if there is a homomorphism $h : J \to J'$ such that for all $a \in dom(J')$, $h(a) = a$. The corresponding homomorphism h is called a *retraction.* A retract is *proper* if it is a proper subinstance of the original instance. A core of a target instance J is a retract of J that has itself no proper retracts. Every (finite) target instance has a unique core, up to isomorphism. Moreover, two instances are homomorphically equivalent if and only if they have isomorphic cores. It follows that, for every schema mapping $\mathcal{M}$, every source instance has at most one core universal solution up to isomorphism. Indeed, if the schema mapping $\mathcal{M}$ is specified by FO s-t tgds then each source instance has *exactly* one core universal solution up to isomorphism [7]. We will therefore freely speak of *the* core.

Example 2.2 Referring back to the schema mapping in Figure 1, the solution J_c is the core for the source instance I. Intuitively, every tuple in J_c must exist in order for (I, J_c) to satisfy Σ. So there are no redundant tuples in J_c. □

In addition to being the smallest universal solution, the core has certain other desirable properties. Specifically, the core also returns the most conservative answers on conjunctive queries with inequalities: if Q is a conjunctive query with inequalities, and J is the core universal solution for a given source instance I, then $Q(J)_\downarrow$ is contained in $Q(J')_\downarrow$ for every universal solution J' of I. Furthermore, in a precise sense, the core is the universal solution that satisfies the most embedded dependencies. Indeed, it is easy to show that if a (arbitrary) tgd or egd holds in a universal solution, then it must also hold in the core. To make this precise, let a *disjunctive embedded dependency* be a first-order sentence of the from $\forall \mathbf{x}(\phi(\mathbf{x}) \to \bigvee_i \exists \mathbf{y}_i . \psi_i(\mathbf{x}, \mathbf{y}_i))$, where ϕ, ψ_i are conjunctions of atomic formulas over the target schema $\mathbf{T}$ and/or equalities. Then the following result holds.

Theorem 2.1 *Let $\mathcal{M}$ be a schema mapping, I be a source instance, let J the core universal solution of I, and let J' be any other universal solution of I, i.e., one that is not a core. Then*

- *Every disjunctive embedded dependency true in J' is true in J,*
- *Some disjunctive embedded dependency true in J is false in J'.*

The naive chase procedure
Input: A schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ and a source instance I where Σ is a finite set of FO$^<$ s-t tgds
Output: A universal solution J for I w.r.t. $\mathcal{M}$

$J := \emptyset$;
for all $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi(\mathbf{x}, \mathbf{y})) \in \Sigma$ **do**
  for all tuples of constants $\mathbf{a}$ such that $I \models \phi(\mathbf{a})$ **do**
    for each $y_i \in \mathbf{y}$, pick a fresh null value N_i for y_i.
    add the facts in $\psi(\mathbf{a}, \mathbf{N})$ to J.
  end for
end for;
return J

Figure 2: Naive chase method for computing universal solutions.

The proof is omitted for lack of space.
Concerning the complexity of computing the core, we have:

Theorem 2.2 ([7]) *Let $\mathcal{M}$ be a schema mapping specified by FO$^<$ s-t tgds. There is a polynomial-time algorithm such that, given a source instance I, the algorithm returns the core universal solution for I.*

Strictly speaking, this result was shown in [7] only for schema mappings specified by s-t tgds and target egds. However, the same argument applies for schema mappings specified by FO$^<$ s-t tgds (and target egds).

Although the data complexity of computing core solutions is polynomial time, the degree of the polynomial depends on the schema mapping in question. Indeed, it was shown in [9] that computing core universal solutions for schema mappings specified by s-t tgds is fixed parameter intractable, where the parameter is the maximum number of variables in occurring in each s-t tgd.

3. USING SQL TO COMPUTE UNIVERSAL SOLUTIONS

In this section, we define canonical universal solutions, and we describe how FO$^<$ s-t tgds can be compiled into SQL queries that, when executed against any source instance I, produce the canonical universal solution for I. As we will see in Section 4, when this method is applied to laconic schema mappings, the SQL queries obtained produce the core universal solution of I.

In [5], it was shown that, for schema mappings specified by s-t tgds, the *chase* procedure can be used to compute a universal solution for a given source instance. In fact, the same holds true for schema mappings specified by FO$^<$ s-t tgds. Figure 2 describes a variant of the chase procedure known as the *naive chase*. For a source instance I and schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds, the result of applying the naive chase is called a *canonical universal solution* of I with respect to $\mathcal{M}$. Observe that the result of the naive chase is unique up to isomorphism, since it depends only on the exact choice of fresh nulls. Also note that, even if two schema mappings are logically equivalent, they may assign different canonical universal solutions to a given source instance.

Example 3.1 The naive chase procedure on the schema mapping $\mathcal{M}$ and source instance I of Figure 1 produces the universal solution J_u shown in the same figure. Intuitively, the first s-t tgd in Σ on the `PTStud` relation caused the first two facts of `Advised` relation to be created. The second s-t tgd in Σ on the `GradStud` relation caused the last two facts of the `Advised` relation and all facts of `Workswith` relation to be created. □

It is easy to see that, for schema mappings specified by FO$^<$ s-t tgds, the naive chase procedure can be implemented using SQL queries. In fact, Clio follows this approach [8]. We illustrate the approach by returning to our running example of the schema mapping in Figure 1.

The first step is to *Skolemize* each s-t tgd in Σ. By this, we mean replacing each existentially-quantified variable with a function term $f(\mathbf{x})$, where f is a fresh function symbol of appropriate arity and $\mathbf{x}$ denotes the set of all universally-quantified variables in the tgd. For example, after this step on Σ, we get:

$\text{PTStud}(x, y) \rightarrow \text{Advised}(y, f(x, y))$
$\text{GradStud}(x, y) \rightarrow \text{Advised}(y, g(x, y)) \wedge \text{Workswith}(y, g(x, y))$

These dependencies are logically equivalent to the following dependencies with a single relational atom in the right-hand-side:

$\text{PTStud}(x, y) \rightarrow \text{Advised}(y, f(x, y))$
$\text{GradStud}(x, y) \rightarrow \text{Advised}(y, g(x, y))$
$\text{GradStud}(x, y) \rightarrow \text{Workswith}(y, g(x, y))$

Next, for each target relation R we collect the dependencies that contain R in the right-hand-side, and we interpret these as constituting a definition of R. In this way, we get the following definitions of `Advised` and `Workswith`.

$\texttt{Advised} := \{(y, f(x, y)) \mid \text{PTStud}(x, y)\} \cup \{(y, g(x, y)) \mid \text{GradStud}(x, y)\}$
$\texttt{Workswith} := \{(y, g(x, y)) \mid \text{GradStud}(x, y)\}$

In general, the definition of a k-ary target relation $R \in \mathbf{T}$ will be of the shape:

$$R := \{(t_1(\mathbf{x}), \ldots, t_k(\mathbf{x})) \mid \phi(\mathbf{x}))\} \cup \cdots \cup \{(t'_1(\mathbf{x}'), \ldots, t'_k(\mathbf{x})) \mid \phi'(\mathbf{x}')\} \quad (1)$$

where $t_1, \ldots, t_k, \ldots, t'_1, \ldots, t'_k$ are *terms* (i.e., variables such as x_1, or functions over terms, such as $f(x_1, x_2)$), $\phi, \ldots, \phi'$ are first-order queries over the source schema. Each $\phi, \ldots, \phi'$ corresponds to a SQL query, and the union of these SQL queries is a query that when executed on any source instance I will compute the canonical universal solution for I. To continue with our running example, the following SQL queries may be generated for `Advised` and `Workswith`:

```
Advised:
select distinct name,
       concat("f(",age,name,")")
from   PTStud
union
select distinct name,
       concat("g(",age,name,")")
from   GradStud
```

```
Workswith:
select distinct name,
       concat("g(",age,name,")")
from   GradStud
```

Evaluating the SQL query associated to `Advised` on the source instance I in Figure 1 yields the tuples { (John, f(32,John)), (Ann, f(30,Ann)), (Bob, g(27,Bob)), (Ann, g(30,Ann)) }. The terms f(32,John), f(30,Ann), g(27,Bob), and g(30,Ann) correspond, respectively, to the nulls N_1, N_2, N_3, N_4 in J_u of Figure 1.

The general idea behind the construction of the SQL queries should be clear from the example. The translation assumes the existence of a `concat` function that returns the concatenation of all its arguments. Intuitively, the result of the `concat` function represents a null.

