Normalization and Optimization of Schema Mappings[*]

Georg Gottlob[†]
Oxford University
Oxford OX1 3QD,UK
georg.gottlob@
comlab.ox.ac.uk

Reinhard Pichler
Technische Universität Wien
A-1040 Vienna, Austria
pichler@
dbai.tuwien.ac.at

Vadim Savenkov
Technische Universität Wien
A-1040 Vienna, Austria
savenkov@
dbai.tuwien.ac.at

ABSTRACT

Schema mappings are high-level specifications that describe the relationship between two database schemas. They are an important tool in several areas of database research, notably in data integration and data exchange. However, a concrete theory of schema mapping optimization including the formulation of optimality criteria and the construction of algorithms for computing optimal schema mappings is completely lacking to date. The goal of this work is to fill this gap. We start by presenting a system of rewrite rules to minimize sets of source-to-target tuple-generating dependencies (st-tgds, for short). Moreover, we show that the result of this minimization is unique up to variable renaming. Hence, our optimization also yields a schema mapping normalization. By appropriately extending our rewrite rule system, we also provide a normalization of schema mappings containing equality-generating target-dependencies (egds). An important application of such a normalization is in the area of defining the semantics of query answering in data exchange, since several definitions in this area depend on the concrete syntactic representation of the st-tgds. This is, in particular, the case for queries with negated atoms and for aggregate queries. The normalization of schema mappings allows us to eliminate the effect of the concrete syntactic representation of the st-tgds from the semantics of query answering. We discuss in detail how our results can be fruitfully applied to aggregate queries.

1. INTRODUCTION

Schema mappings are high-level specifications that describe the relationship between two database schemas. They are an important tool in several areas of database research, notably in data integration [12, 18] and data exchange [9]. A schema mapping is usually given in the form $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$, indicating the two database schemas $\mathbf{S}$ and $\mathbf{T}$ plus a set Σ of dependencies. These dependencies express conditions that an instance of $\mathbf{S}$ and an instance of $\mathbf{T}$ must fulfill. In data exchange, $\mathbf{S}$ and $\mathbf{T}$ are referred to as source and target schema. The dependencies Σ specify, given a source instance (i.e., an instance of $\mathbf{S}$), what a legal target instance (i.e., an instance of $\mathbf{T}$) may look like. Similarly, in data integration, a schema mapping $\mathcal{M}$ describes the relationship between a local data source and a global mediated schema.

Over the past years, schema mappings have been extensively studied (see [16, 6] for numerous pointers to the literature). However, only recently, the question of *schema mapping optimization* has been raised. In [10], the foundation for optimization has been laid by defining various forms of equivalence of schema mappings and by proving important properties of the resulting notions. However, a concrete theory of schema mapping optimization including the formulation of optimality criteria and the construction of algorithms for computing optimal schema mappings is completely lacking to date. The goal of this work is to fill this gap. Below, we illustrate the basic ideas of our approach by a series of simple examples, where it is clear "at a glance" what the optimal form of the schema mappings should look like. In fact, one would expect that a human user designs these mappings in their optimal form right from the beginning. However, as more and more progress is made in the area of automatic generation and processing of schema mappings [6, 5] we shall have to deal with schema mappings of ever increasing complexity. The optimality of these automatically derived schema mappings is by no means guaranteed and schema mapping optimization will become a real necessity.

For the most common form of schema mappings considered in the literature, the dependencies in Σ are source-to-target tuple-generating dependencies (st-tgds, for short). These are first-order sentences $\forall \vec{x} \, (\varphi(\vec{x}\,) \rightarrow \exists \vec{y} \; \psi(\vec{x}, \vec{y}\,))$, where the antecedent φ is a conjunctive query (CQ) over $\mathbf{S}$ and the conclusion ψ is a CQ over $\mathbf{T}$. The universal quantification is usually not denoted explicitly. Instead, it is assumed implicitly for all variables in $\varphi(\vec{x}\,)$.