Note that, in this example, the resulting SQL queries are unions of select-project-join queries (i.e., unions of conjunctive queries) augmented with the use of the `concat` function. In particular, they do not contain any `GROUP BY` clauses or any aggregate functions. In the case of schema mappings specified by FO s-t tgds, the

(a) $P(x) \rightarrow \exists yz.R(x,y) \wedge R(x,z)$
(a′) $P(x) \rightarrow \exists y.R(x,y)$

(b) $P(x) \rightarrow \exists y.R(x,y)$
$P(x) \rightarrow R(x,x)$
(b′) $P(x) \rightarrow R(x,x)$

(c) $R(x,y) \rightarrow S(x,y)$
$P(x) \rightarrow \exists y.S(x,y)$
(c′) $R(x,y) \rightarrow S(x,y)$
$P(x) \wedge \neg\exists y.R(x,y) \rightarrow \exists y.S(x,y)$

(d) $R(x,y) \rightarrow \exists z.S(x,y,z)$
$R(x,x) \rightarrow S(x,x,x)$
(d′) $R(x,y) \wedge x \neq y \rightarrow \exists z.S(x,y,z)$
$R(x,x) \rightarrow S(x,x,x)$

(e) $R(x,y) \rightarrow \exists z.(S(x,z) \wedge S(y,z))$
(e′) $(R(x,y) \vee R(y,x)) \wedge x \leq y \rightarrow \exists z.(S(x,z) \wedge S(y,z))$

(f) $\text{PTStud}(x,y) \rightarrow \exists z.\text{Advised}(y,z)$
$\text{GradStud}(x,y) \rightarrow \exists z.(\text{Advised}(y,z) \wedge \text{Workswith}(y,z))$
(f′) $\text{PTStud}(x,y) \wedge \neg\exists u.\text{GradStud}(u,y) \rightarrow \exists z.\text{Advised}(y,z)$
$\text{GradStud}(x,y) \rightarrow \exists z.(\text{Advised}(y,z) \wedge \text{Workswith}(y,z))$

Figure 3: Examples of non-laconic schema mappings (a-f) and their laconic equivalents (a′-f′).

same approach will give rise to SQL queries that use the difference (EXCEPT) operator but still do not contain any GROUP BY clauses or any aggregate functions. Finally, in the case of schema mappings specified by FO$^<$ s-t tgds, the resulting SQL queries require the use of comparisons of the form $x < y$ in the WHERE clause, but no further constructs. Also note that the translation from schema mappings to SQL queries computing the canonical universal solution is polynomial.

To summarize, we have explained how, for schema mappings specified by FO$^<$ s-t tgds, canonical universal solutions can be obtained using SQL queries that do not contain any GROUP BY clauses or any aggregate functions, i.e., that belong to the (pure) relational calculus fragment of SQL, except for the use of string concatenation.

4. LACONIC SCHEMA MAPPINGS

In this section, we present an algorithm for tranforming any schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds into a logically equivalent one $\mathcal{M}'$, such that the naive chase procedure applied to $\mathcal{M}'$ and to a source instance I produces the core universal solution for I and $\mathcal{M}$. In particular, this shows that, for schema mappings specified by FO$^<$ s-t tgds, the core universal solution can be computed using SQL queries.

Definition 4.1 A schema mapping is *laconic* if for every source instance I, the canonical universal solution of I with respect to $\mathcal{M}$ is the core universal solution of I with respect to $\mathcal{M}$. □

Note that the definition only makes sense for schema mappings specified by FO$^<$ s-t tgds, because we have defined the notion of a canonical universal solution only for such schema mappings.

Examples of laconic and non-laconic schema mappings are given in Figure 3. For Example 3(d), the canonical universal solution of the source instance $I = \{R(a,a)\}$ is $\{S(a,a,N), S(a,a,a)\}$, which is not the core universal solution of I. Clearly, one should only "translate" according to the first s-t tgd in Example 3(d) if $x \neq y$, which explains the antecedent of the first tgd in Example 3(d′). It is easy to see that every schema mapping specified by full s-t tgds only (i.e., s-t tgds without existential quantifiers) is laconic. Indeed, in this case, the canonical universal solution does not contain any nulls, and hence is guaranteed to be the core. Thus, being specified by full s-t tgds is a sufficient condition for laconicity, although a rather uninteresting one. The following provides us with a necessary condition, which explains why the schema mapping in Figure 3(a) is not laconic. Given an s-t tgd $\forall\mathbf{x}(\phi \rightarrow \exists\mathbf{y}.\psi)$, by *the canonical instance of* ψ, we will mean the (unordered) target instance whose facts are the conjuncts of ψ, where the $\mathbf{x}$ variables are treated as constants and the $\mathbf{y}$ variables as nulls.

Proposition 4.1 *If a schema mapping $(\mathbf{S}, \mathbf{T}, \Sigma)$ specified by s-t tgds is laconic, then for each s-t tgd $\forall\mathbf{x}(\phi \rightarrow \exists\mathbf{y}.\psi) \in \Sigma_{st}$, the canonical instance of ψ is a core.*

The proof is omitted for lack of space.

In the case of schema mapping (e) in Figure 3, the linear order is used in order to obtain a logically equivalent laconic schema mapping (e′). Note that the schema mapping (e′) is *order-invariant* in the sense that the set of solutions of a source instance I does not depend on the interpretation of the $<$ relation in I, as long as it is a linear order. Still, the use of the linear order cannot be avoided, as we will show in Section 5.1. What is really going on, in this example, is that the right hand side of (e) has a non-trivial automorphism (viz. the map sending x to y and vice versa), and the conjunct $x \leq y$ in the antecedent of (e′) plays, intuitively, the role of a tie-breaker, cf. Section 4.1.3.

Testing whether a given schema mapping is laconic is not a tractable problem:

Proposition 4.2 *Testing laconicity of schema mappings specified by FO s-t tgds is undecidable. It is coNP-hard already for schema mappings specified by LAV s-t tgds.*

In fact, testing laconicity of schema mappings specified by s-t tgds is coNP-complete. We omit the proof for lack of space.

4.1 Making schema mappings laconic

In this section, we present a procedure for transforming any schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds into a logically equivalent *laconic* schema mapping $\mathcal{M}'$ specified by FO$^<$ s-t tgds. The laconic schema mapping can then be translated into SQL queries, as described in Section 3, which when executed on any source instance will produce the core universal solution.

To simplify the notation, throughout this section, we assume a fixed input schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, with Σ a finite set of FO$^<$ s-t tgds. Moreover, we will assume that the FO$^<$ s-t tgds $\forall\mathbf{x}(\phi \rightarrow \exists\mathbf{y}.\psi) \in \Sigma$ are non-decomposable [7], meaning that the fact graph of $\exists\mathbf{y}.\psi(\mathbf{x},\mathbf{y})$, where the facts are the conjuncts of ψ and two facts are connected if they have an existentially quantified variable in common, is connected. This assumption is harmless: every FO$^<$ s-t tgd can be decomposed into a logically equivalent finite set of non-decomposable FO$^<$ s-t tgds (with identical left-hand-sides, one for each connected component of the fact graph) in polynomial time.

The outline of the procedure for making schema mappings laconic is as follows (the items correspond to subsections of the present section):

(1) Construct a finite list *fact block types*: these are descriptions of potential "patterns" of tuples in the core. (See Section 4.1.1.)

(2) Compute for each of the fact block types a *precondition*: a first-order formula over the source schema that tells exactly when

the core will contain a fact block of the given type. (See Section 4.1.2.)

(3) If any of the fact block types has non-trivial automorphisms, add an additional side-condition, consisting of a Boolean combination of formulas of the form $x_i < x_j$. Side conditions prevent the creation of multiple copies of the same fact block in the canonical universal solution. (See Section 4.1.3.)

(4) Construct the new schema mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$, where Σ' contains an FO$^<$ s-t tgd for each of the fact block types. The left-hand-side of the FO$^<$ s-t tgd is the conjunction of the pre-condition and side-condition of the respective fact block type, while the right-hand-side is the fact block type itself. (See Section 4.1.4.)

Next, we illustrate our approach with an example. The technical notions that we use in discussing the example will be formally defined in the next subsections.

Example 4.1 Consider the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S} = \{P, Q\}$, $\mathbf{T} = \{R_1, R_2\}$, and Σ consists of the s-t tgds

$$P(x) \rightarrow \exists y.R_1(x, y)$$
$$Q(x) \rightarrow \exists yzu.(R_2(x, y) \wedge R_2(z, y) \wedge R_1(z, u))$$

In this case, there are exactly three relevant fact block types. They are listed below, together with their preconditions.

Fact block type			*Precondition*
$t_1(x; y)$	$=$	$\{R_1(x, y)\}$	$pre_{t_1}(x) = P(x)$
$t_2(x; yzu)$	$=$	$\{R_2(x, y), R_2(z, y), R_1(z, u)\}$	$pre_{t_2}(x) = Q(x) \wedge \neg P(x)$
$t_3(x; y)$	$=$	$\{R_2(x, y)\}$	$pre_{t_3}(x) = Q(x) \wedge P(x)$

We use the notation $t(\mathbf{x}; \mathbf{y})$ for a fact block type to indicate that the variables $\mathbf{x}$ stand for constants and the variables $\mathbf{y}$ stand for distinct nulls.

As it happens, the above fact block types have no non-trivial automorphisms. Hence, no side-conditions need to be added, and Σ' will consist of the following FO s-t tgds:

$$\begin{array}{lcl} P(x) & \rightarrow & \exists y.R_1(x, y) \\ Q(x) \wedge \neg P(x) & \rightarrow & \exists yzu.(R_2(x, y) \wedge R_2(z, y) \wedge R_1(z, u)) \\ Q(x) \wedge P(x) & \rightarrow & \exists y.(R_2(x, y)) \end{array}$$

The reader may verify that in this case, the obtained schema mapping is indeed laconic. We prove in Section 4.1.4 that the output of our transformation is guaranteed to be a laconic schema mapping that is logically equivalent to the input schema mapping. □

We now proceed to define the notions appearing in this example.

4.1.1 Generating the fact block types

The first step is to generate all fact block types of the schema mapping. To formalize the notion of a fact block type, we first define the concept of a fact graph of an instance. The *fact graph* of an instance I is the graph whose nodes are the facts $R(\mathbf{v})$ (with R a k-ary relation and $\mathbf{v} \in (\underline{\mathsf{Const}} \cup \underline{\mathsf{Vars}})^k$, $k \geq 0$) true in I, and such that there is an edge between two facts if they have a null value in common. A *fact block*, or *f-block* for short, of an instance is a connected component of the fact graph of the instance. We know from [6] that, for any schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds, the size of f-blocks in core of any source instance for $\mathcal{M}$ is bounded by the maximum number of atomic formulas in the right-hand-side of the FO$^<$ s-t tgds in $\mathcal{M}$.[1] Consequently, there is a finite number of f-block types, such that every core universal solution consists of f-blocks of these types. This is a crucial observation that we will exploit in our construction.

Formally, an *f-block type* $t(\mathbf{x}; \mathbf{y})$ will be a finite set of atomic formulas in $\mathbf{x}, \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ are disjoint sets of variables. We will refer to $\mathbf{x}$ as the *constant variables* of t and $\mathbf{y}$ as the *null variables*. We say that a f-block type $t(\mathbf{x}; \mathbf{y})$ is a *renaming* of a f-block type $t'(\mathbf{x}'; \mathbf{y}')$ if there is a bijection g between $\mathbf{x}$ and $\mathbf{x}'$ and between $\mathbf{y}$ and $\mathbf{y}'$, such that $t' = \{R(g(\mathbf{v})) \mid R(\mathbf{v}) \in t\}$. In this case, we write $g : t \cong t'$ and we call g also a renaming. We will not distinguish between f-block types that are renamings of each other. We say that a f-block B has type $t(\mathbf{x}; \mathbf{y})$ if B can be obtained from $t(\mathbf{x}; \mathbf{y})$ by replacing constant variables by constants and null variables to distinct nulls, i.e., if $B = t(\mathbf{a}; \mathbf{N})$ for some sequence of constants $\mathbf{a}$ and sequence of distinct nulls $\mathbf{N}$. Note that we require the relevant substitution to be injective on the null variables but not necessarily on the constant variables. If a target instance J contains a block $B = t(\mathbf{a}; \mathbf{N})$ of type $t(\mathbf{x}; \mathbf{y})$ then we say that $t(\mathbf{x}; \mathbf{y})$ is *realized* in J *at* $\mathbf{a}$. Note that, in general, a f-block type may be realized more than once at a tuple of constants $\mathbf{a}$, but this will not happen if the target instance J is a core universal solution.

We are interested in the f-block types that may be realized in core universal solutions. Eventually, the schema mapping $\mathcal{M}'$ that we will construct from $\mathcal{M}$ will contain an FO$^<$ s-t tgd for each relevant f-block type. Not every f-block type as defined above can be realized. We may restrict attention to a subclass. Below, by the canonical instance of a f-block type $t(\mathbf{x}; \mathbf{y})$, we will mean the instance containing the facts in $t(\mathbf{x}; \mathbf{y})$, considering $\mathbf{x}$ as constants and $\mathbf{y}$ as nulls.

Definition 4.2 The set $\text{TYPES}_{\mathcal{M}}$ of f-block types generated by $\mathcal{M}$ consists of all f-block types $t(\mathbf{x}; \mathbf{y})$ satisfying the following conditions:

(a) Σ contains an FO$^<$ s-t tgd $\forall \mathbf{x}'(\phi(\mathbf{x}') \rightarrow \exists \mathbf{y}'.\psi(\mathbf{x}', \mathbf{y}'))$ with $\mathbf{y} \subseteq \mathbf{y}'$, and $t(\mathbf{x}; \mathbf{y})$ is the set of conjuncts of ψ in which the variables $\mathbf{y}' - \mathbf{y}$ do not occur;

(b) The canonical instance of $t(\mathbf{x}; \mathbf{y})$ is a core;

(c) The fact graph of the canonical instance of $t(\mathbf{x}; \mathbf{y})$ is connected.

If some f-block types generated by $\mathcal{M}$ are renamings of each other, we add only one of them to $\text{TYPES}_{\mathcal{M}}$. □

The main result of this subsection is:

Proposition 4.3 *Let J be the core of a source instance I with respect to $\mathcal{M}$. Then each f-block of J has type $t(\mathbf{x}; \mathbf{y})$ for some $t(\mathbf{x}; \mathbf{y}) \in \text{TYPES}_{\mathcal{M}}$.*

PROOF. Let B be any f-block of J. Since J is a core universal solution, it is, up to isomorphism, an induced subinstance of the canonical universal solution J' of I. It follows that J' must have a f-block B' such that B is the restriction of B' to domain of J. Since B' is a connect component of the fact graph of J', it must have been created in a single step during the naive chase. In other words, there is an FO$^<$ s-t tgd

$$\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow \exists \mathbf{y}.\psi(\mathbf{x}, \mathbf{y}))$$

and an assignment g of constants to the variables $\mathbf{x}$ and distinct nulls to the variables $\mathbf{y}$ such that B' is contained in the set of conjuncts of $\psi(g(\mathbf{x}), g(\mathbf{y}))$. Moreover, since we assume the FO$^<$ s-t tgds of $\mathcal{M}$ to be non-decomposable and B' is a a connected component of the fact graph of J, B' must be exactly the set of facts listed in $\psi(g(\mathbf{x}), g(\mathbf{y}))$. In other words, if we let $t(\mathbf{x}; \mathbf{y})$ be the set of all facts listed in ψ, then B' has type $t(\mathbf{x}; \mathbf{y})$. Finally, let

[1] This is stated in [6] for schema mappings specified by s-t tgds, but the same holds for FO$^<$ s-t tgd.

$t'(\mathbf{x}';\mathbf{y}') \subseteq t(\mathbf{x};\mathbf{y})$ be the set of all facts from $t(\mathbf{x};\mathbf{y})$ containing only variables y_i for which $g(y_i)$ occurs in B. Since B is the restriction of B' to the domain of J, we have that B is of type $t'(\mathbf{x}';\mathbf{y}')$. Moreover, the fact graph of the canonical instance of J is connected because B is connected, and the canonical instance of $t'(\mathbf{x}';\mathbf{y}')$ is a core, because, if it would not be, then B would not be a core either, and hence J would not be a core either, which would lead to a contradiction. It follows that $t'(\mathbf{x}';\mathbf{y}') \in \text{TYPES}_{\mathcal{M}}$. □

Note that $\text{TYPES}_{\mathcal{M}}$ contains only finitely many f-block types. Still, the number is in general exponential in the size of the schema mapping, as the following example shows.

Example 4.2 Consider the schema mapping specified by the following s-t tgds:

$$P_i(x) \rightarrow P_i'(x) \qquad \text{(for each } 1 \leq i \leq k\text{)}$$
$$Q(x) \rightarrow \exists y_0 y_1 \ldots y_k (R(x,y_0) \wedge \textstyle\bigwedge_{1\leq i\leq k}(R(y_i,y_0) \wedge P_i'(y_i)))$$

For each $S \subseteq \{1,\ldots,k\}$, the f-block type

$$t_S(x;(y_i)_{i\in S\cup\{0\}}) = \{R(x,y_0)\} \cup \{R(y_i,y_0), P_i'(y_i) \mid i \in S\}$$

belongs to $\text{TYPES}_{\mathcal{M}}$. Indeed, each of these 2^k f-block types is realized in the core universal solution of a source instance. The example can be modified to use a fixed source and target schema: replace $P_i'(x)$ by $S(x,x_1)\wedge S(x_1,x_2)\wedge \ldots S(x_{i-1},x_i)\wedge S(x_i,x_i)$. □

The same example can be used to show that the smallest logically equivalent schema mapping that is laconic can be exponentially longer.

Generating f-block types for $\mathcal{M}$. Our algorithm generates f-block types based of each FO$^<$ s-t tgd in $\mathcal{M}$. For each FO$^<$ s-t tgd $\forall \mathbf{x}'(\phi(\mathbf{x}') \rightarrow \exists \mathbf{y}'.\psi(\mathbf{x}',\mathbf{y}'))$ in $\mathcal{M}$, we exhaustively consider all subsets $\mathbf{y} \subseteq \mathbf{y}'$ and determine the set $t(\mathbf{x};\mathbf{y})$ of conjuncts of ψ in which the variables $\mathbf{y}' - \mathbf{y}$ do not occur. Subsequently, $t(\mathbf{x};\mathbf{y})$ is determined to be in $\text{TYPES}_{\mathcal{M}}$ if (1) the canonical instance of $t(\mathbf{x};\mathbf{y})$ is a core and (2) the corresponding fact graph is connected. The test for (1) involves determining whether a proper retraction is possible. Finally, to ensure that $\text{TYPES}_{\mathcal{M}}$ does not contain two renamings of the same f-block type, we test for each pair of types whether there is a renamings, and if so, we eliminate one of them.