EXAMPLE 1.1. Consider a schema mapping $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ with $\mathbf{S} = \{L(\cdot,\cdot,\cdot), P(\cdot,\cdot)\}$ and $\mathbf{T} = \{C(\cdot,\cdot)\}$, where L, P, and C denote the relational schemas Lecture(title, year, prof), Prof(name, area), and Course (title, prof-area), respectively. Moreover, suppose that Σ consists of two rules expressing the following constraints: If there exists any lecture at all in the source instance, then the title of all lectures for 3rd year students as well as the area of the professor giving this lecture should be present in the Course-relation of the target instance. Moreover, Σ contains a specific rule which takes care of the lectures given by professors from the database area. We

[*]R. Pichler and V. Savenkov were supported by the Vienna Science and Technology Fund (WWTF), project ICT08-032. V. Savenkov receives a scholarship from the European program "Erasmus Mundus External Cooperation Window". Georg Gottlob's work was supported by EPSRC grant EP/E010865/1 "Schema Mappings and Automated Services for Data Integration and Exchange". G. Gottlob is the holder of a Royal Society Wolfson Research Merit Award.

[†]Computing Laboratory and Oxford-Man Institute of Quantitative Finance, University of Oxford.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '09, August 24-28, 2009, Lyon, France
Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

thus get the following set Σ of st-tgds:

$$\Sigma = \{L(x_1, x_2, x_3) \wedge L(x_4, 3, x_5) \wedge P(x_5, x_6) \to C(x_4, x_6), \\ L(x_1, 3, x_2) \wedge P(x_2, '\mathsf{db}') \to C(x_1, '\mathsf{db}')\} \qquad \Box$$

The above schema mapping has a specific form called GAV (global-as-view) [18], i.e., we only have st-tgds $\varphi(\vec{x}\,) \to A(\vec{x}\,)$, where the conclusion is a single atom $A(\vec{x}\,)$ without existentially quantified variables. In this special case, a close relationship of schema mappings with unions of conjunctive queries (UCQs) becomes apparent. Indeed, given a source instance I over $\mathbf{S}$, the tuples which have to be present in any legal target instance J according to the above schema mapping $\mathcal{M}$ are precisely the tuples which are in the result of the following UCQ:

$$ans(x_4, x_6) \text{ :- } L(x_1, x_2, x_3) \wedge L(x_4, 3, x_5) \wedge P(x_5, x_6) \\ ans(x_1, '\mathsf{db}') \text{ :- } L(x_1, 3, x_2) \wedge P(x_2, '\mathsf{db}').$$

The goal of UCQ-optimization is usually twofold [7, 21], namely to minimize the number of CQs and to minimize the number of atoms in each CQ. In the above UCQ, we would thus delete the second CQ and, moreover, eliminate the first atom from the body of the first CQ. In total, the above UCQ can be replaced by a single CQ $ans(x_4, x_6) \text{ :- } L(x_4, 3, x_5) \wedge P(x_5, x_6)$. Analogously, we would naturally reduce the set Σ of two st-tgds in Example 1.1 to the singleton $\Sigma' = \{L(x_4, 3, x_5) \wedge P(x_5, x_6) \to C(x_4, x_6)\}$.

As mentioned above, GAV mappings are only a special case of schema mappings given by st-tgds which, in the general case, may have existentially quantified variables and conjunctions of atoms in the conclusion. Note that the existentially quantified variables are used to represent incomplete data (in the form of marked nulls [14]) in the target instance. Hence, as an additional optimization goal, we would like to minimize the number of existentially quantified variables in each st-tgd. Moreover, we would now like to minimize also the atoms in the CQ of the conclusion.

EXAMPLE 1.2. We revisit Example 1.1 and consider a new mapping $\mathcal{M}$ in the reverse direction so to speak: Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ with $\mathbf{S} = \{C(\cdot, \cdot)\}$ and $\mathbf{T} = \{L(\cdot, \cdot, \cdot), P(\cdot, \cdot)\}$ where L, P, and C are as before. Moreover, let Σ be defined as follows:

$$\Sigma = \{C(x_1, x_2) \to (\exists y_1, y_2, y_3, y_4) L(y_1, y_2, y_3) \wedge \\ L(x_1, 3, y_4) \wedge P(y_4, x_2), \\ C(x_1, '\mathsf{db}') \to (\exists y_1) L(x_1, 3, y_1) \wedge P(y_1, '\mathsf{db}')\}$$