4.1.2 Computing the precondition of a f-block type

The main result of this subsection is Proposition 4.4 below, which shows that whether a f-block type is realized in the core at a given sequence of constants $\mathbf{a}$ is something that can be tested by a first-order query on the source.

Our construction makes use of the notion of *certain answers*. Given a source instance I, a schema mapping $\mathcal{M}$, and a target query q, we will denote by $certain_{\mathcal{M},q}(I)$, the set of *certain answers* to q in I with respect to $\mathcal{M}$, i.e., the intersection $\bigcap_{J\in \mathsf{Sol}_{\mathcal{M}}(I)} q(J)$. In other words, a tuple of values is a certain answer to q if it belongs to the set of answers of q, no matter which solution of I one picks. There are two methods to compute certain answers to a conjunctive query. The first method uses *universal solutions* [5] and the second uses *query rewriting*. Next, we shall briefly describe the method based on query rewriting, which is relevant for our construction of the precondition of a f-block type. In the query rewriting method, a given query q_T over the target schema $\mathbf{T}$ is rewritten to a query q_S over the source schema $\mathbf{S}$ such that $q_S(I)$ directly computes the certain answers to the original query.

Theorem 4.1 *Let L be any of UCQ, UCQ$^<$, FO, FO$^<$. Then for every schema mapping $\mathcal{M}$ specified by s-t tgds and for every L-query q over $\mathbf{T}$, one can compute in exponential time an L-query q_S over $\mathbf{S}$ defining $certain_{\mathcal{M},q}$. That is, for every source instance I, it is the case that $q_S(I) = certain_{\mathcal{M},q}(I)$.*

There are various ways in which such certain answer queries can be obtained. One possibility is to split up the schema mapping $\mathcal{M}$ into a composition $\mathcal{M}_1 \circ \mathcal{M}_2$, with $\mathcal{M}_1$ specified by full s-t tgds and $\mathcal{M}_2$ specified by LAV s-t tgds, and then to successively apply the known query rewriting techniques of MiniCon [13] and full s-t tgd unfolding (cf. [11]). In [15], an alternative rewriting method was given for the case where $L =$ FO or $L =$ FO$^<$, which can be used to transform an target query q into a source query q' defining $certain_{\mathcal{M},q}$ over source instances whose domain contains at least two elements, in polynomial time (combined complexity).

Next, we use Theorem 4.1 to construct, for each f-block type $t(\mathbf{x};\mathbf{y})$, a FO$^<$ query q such that for any given source instance I, the results $q(I)$ contain exactly the tuples of constants at which $t(\mathbf{x};\mathbf{y})$ is realised in the core universal solution.

Proposition 4.4 *For each $t(\mathbf{x};\mathbf{y}) \in \text{TYPES}_{\mathcal{M}}$, there is a FO$^<$ query $precon_t(\mathbf{x})$, such that for every source instance I with core universal solution J, and for every tuple of constants $\mathbf{a}$, the following are equivalent:*

- $\mathbf{a} \in precon_t(I)$
- *$t(\mathbf{x};\mathbf{y})$ is realized in J at $\mathbf{a}$.*

PROOF. We first define an intermediate formula $precon'_t(\mathbf{x})$ that almost satisfies the required properties, but not quite yet. For each f-block type $t(\mathbf{x};\mathbf{y})$, let $precon'_t(\mathbf{x})$ be the following formula:

$$\begin{aligned} & certain_{\mathcal{M}}(\exists \mathbf{y}. \textstyle\bigwedge t)(\mathbf{x}) \\ & \quad \wedge \bigwedge_{i\neq j} \neg certain_{\mathcal{M}}(\exists \mathbf{y}_{-i}. \textstyle\bigwedge t[y_i/y_j])(\mathbf{x}) \\ & \quad \wedge \bigwedge_{i} \neg\exists x'.certain_{\mathcal{M}}(\exists \mathbf{y}_{-i}. \textstyle\bigwedge t[y_i/x'])(\mathbf{x},x') \end{aligned} \tag{2}$$

where $\mathbf{y}_{-i}$ stands for the sequence $\mathbf{y}$ with y_i removed, and $t[u/v]$ is the result of replacing each occurrence of u by v in t. By construction, if $precon_t(\mathbf{a})$ holds in I, then every universal solution J satisfies $t(\mathbf{a};\mathbf{N})$ for some some sequence of distinct nulls $\mathbf{N}$. Still, it may not be the case that $t(\mathbf{x};\mathbf{y})$ is realized at $\mathbf{a}$, since it may be that that $t(\mathbf{a};\mathbf{N})$ is part of a bigger f-block. To make things more precise, we introduce the notion of an embedding. For any two f-block types, $t(\mathbf{x};\mathbf{y})$ and $t'(\mathbf{x}';\mathbf{y}')$, an *embedding* of the first into the second is a function h mapping $\mathbf{x}$ into $\mathbf{x}'$ and mapping $\mathbf{y}$ injectively into $\mathbf{y}'$, such that whenever t contains an atomic formula $R(\mathbf{z})$, then $R(h(\mathbf{z}))$ belongs to of t'. The embedding h is *strict* if t' contains an atomic formula that is not of the form $R(h(\mathbf{z}))$ for any $R(\mathbf{z}) \in t$. Intuitively, the existence of a strict embedding means that t' describes a f-block that properly contains the f-block described by t.

Let I be any source instance, J the core universal solution of I, $t(\mathbf{x};\mathbf{y}) \in \text{TYPES}_{\mathcal{M}}$, and $\mathbf{a}$ a sequence of constants.

Claim 1: If t is realized in J at $\mathbf{a}$, then $\mathbf{a} \in precon'_t(I)$.

Proof of claim: Clearly, since t is realized in J at $\mathbf{a}$ and J is a universal solution, the first conjunct of $precon'_t$ is satisfied. That the rest of the query is satisfied is also easily seen: otherwise J would not be a core. *End of proof of claim 1.*

Claim 2: If $\mathbf{a} \in precon'_t(I)$, then either t is realized in J at $\mathbf{a}$ or some f-block type $t'(\mathbf{x}';\mathbf{y}') \in \text{TYPES}_{\mathcal{M}}$ is realized at a tuple of constants $\mathbf{a}'$, and there is a strict embedding $h : t \to t'$ such that $a_i = a'_j$ whenever $h(x_i) = x'_j$.

Proof of claim: It follows from the construction of $precon'_t$, and the definition of $\text{TYPES}_{\mathcal{M}}$ types, that the witnessing assignment for its truth must send all existential variables to distinct nulls, which belong to the same block. By Proposition 4.3, the diagram of this block is a specialization of a f-block type $t' \in \text{TYPES}_{\mathcal{M}}$. It follows that t is embedded in t' and $\mathbf{a}$, together with possible some additional values in Const, realize t'. *End of proof of claim 2.*

We now define $precon_t(\mathbf{x})$ to be the following formula:

$$precon'_t(\mathbf{x}) \ \wedge \bigwedge_{\substack{t'(\mathbf{x}';\mathbf{y}') \in \text{TYPES}_{\mathcal{M}} \\ h\,:\,t(\mathbf{x};\mathbf{y}) \to t'(\mathbf{x}';\mathbf{y}') \text{ a strict embedding}}} \neg\exists\mathbf{x}'.\Big(\bigwedge_i (x_i = h(x_i)) \ \wedge \ precon'_{t'}(\mathbf{x}')\Big) \tag{3}$$

This formula satisfies the required conditions: $\mathbf{a} \in precon_t(I)$ iff $t(\mathbf{x};\mathbf{y})$ is realized in J at $\mathbf{a}$. The left-to-right direction follows from Claim 1 and 2, while the right-to-left direction follows from Claim 1 and 2 together with the fact that J is a core. □

Example 4.3 To see how Equation (2) in the above construction is put into effect, consider Example (d) of Figure 3. There are two f-block types, corresponding to $t_1(xy;z) = S(x,y,z)$ and $t_2(x) = S(x,x,x)$. For the first f-block type t_1, the first conjunct of Equation 2 will return $R(x,y)$. The second conjunct of Equation 2 rewrites to true, since there is only one existential variable y which cannot be replaced by a distinct other. The third conjunct, in effect, translates to $\neg\exists x'.certain_{\mathcal{M}}(S(x,y,x'))$. Since only $R(x,x)$ will produce S-tuples that when evaluated with $S(x,y,x')$ will produce tuples that consists entirely of constants, this means that $\neg\exists x'.certain_{\mathcal{M}}(S(x,y,x'))$ rewrites to $\neg\exists x'(R(x,x) \wedge x = y = x')$. Putting all the conjuncts together and simplifying them, the conjuncts are equivalent to $R(x,y) \wedge x \neq y$, which explains the antecedent in the first FO s-t tgd in Example (d′). Note that there are no strict embeddings in this example.

To see how Equation (3) in the above construction is put into effect, consider our example in Figure 1 (also shown as Example (f) in Figure 3). There are two fact block types, corresponding to $t_1(y;z) = \{\text{Advised}(y,z)\}$ and $t_2(v;w) = \{\text{Advised}(v,w), \text{Workswith}(v,w)\}$. For $precon'_{t_1}(y)$, Equation (2) generates $(\exists x.\text{PTStud}(x,y) \vee \exists x.\text{GradStud}(x,y))$. Equation (3) adds $\neg\exists v.(y = v \wedge \exists u.\text{GradStud}(u,v))$ to $precon'_{t_1}(y)$. This is because there is a strict embedding from t_1 to t_2 that maps $y \mapsto v$ and $z \mapsto w$. The resulting precondition $precon_{t_1}(y)$ is thus: $\text{PTStud}(x,y) \wedge \neg\exists u.\text{GradStud}(u,y)$. This explains the antecedent in the first FO s-t tgd in Example (f′). □

Generating preconditions for a f-block type. Our procedure for generating the preconditions for a f-block type relies on the rewriting algorithm of [13] and query unfolding algorithm of full s-t tgds (c.f [11]). The schema mapping $\mathcal{M}$ is first split into a sequence of two schema mappings $\mathcal{M}_1 = (\mathbf{S}, \mathbf{U}, \Sigma_1)$ and $\mathcal{M}_2 = (\mathbf{U}, \mathbf{T}, \Sigma_2)$ as follows: For each s-t tgd $\phi_i(\mathbf{x}_i) \to \exists\mathbf{y}_i.\psi(\mathbf{x}_i,\mathbf{y}_i) \in \Sigma$, the set Σ_1 will contain a full s-t tgd $\phi_i(\mathbf{x}_i) \to U_i(\mathbf{x}_i)$, and Σ_2 will contain a LAV s-t tgd $U_i(\mathbf{x}_i) \to \exists\mathbf{y}_i.\psi(\mathbf{x}_i,\mathbf{y}_i)$. The sets Σ_1 and Σ_2 consists of only such tgds. It is easy to see that since each s-t tgd σ_i in Σ has a unique corresponding relational schema $U_i \in \mathbf{U}$ that exports all universally-quantified variables of σ_i, the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ is logically equivalent to $\mathcal{M}$. Given a query over the target schema $\mathbf{T}$, we first apply the rewriting algorithm of [13] for LAV s-t tgds over $\mathcal{M}_2$ to obtain an intermediate query over the schema $\mathbf{U}$. Subsequently, we apply the query unfolding algorithm (c.f [11]) to obtain a query over the schema $\mathbf{S}$. This explains how $precon'_t(\mathbf{x})$ (i.e., Equation (2)) is obtained in general. The next step is to find all strict embeddings among f-block types as required by Equation (3). This involves an exhaustive test for all possible strict embeddings among all pairs of f-block types. There may be more efficient ways to compute preconditions, for example using the polynomial time algorithm for constructing certain answer queries from [15] we mentioned earlier, and exploring this is part of our future work.

4.1.3 Computing the side-conditions of a f-block type

The issue we address in this subsection, namely that of *non-rigid* f-block types, is best explained by an example.

Example 4.4 Consider again schema mapping (e) in Figure 3. This schema mapping is not laconic. Indeed, if we have the source instance with two tuples $\{R(a,b), R(b,a)\}$, where $a \neq b$, the canonical universal solution is $\{S(a,N_1), S(b,N_1), S(b,N_2), S(a,N_2)\}$. This canonical universal solution has two distinct nulls N_1 and N_2 and it is clearly not the core. The essence of the problem is in the fact that the right-hand-side of the dependency is "symmetric": it is a non-trivial renaming of itself, the renaming in question being $\{x \mapsto y, y \mapsto x\}$. According to the terminology that we will introduce below, the right-hand-side of this dependency is *non-rigid*. Intuitively, this mean that there are two distinct ways in which identical target fact blocks (up to renaming of nulls) may be generated. Schema mapping (e′) from Figure 3 does not suffer from this problem, because it contains $x \leq y$ in the antecedent, and we are assuming $<$ to be a linear order on the values in the source instance. □

In order to capture when identical target facts (up to renaming of nulls) are generated, we say that two f-blocks, B, B', are *copies of each other*, if there is a bijection g from Const to Const and from Vars to Vars such that $g(a) = a$ for all $a \in$ Const and $B' = \{R(g(v_1),\ldots,g(v_k)) \mid R(v_1,\ldots,v_k) \in B\}$. In other words, B' can be obtained from B by renaming null values. Next, we formalize the condition under which there cannot be two distinct ways of generating copies of a f-block.

Definition 4.3 A f-block type $t(\mathbf{x};\mathbf{y})$ is *rigid* if for any two sequences of constants $\mathbf{a}, \mathbf{a}'$ and for any two sequences of distinct nulls $\mathbf{N}, \mathbf{N}'$, if $t(\mathbf{a};\mathbf{N})$ and $t(\mathbf{a}';\mathbf{N}')$ are copies of each other, then $\mathbf{a} = \mathbf{a}'$. □

Clearly, the s-t tgd from Example 4.4 is non-rigid. A simple variation of the argument in the same example shows:

Proposition 4.5 *If a f-block type $t(\mathbf{x};\mathbf{y})$ is non-rigid, then the schema mapping specified by the FO (in fact LAV) s-t tgd $\forall\mathbf{x}(R(\mathbf{x}) \to \exists\mathbf{y}.\bigwedge t(\mathbf{x};\mathbf{y}))$ is not laconic.*

In other words, if a f-block type is non-rigid, the s-t tgd that corresponds to the f-block cannot be naively chased without running the risk of non-laconicity. Fortunately, it turns out that f-block types can be made rigid by the addition of suitable side-conditions. By a *side-condition* $\Phi(\mathbf{x})$, we mean a Boolean combination of formulas of the form $x_i < x_j$ or $x_i = x_j$. Intuitively, the role of the side-condition is to ensures that there cannot be two distinct ways of generating copies of a f-block.

Definition 4.4 A f-block type $t(\mathbf{x};\mathbf{y})$ is *rigid relative to a side condition* $\Phi(\mathbf{x})$ if for any two sequences of constants $\mathbf{a}, \mathbf{a}'$ satisfying

$\Phi(\mathbf{a})$ and $\Phi(\mathbf{a}')$ and for any two sequences of distinct nulls $\mathbf{N}, \mathbf{N}'$, if $t(\mathbf{a}; \mathbf{N})$ and $t(\mathbf{a}'; \mathbf{N}')$ are copies of each other, then $\mathbf{a} = \mathbf{a}'$. □

At the same time, we would like to ensure that the side-condition is not too strong. Intuitively, this means that whenever a f-block type should be realized in a core universal solution, there will be at least one way of arranging the variables so that the side-condition is satisfied.

Definition 4.5 A side-condition $\Phi(\mathbf{x})$ is *safe* for a f-block type $t(\mathbf{x}; \mathbf{y})$ if for every f-block $t(\mathbf{a}; \mathbf{N})$ of type t there is a f-block $t(\mathbf{a}'; \mathbf{N}')$ of type t satisfying $\Phi(\mathbf{a}')$ such that the two are copies of each other. □

The next result shows that every f-block type can be safely turned rigid by a side-condition.

Proposition 4.6 *For every f-block type $t(\mathbf{x}; \mathbf{y})$ there is a side condition $sidecon_t(\mathbf{x})$ such that $t(\mathbf{x}; \mathbf{y})$ is rigid relative to $sidecon_t(\mathbf{x})$, and $sidecon_t(\mathbf{x})$ is safe for $t(\mathbf{x}; \mathbf{y})$.*

PROOF. We will construct a sequence of side-conditions $\Phi_0(\mathbf{x}), \ldots, \Phi_n(\mathbf{x})$ safe for $t(\mathbf{x}; \mathbf{y})$, such that each Φ_{i+1} logically strictly implies Φ_i, and such that $t(\mathbf{x}; \mathbf{y})$ is rigid relative to $\Phi_n(\mathbf{x})$.

For $\Phi_0(\mathbf{x})$ we pick the tautology $\top$, which is trivially safe for $t(\mathbf{x}; \mathbf{y})$.