Clearly, Σ is equivalent to the singleton

$$\Sigma' = \{C(x_1, x_2) \to (\exists y_4) L(x_1, 3, y_4) \wedge P(y_4, x_2)\}. \qquad \Box$$

The above schema mapping corresponds to the special case of LAV (local-as-view) [18] with st-tgds $A(\vec{x}\,) \to \exists \vec{y}\; \psi(\vec{x}, \vec{y}\,)$, where the antecedent is a single atom $A(\vec{x}\,)$ and all variables in $A(\vec{x}\,)$ actually do occur in the conclusion. In the most general case (referred to as GLAV mappings), no restrictions are imposed on the CQs in the antecedent and conclusion nor on the variable occurrences. In order to formulate an optimality criterion for schema mappings with st-tgds of this general form, the analogy with UCQs does not suffice. Indeed, the following example illustrates that we may get a highly unsatisfactory result if we just aim at the minimization of the number of st-tgds and of the number of atoms inside each st-tgd.

EXAMPLE 1.3. Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ with $\mathbf{S} = \{L(\cdot, \cdot, \cdot)\}$ and $\mathbf{T} = \{C(\cdot, \cdot), E(\cdot, \cdot)\}$ where L, and C are as before and E denotes the relational schema Equal-Year(course1, course2), i.e., E contains pairs of courses designed for students in the same year. Moreover, let Σ be defined as follows:

$$\Sigma = \{L(x_1, x_2, x_3) \to (\exists y) C(x_1, y), \\ L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \to E(x_1, x_4)\}$$

Then Σ is equivalent to the singleton

$$\Sigma' = \{L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \to (\exists y) C(x_1, y) \wedge E(x_1, x_4)\}$$

Now suppose that the name-attribute is a key in Lecture. Let l_i denote the name of some lecture in a source instance I and suppose that I contains m lectures for students in the same year as l_i. Then the computation of the *canonical solution* (for details, see Section 2) yields two results of significantly different quality depending on whether we take Σ or Σ': In case of Σ, we get one tuple $C(l_i, y)$ with this course name l_i. In contrast, for Σ', we get m tuples $C(l_i, y_1), \ldots, C(l_i, y_m)$ with the same course name l_i. The reason for this is that the st-tgd "fires" for every possible combination of key values x_1 and x_4, even though for the first conjunct $C(x_1, y)$ of the conclusion, only the value of x_1 is relevant. $\Box$

We shall refer to the two st-tgds in Σ of the above example as the split form of the st-tgd in Σ'. We shall formally define *splitting* of st-tgds in Section 3. Intuitively, splitting aims at breaking up the conclusion of an st-tgd in smaller parts such that the variables in the antecedent are indeed related to the atoms in the conclusion. Without this measure, any target instance would be artificially inflated with marked nulls as we have seen with Σ' in the above example. Splitting helps to avoid such anomalies. Indeed, it can be seen as an analogous operation to the decomposition of relational schemas into normal form where we also want to exclude that some attributes are fully determined by parts of a key. Carrying over this idea to st-tgds, we want to exclude that some atoms in the conclusion are fully determined by parts of the atoms in the antecedent. Our first *optimization goal* for schema mappings will therefore be to minimize the number of st-tgds only to the extent that splitting should be applied whenever possible. Minimizing the size of each st-tgd and the number of existentially quantified variables in the conclusion will, of course, be pursued as another optimization goal. We thus have the following optimality criteria for sets Σ of st-tgds:

- *cardinality-minimality*, i.e., the number of st-tgds in Σ shall be minimal;
- *antecedent-minimality*, i.e., the total size of the antecedents of the st-tgds in Σ shall be minimal;
- *conclusion-minimality*, i.e., the total size of the conclusions of the st-tgds in Σ shall be minimal;
- *variable-minimality*, i.e., the total number of existentially quantified variables in the conclusions shall be minimal.