Suppose that $t(\mathbf{x}; \mathbf{y})$ is not rigid relative to $\Phi_i(\mathbf{x})$, for some $i \geq 0$. By definition, this means that there are two sequences of constants $\mathbf{a}, \mathbf{a}'$ satisfying $\Phi_i(\mathbf{a})$ and $\Phi_i(\mathbf{a}')$ and two sequences of distinct nulls $\mathbf{N}, \mathbf{N}'$, such that $t(\mathbf{a}; \mathbf{N})$ and $t(\mathbf{a}'; \mathbf{N}')$ are copies of each other, but $\mathbf{a}$ and $\mathbf{a}'$ are not the same sequence, i.e., they differ in some coordinate. Let $\psi(\mathbf{x})$ be the conjunction of all formulas of the form $x_i < x_j$ or $x_i = x_j$ that are true under the assignment sending $\mathbf{x}$ to $\mathbf{a}$, and let $\Phi_{i+1}(\mathbf{x}) = \Phi_i(\mathbf{x}) \wedge \neg\psi(\mathbf{x})$. It is clear that Φ_{i+1} is strictly stronger than Φ_i. Moreover, Φ_{i+1} is still safe for $t(\mathbf{x}; \mathbf{y})$: consider any f-block $t(\mathbf{b}, \mathbf{M})$ of type $t(\mathbf{x}; \mathbf{y})$. Since Φ_i is safe for t, we can find a f-block $t(\mathbf{b}', \mathbf{M}')$ of type t such that $\Phi_i(\mathbf{b}')$ and the two blocks are copies of each other. If $\neg\psi(\mathbf{b}')$ holds, then in fact $\Phi_{i+1}(\mathbf{b}')$ holds, and we are done. Otherwise, we have that $t(\mathbf{b}', \mathbf{M}')$ is isomorphic to $t(\mathbf{a}, \mathbf{N})$ (via an isomorphism that sends each constant b'_i to the corresponding a_i) and the preimage of $t(\mathbf{a}', \mathbf{N}')$ under this isomorphism will be again a copy of $t(\mathbf{b}', \mathbf{M}')$ (and therefore also of $t(\mathbf{b}, \mathbf{M})$) that satisfies $\Phi_i(\mathbf{b}') \wedge \neg\psi(\mathbf{b}')$, i.e., $\Phi_{i+1}(\mathbf{b}')$. □

Example 4.5 To illustrate the construction of side-conditions, consider Example (e) of Figure 3. The only f-block type is $t(xy; z) = \{S(x, z), S(y, z)\}$. It is easy to see that $t(ba; N_1)$ and $t(ab; N_2)$ are copies of each other. Hence, the side-condition that is generated is $\neg(y < x)$, which is the same as $x \leq y$.

As another example, consider $R(x, y, z) \rightarrow \exists w(S(x, y, w) \wedge S(x, z, w))$. The only f-block type is $t(xyz; w) = \{S(x, y, w), S(x, z, w)\}$, and $t(aab; N_1)$ and $t(aba; N_2)$ are copies of each other. Hence, the condition $\neg(x = y \wedge y < z)$ is generated. It is also the case that $t(abc; N_1)$ and $t(acb; N_2)$ are copies of each other. Hence, the condition is expanded to $\neg(x = y \wedge y < z) \wedge \neg(x < y \wedge y < z)$. No other pairs of constants and nulls make $t(\mathbf{x}; \mathbf{y})$ copies of each other. Hence, $\neg(x = y \wedge y < z) \wedge \neg(x < y \wedge y < z)$ is the final side-condition. □

The proof of Proposition 4.6 shows how to compute for each f-block type a suitable side-condition. Proposition 4.7 provides a further improvement. Intuitively, it shows that we only need to consider comparisons of the form $x_i < x_j$ or $x_i = x_j$ for which x_i and x_j occur in similar attribute positions of relations inside the f-block type. Formally, given a f-block type $t(\mathbf{x}; \mathbf{y})$, and variables $x_i, x_j \in \mathbf{x}$, let us write $x_i \sim_t x_j$ if x_i and x_j occur in the same attribute position of a relation, i.e., if t contains two atomic formulas with the same relation symbol R, one having x_i as the k-th argument and one having x_j as the k-th argument (for some $k \leq arity(R)$). Let $\equiv_t$ be the equivalence relation on $\mathbf{x}$ generated by $\sim_t$. We call a side-condition $\Phi(\mathbf{x})$ $\equiv_t$*-local* if all atomic formulas in it are of the form $x_i < x_j$ or $x_i = x_j$ with $x_i \equiv_t x_j$.

Algorithm ConvertToLaconic
Input: A schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is specified by a finite set of FO$^<$ s-t tgds
Output: A laconic schema mapping $\mathcal{M}' = (\mathbf{S}, \mathbf{T}, \Sigma')$, where Σ' is specified by a finite set of FO$^<$ s-t tgds
  $\Sigma' := \emptyset$;
  Generate TYPES$_\mathcal{M}$;
  for all $t(\mathbf{x}; \mathbf{y}) \in$ TYPES$_\mathcal{M}$ **do**
    Compute precondition $precon_t(\mathbf{x})$;
    Compute side-condition $sidecon_t(\mathbf{x})$;
    Add the following FO$^<$ s-t tgd to Σ':
      $\forall\mathbf{x}(precon_t(\mathbf{x}) \wedge sidecon_t(\mathbf{x}) \rightarrow \exists\mathbf{y}. \bigwedge t(\mathbf{x}; \mathbf{y}))$
  end for;
  return $(\mathbf{S}, \mathbf{T}, \Sigma')$

Figure 4: Algorithm for constructing laconic schema mappings

Proposition 4.7 *For every f-block type $t(\mathbf{x}; \mathbf{y})$ there is a $\equiv_t$-local side condition $sidecon_t(\mathbf{x})$ such that $t(\mathbf{x}; \mathbf{y})$ is rigid relative to $sidecon_t(\mathbf{x})$, and $sidecon_t(\mathbf{x})$ is safe for $t(\mathbf{x}; \mathbf{y})$.*

The proof of Proposition 4.7 is omitted.

Generating side-conditions of a f-block type. Given a f-block type $t(\mathbf{x}; \mathbf{y})$, we first compute the equivalence classes of the equivalence relation $\equiv_t$. Next, we determine whether a side-condition is required by testing whether the f-block type is rigid with respect to the trivial side-conditions $\Phi_0(\mathbf{x}) := \top$. This test involves an exhaustive generation of sequences of constants, $\mathbf{a}$ and $\mathbf{a}'$, respecting the equivalence relation $\equiv_t$, and sequences of nulls, $\mathbf{N}$ and $\mathbf{N}'$, and testing whether $t(\mathbf{a}, \mathbf{N})$ and $t(\mathbf{a}'; \mathbf{N}')$ are copies of each other (while Const and Vars are infinite in general, for reasons of symmetry it suffices here to consider only a fixed subset of both consisting of at most as many values as the number of variables in $\mathbf{x}$ and $\mathbf{y}$ respectively). Whenever such sequences of constants and nulls are found, we construct part of the side-condition $\psi(\mathbf{x})$ as the conjunction of all formulas of the form $x_i < x_j$ or $x_i = x_j$ that are true under the assignment sending $\mathbf{x}$ to $\mathbf{a}$, with $x_i \equiv_t x_j$, and let $\Phi_1(\mathbf{x}) = \Phi_0(\mathbf{x}) \wedge \neg\psi(\mathbf{x})$. We repeat the process until we reach a side-condition $\Phi_i(\mathbf{x})$ with respect to which $t(\mathbf{x}; \mathbf{y})$ is rigid.

4.1.4 Putting things together: constructing the laconic schema mapping

Our algorithm for constructing laconic schema mappings from a given schema mapping that is specified by a finite set of FO$^<$ s-t tgds is described in Sections 4.1.1 to 4.1.3. Figure 4 summarizes the steps taken by our algorithm.

Theorem 4.2 *For every schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds, the algorithm in Figure 4 computes a laconic schema mapping $\mathcal{M}'$ specified by FO$^<$ s-t tgds that is logically equivalent to $\mathcal{M}$.*

PROOF. It is clear that $\mathcal{M}'$ is a schema mapping specified by FO$^<$ s-t tgds. In order to show that $\mathcal{M}'$ is laconic and logically

equivalent to $\mathcal{M}$, it is enough to show that, for every source instance I, the canonical universal solution J of I with respect to $\mathcal{M}'$ is a core universal solution for I with respect to $\mathcal{M}$. This follows from the following three facts:

- Every f-block of J is a copy of a f-block of the core universal solution of I. This follows from Proposition 4.4.
- Every f-block of the core universal solution of I is a copy of a f-block of J. This follows from Proposition 4.3 and Proposition 4.4, together with the safety part of Proposition 4.6.
- No two distinct f-blocks of J are copies of each other. This follows from the rigidity part of Proposition 4.6 together with the fact that $\text{TYPES}_{\mathcal{M}}$ contains no two distinct f-block types that are renamings of each other. □

The output schema mapping is in general exponentially longer than the input schema mapping $\mathcal{M}$. More precisely, Let n be the number of FO$^<$ s-t tgds, k the length of each FO$^<$ s-t tgd, and r the maximum number of variables occurring in each FO$^<$ s-t tgd. It is easy to see that $|\text{TYPES}_{\mathcal{M}}| \leq n \times 2^r$. Moreover, each f-block type $t(\mathbf{x};\mathbf{y}) \in \text{TYPES}_{\mathcal{M}}$ has length at most k, the precondition of each f-block type has size $2^{O(r \log r)} \cdot poly(n \cdot k)$ if computed using the techniques of [15], and the side condition of each f-block type has length at most $2^{O(r \log r)}$. This blowup will be discussed in more detail in Section 5.3.

Incidentally, if the side-conditions are left out, then the resulting schema mapping is still logically equivalent to the original mapping $\mathcal{M}$, but it may not be laconic. It will still satisfy a weak form of laconicity: a variant of the chase defined in [5], which only fires dependencies whose right hand side is not yet satisfied, will produce the core universal solution.

5. OPTIMALITY OF OUR ALGORITHM

In the previous sections, we have seen that our algorithm ConvertToLaconic transforms a schema mapping specified by s-t tgds into a laconic schema mapping that is specified, in general, by FO$^<$ s-t tgds which may be exponential in the size of the original schema mapping. In this section, we show that ConvertToLaconic is optimal in the sense that the use of linear order, the use of negation, and the fact that ConvertToLaconic generates a laconic schema mapping that may be exponential in the size of the original schema mapping, are essentially unavoidable.

5.1 The Use of Linear Order

The schema mapping produced by our algorithm ConvertToLaconic is specified by FO$^<$ s-t tgds that may contain comparisons of the form $x_i < x_j$, even if the input schema mapping is specified by s-t tgds only. In practice, however, the use of a linear order can be avoided in many situations. In particular, this is the case when the input schema mapping is specified by FO s-t tgds whose right-hand-sides have no self-joins, i.e., the right-hand-side of each s-t tgd is such that every atomic formula uses a different relation.

Theorem 5.1 *If the input schema mapping is specified by FO s-t tgds and the right-hand-side of each FO s-t tgd is such that every atomic formula uses a different relation, then the output of our algorithm ConvertToLaconic is a schema mapping specified by FO s-t tgds.*

This follows immediately from the fact that every f-block type in which every relation from the target schema occurs at most one is rigid. A special case of this result was obtained in [3], namely that every schema mappings specified by s-t tgds whose right hand side consists of a single atomic formula without repetition of existentially quantified variables is logically equivalent to a laconic schema mapping specified by FO s-t tgds.

It is also worth pointing out that if the source and target schema are typed, in the sense that each attribute of each relation has an associated data type and the variables respect the data types, then the schema mapping computed by our algorithm will respect the typing, in the sense that it will only involve comparisons $x < y$ for x, y of the same data type. This follows from the fact that the side-condition is $\equiv_t$-local (cf. Proposition 4.7).

In the remainder, we show that the use of the linear order cannot be avoided in general. Ideally, we would like to prove that some schema mappings specified by s-t tgds cannot be compiled into SQL queries that compute the core universal solution without the use of comparisons of the form $x < y$. However, this is difficult since SQL is a complex language and we don't have a rigorous definition of what it means to compute a solution with nulls by means of SQL queries. Therefore, we will first introduce the notion of FO-term interpretations and FO$^<$-term interpretations, which capture the essence of the way in which we have used SQL queries for computing solutions with nulls in Section 3. We will then show that there is a schema mapping $\mathcal{M}$ specified by LAV s-t tgds, for which there is no FO-term interpretation that computes for each source instance the core universal solution w.r.t. $\mathcal{M}$.

A formal account of what we did in Section 3, abstracting away from the specifics of string concatenation, is as follows. Fix a countably infinite set of function symbols of arity n, for each $n \geq 0$. For any set X, denote by $\underline{\mathsf{Terms}}[X]$ be the set of all *terms* built up from elements of X using these function symbols, and denote by $\underline{\mathsf{PTerms}}[X] \subseteq \underline{\mathsf{Terms}}[X]$ the set of all *proper terms*, i.e., those with at least one occurrence of a function symbol. For instance, if g is a unary function and h is a binary function, then $h(g(x), y)$, $g(x)$ and x belong to $\underline{\mathsf{Terms}}[\{x, y\}]$, but only the first two belong to $\underline{\mathsf{PTerms}}[\{x, y\}]$. It is important to distinguish between proper terms, which contain at least one function symbol, and terms that consist of just a single constant, as only the former will be treated as nulls. More precisely, we assume that $\underline{\mathsf{PTerms}}[\underline{\mathsf{Const}}] \subseteq \underline{\mathsf{Vars}}$. Recall that $\underline{\mathsf{Const}} \cap \underline{\mathsf{Vars}} = \emptyset$.

Definition 5.1 An *FO-term interpretation* Π is a map assigning to each k-ary relation symbol $R \in \mathbf{T}$ a union of expressions of the form shown in equation (1), where $t_1, \ldots, t_k \in \underline{\mathsf{Terms}}[\mathbf{x}]$, $t'_1, \ldots, t'_k \in \underline{\mathsf{Terms}}[\mathbf{x}']$, and $\phi(\mathbf{x}), \ldots, \phi'(\mathbf{x}')$ are FO-queries over $\mathbf{S}$. *FO$^<$-term interpretations* are defined in the same way, using FO$^<$-queries. □

Given a source instance I, an FO$^<$-term interpretation Π generates an target instance $\Pi(I)$, in the obvious way. Note that $\Pi(I)$ may contain constants as well as nulls. Although the program specifies exactly which nulls are generated, we will consider $\Pi(I)$ only up to isomorphism, and hence the meaning of an FO$^<$-term interpretation does not depend on which function symbols it uses.

FO$^<$-term interpretations and FO-term interpretations provide a convenient level of abstraction. They capture the essence of the SQL queries computing canonical universal solutions described in Section 3 (with and without the use of comparisons), while abstracting away from the specifics of creating labeled nulls by means of string concatenation. The translation described in Section 3, when stated in terms of FO$^<$-term interpretations, gives us:

Proposition 5.1 *For every schema mapping specified by FO (resp. FO$^<$) s-t tgds, there is an FO- (resp. FO$^<$-)term interpreta-*

tion that yields, for each source instance I, the canonical universal solution for I.

The following result shows the importance of having a linear order for our approach to core computation: no FO-term interpretation can be used to compute the core universal solution in general.

Theorem 5.2 *Consider the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S} = \{R\}$, $\mathbf{T} = \{S\}$ and Σ consists of the single LAV s-t tgd $R(x, y) \rightarrow \exists z(S(x, z) \wedge S(y, z))$. (This is Example 3(e) of Figure 3.) No FO-term interpretation yields, for each source instance I, the core universal solution of I w.r.t $\mathcal{M}$.*

PROOF. The argument uses the fact that FO formulas are invariant for automorphisms. Let I be the source instance whose domain consists of the constants a, b, c, d, and such that R is the total relation over this domain. Note that every permutation of the domain is an automorphism of I. Suppose for the sake of contradiction that there is an FO-term interpretation Π such that the $\Pi(I)$ is the core universal solution of I. Then the domain of $\Pi(I)$ consists of the constants a, b, c, d and a distinct null term, call it $N_{\{x,y\}} \in \underline{\mathsf{PTerms}}[\mathbf{x}]$, for each pair of distinct constants $x, y \in \{a, b, c, d\}$, and $\Pi(I)$ contains the facts $R(x, N_{\{x,y\}})$ and $R(y, N_{\{x,y\}})$ for each of these nulls $N_{\{x,y\}}$. Now consider the term $N_{\{a,b\}}$. We can distinguish two cases. The first case is where the term $N_{\{a,b\}}$ does not contain any constants as arguments. In this case, it follows from the invariance of FO formulas for automorphisms that $\Pi(I)$ contains $R(x, N_{\{a,b\}})$ for every $x \in \{a, b, c, d\}$, which is clearly not true. The second case is where $N_{\{a,b\}}$ contains at least one constant as an argument. If $N_{\{a,b\}}$ contains the constant a or b then let t' be obtained by switching all occurrences of a and b in $N_{\{a,b\}}$, otherwise let t' be obtained by switching all occurrences of c and d in $N_{\{a,b\}}$. Either way, we obtain that there is a second null, namely t', which is distinct from $N_{\{a,b\}}$, and which stands in exactly the same relations to a and b as $N_{\{a,b\}}$ does. This again contradicts our assumption that J is the core universal solution of I. □

Corollary 5.1 *Let $\mathcal{M}$ be the schema mapping from Theorem 5.2. There is no schema mapping $\mathcal{M}'$ specified by FO s-t tgds, such that for very source instance I, the canonical universal solution of I w.r.t. $\mathcal{M}'$ is the core universal solution of I w.r.t. $\mathcal{M}$.*

Hence, the use of a linear order in the equivalent laconic schema mapping given in Figure 1(e′) is unavoidable.

Incidentally, there appears to be a close connection with the copy-elimination problem for query languages involving object creation, cf. [4].

5.2 The Use of Negation

In the previous section, we have shown that the use of linear order is essentially unavoidable, even for an input schema mapping that is specified by LAV s-t tgds. At the same time, we have shown that every schema mapping specified by FO$^<$ s-t tgds has an equivalent laconic schema mapping specified in the same language. A natural question is therefore whether one needs the full expressive power of FO$^<$ in specifying an equivalent laconic schema mapping of an input schema mapping that is given by s-t tgds.

Let BCCQ$^<$ denote the class of Boolean combinations of conjunctive queries that may refer to the linear order. The next theorem tells us the general "shape" of the output dependencies, if the input schema mapping is specified by s-t tgds. The subsequent proposition shows that the result is rather tight.

Theorem 5.3 *If $\mathcal{M}$ is specified by s-t tgds, then the output of our algorithm ConvertToLaconic is a schema mapping specified by BCCQ$^<$ s-t tgds.*

PROOF HINT. Close inspection of the translation. □

Proposition 5.2 *For the schema mapping given in Figure 3(c), there is no UCQ$^<$-term interpretation that computes the core universal solutions for each source instance.*

PROOF SKETCH. Pick constants $a < b$. Let I be the source instance containing only the fact $P(a)$, and let I' be the source instance containing the facts $P(a)$ and $R(a, b)$. Clearly, $I \subseteq I'$ and hence, for every union of conjunctive queries q, the answers to q in I' will include the answers to q in I. The core universal solution of I consists of the f-block $\{S(a, N)\}$ for some null N. It follows that the UCQ$^<$-term interpretation must map S to something that generates a tuple (a, t), with t a term. It follows quite easily from the definition of UCQ$^<$-term interpretations and the monotonicity of unions of conjunctive queries that the same tuple will be generated for the relation S in the case of source instance I'. But this contradicts the fact that the the relation S contains only tuples of constants in the core universal solution of I'. □

5.3 Exponential blowup

Our transformation involves a exponential blowup: the length of the specification of the output schema mapping is in general bounded by a single exponential in the length of the specification of the input schema mapping. This exponential blowup cannot be avoided:

Theorem 5.4 *There is a sequence of schema mappings $\mathcal{M}_1, \mathcal{M}_2, \ldots$ specified by LAV s-t tgds such that the specification of each $\mathcal{M}_k$ is of length $O(k)$, and such that every laconic schema mapping logically equivalent to $\mathcal{M}_k$ specified by FO$^<$ s-t tgds contains at least 2^k many FO$^<$ s-t tgds.*

PROOF SKETCH. Consider again the schema mapping from Example 4.2 (which is parametrized by a natural number k). As we pointed out earlier, the number of f-block types realized in core solutions is exponential in k. Now consider any logically equivalent laconic schema mapping $\mathcal{M}'$ specified by a finite set of FO$^<$ s-t tgds Σ. We may assume without loss of generality that the FO$^<$ s-t tgds in Σ are non-decomposable and that their left-hand-sides are satisfiable. Next, it is easy to see that, if the right-hand-side of one of the FO$^<$ s-t tgds contains two different universally quantified variables, x, x' the left-hand-side must logically imply $x = x'$: if not, then, since the FO$^<$ s-t tgd in question is non-decomposable, there would be source instances for which the canonical universal solution with respect to $\mathcal{M}'$ has a block containing at least two nulls, which contradicts the laconicity of $\mathcal{M}'$. Hence, we may assume that the right-hand-side of each FO$^<$ s-t tgd contains only a single universally quantified variable. Finally, the right-hand-side of each FO$^<$ s-t tgd has to describe a specific fact block, in order for the schema mapping to be laconic. All in all, this shows that $\mathcal{M}$ needs to have as many FO$^<$ s-t tgds as there are f-block types, which is exponential in k. □

In some cases, the exponential blowup can be avoided. In particular, this is the case when the number of conjuncts in the right-hand-sides of the dependencies of the input schema mapping is bounded. A close inspection of the proof of our main result shows:

Theorem 5.5 *Fix any $k \geq 0$. Then for every input schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds in which the number of atomic formulas in the right-hand side of each FO$^<$ s-t tgd is at most k, the output schema mapping is of size polynomial in the size of $\mathcal{M}$.*

6. TARGET CONSTRAINTS

In this section we consider schema mappings with target constraints and we address the question whether our main result can be extended to this setting. The answer will be negative. However, first we need to revisit our basic notions, as some subtle issues arise in the case with target dependencies.

There are two types of target constraints that are generally considered in the data exchange framework [5]: target tgds and target equality generating dependencies *(egds)*. Target tgds are simply tgds where both left-hand-side and right-hand-side are conjunction of atomic formulas over the target schema. Target egds are of the form $\forall \mathbf{x}(\phi(\mathbf{x}) \rightarrow x_1 = x_2)$, where x_1 and x_2 are distinct variables among $\mathbf{x}$, and $\phi(\mathbf{x})$ is a conjunction of atomic formulas over the target schema.

It is clear that we cannot expect to compute core universal solutions for schema mappings with target dependencies by means of FO$^<$-term interpretations. Even for the simple schema mapping defined by the s-t tgd $R(x,y) \rightarrow R'(x,y)$ and the full target tgd $R'(x,y) \wedge R'(y,z) \rightarrow R'(x,z)$ computing the core universal solution (or any other universal solution) means computing the transitive closure of R, which we know cannot be done in FO logic even on finite ordered structures. Still, we can define a notion of laconicity for schema mappings with target dependencies. Let $\mathcal{M}$ be any schema mapping specified by a finite set of FO$^<$ s-t tgds Σ_{st} and a finite set of target tgds and target egds Σ_t, and let I be a source instance. We define the *canonical universal solution of I with respect to $\mathcal{M}$* as the target instance (if it exists) obtained by taking the canonical universal solution of I with respect to Σ_{st} and chasing it with the target dependencies Σ_t. We assume a standard chase but will not make any assumptions on the chase order. Laconicity is now defined as before: a schema mapping is laconic if for each source instance, the canonical universal solution coincides with the core universal solution.

Recall that, according our main result, every schema mapping $\mathcal{M}$ specified by FO$^<$ s-t tgds is logically equivalent to a laconic schema mapping $\mathcal{M}'$ specified by FO$^<$ s-t tgds. In particular, this implies that, for each source instance I, the core universal solution for I with respect to $\mathcal{M}$ is the canonical universal solution for I with respect to $\mathcal{M}'$. The following theorem shows that no such result holds for input schema mappings specified by LAV s-t tgds and full target tgds. The proof has been omitted due to space constraints. We conjecture that, by similar arguments, an analogous negative result can be obtained for schema mappings specified by LAV s-t tgds and target egds.

Theorem 6.1 *There is a schema mapping $\mathcal{M}$ specified by finitely many LAV s-t tgds and full target tgds, for which there is no schema mapping $\mathcal{M}'$ specified of FO$^<$ tgds, target tgds and target egds, such that for every source instance I, the canonical universal solution of I with respect to $\mathcal{M}'$ is the core universal solution of I with respect to $\mathcal{M}$.*

7. CONCLUSION

We presented an algorithm for transforming a schema mapping specified by FO$^<$ s-t tgds to an equivalent laconic schema mapping specified by FO$^<$ s-t tgds. Laconic schema mappings have the advantage that they can be easily compiled into SQL queries, which, when executed on any source instance, will generate the core universal solution of that source instance. Our method paves the way for leveraging existing DBMS technology to directly produce the core universal solution in data exchange systems, such as Clio.

Since it requires evaluating exponentially many SQL queries over the source database, it may seem that our method for computing core universal solutions is rather inefficient. However, we remark that other approaches face similar difficulties. Indeed, all known approaches to computing core universal solutions that we are aware of involve repeatedly testing for the existence of a homomorphism, which is a task that is on a par with evaluating a conjunctive query. Furthermore, the number of such homomorphism tests depends on the size of the source database, and is therefore not bounded by a function of the schema mapping. It would thus be interesting to conduct an experimental evaluation in order to determine which approach is more efficient.

We showed that our results are optimal, i.e., the use of the linear order and negation are unavoidable. Furthermore, our method cannot be extended to schema mappings with target constraints. We expect that the restricted (linear) form of recursion offered by SQL:99 will not help either in extending or simplifying our results.

Acknowledgements. This work was carried out during a visit of Ten Cate to UC Santa Cruz and IBM Almaden. Ten Cate is partially supported by NWO grant 639.021.508 and by ERC Advanced Grant Webdam on Foundation of Web data management. Kolaitis is partially supported by NSF grant IIS-0430994. Tan is partially supported by NSF CAREER Award IIS-0347065 and NSF grant IIS-0430994.

8. REFERENCES

[1] A. Bonifati, E. Q. Chang, T. Ho, L. V. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB (demo)*, pages 1267–1270, 2006.

[2] A. K. Chandra and P. M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In *STOC*, pages 77–90, 1977.

[3] L. Chiticariu. Computing the Core in Data Exchange: Algorithmic Issues. *MS project report*. CS Dept., UCSC, 2005.

[4] J. V. den Bussche and D. V. Gucht. A semideterministic approach to object creation and nondeterminism in database queries. *J. Comput. Syst. Sci.*, 54(1):34–47, 1997.

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

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

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

[8] A. Fuxman, M. A. Hernández, H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested Mappings: Schema Mapping Reloaded. In *VLDB*, pages 67–78, 2006.

[9] G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *JACM*, 55(2):1–49, 2008.

[10] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *SIGMOD*, pages 805–810, 2005.

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

[12] G. Mecca, P. Papotti, and S. Raunich. Core schema mappings. In *SIGMOD*, 2009.

[13] R. Pottinger and A. Halevy. MiniCon: A Scalable Algorithm for Answering Queries using Views. *VLDB Journal*, 10(2-3):182–198, 2001.

[14] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum. EXPRESS: A Data EXtraction, Processing, amd REStructuring System. *TODS*, 2(2):134–174, 1977.

[15] B. ten Cate and P. G. Kolaitis. Structural Characterizations of Schema Mapping Languages. In *ICDT*, 2009.