Then a set of st-tgds is *optimal*, if it is minimal w.r.t. each of these four criteria. Following the above discussion, we only take st-tgds into consideration for which no further splitting is possible. (We shall give a formal definition of this property and of the four optimality criteria in Section 3). Cardinality-minimality together with antecedent-minimality means that the *cost of the join-operations* is minimized when computing a canonical solution for some given source instance. Conclusion-minimality (resp. variable-minimality) means that no obvious redundancy (resp. incompleteness) is introduced in the canonical solution. For the transformation of arbitrary sets of st-tgds into optimal ones, we shall present a *new system of rewrite rules*. Moreover, we shall show that the optimal form of a set of st-tgds is *unique up to variable renaming*.

In other words, our optimization of schema mappings is also a *normalization of schema mappings*. As an immediate benefit of a normalization, we get a purely syntactical criterion for testing the equivalence of two schema mappings. Another, even more important application of such a normalization is in the area of defining the *semantics of query answering in data exchange*. Several definitions in this area depend on the concrete syntactic representation of the st-tgds. This is, in particular, the case for queries with negated atoms (see e.g., [2, 19]) and for aggregate queries (see [1]). Hence, it would be desirable to have a unique normal form of st-tgds on which the definition of the semantics of query answering should be based. Since the minimal set of st-tgds produced by our rewrite rules is unique up to variable renaming, we can use it as the desired normal form which eliminates the effect of the concrete representation of the st-tgds from the semantics of query answering.

EXAMPLE 1.4. Consider a schema mapping $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ with $\mathbf{S} = \{S(\cdot,\cdot,\cdot)\}$, $\mathbf{T} = \{L(\cdot,\cdot,\cdot), P(\cdot,\cdot)\}$, where L and P are as in Example 1.2. S denotes the relational schema Student(name, year, area). Moreover, let Σ express the following constraints: If there exists a student in any year, then there should exist at least one lecture for this year. Moreover, if a student specializes in a particular area, then there should be a professor in this area teaching at least one lecture for this year. We thus have the following set Σ with a single st-tgd:

$$\Sigma = \{S(x_1,x_2,x_3) \rightarrow (\exists y_1,y_2,y_3,y_4,y_5) L(y_1,x_2,y_3) \wedge L(y_4,x_2,y_5) \wedge P(y_5,x_3)\}.$$

Clearly, the first atom in the conclusion may be deleted.

Now consider the source instance $I = \{S(\text{'bob'}, 3, \text{'db'})\}$ and suppose that we want to evaluate the query

$$ans(x_2) \text{ :- } L(x_1,x_2,x_3), \neg P(x_3,x_4)$$

over the target instance, i.e., we want to check if, in some year, there exists a lecture which has not been assigned to a professor. In [2, 19], query answering via the "canonical solution" (for details, see Section 2) is proposed. Depending on whether the st-tgd in Σ has been simplified or not, we either get $J = \{L(u_1,3,u_2), L(u_3,3,u_4), P(u_4,\text{'db'})\}$ or $J' = \{L(u_1,3,u_2), P(u_2,\text{'db'})\}$ as canonical solution. In the first case, the query yields the result $\{\langle 3 \rangle\}$ whereas, in the second case, we get $\emptyset$. □

Similarly, a unique normal form of the st-tgds is crucial for the semantics of aggregate queries in data exchange, whose investigation has been initiated recently by Afrati and Kolaitis [1]. Aggregate queries are of the form SELECT f FROM R, where f is an aggregate operator min$(R.A)$, max$(R.A)$, count$(R.A)$, count$(*)$, sum$(R.A)$, or avg$(R.A)$, and where R is a target relation symbol or, more generally, a conjunctive query over the target schema and A is an attribute of R. The main contribution in [1] was twofold. On the one hand, Afrati and Kolaitis defined an interesting and non-trivial semantics of aggregate queries in data exchange. On the other hand, they showed that the most important aggregate queries can be evaluated in polynomial time (data complexity). In particular for the avg$(R.A)$ operator, this result is highly non-trivial. In this paper, we shall show how our normalization of schema mappings can be fruitfully applied to aggregate queries. Moreover, we shall extend the tractability results from [1] to more expressive schema mappings, including also equality-generating target-dependencies (egds) to be discussed next.

So far, we have only mentioned mappings $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$, where Σ is a set of st-tgds. In addition, Σ may contain constraints on the target instance alone. The most important form of target constraints are equality-generating target-dependencies (egds), which can be considered as a generalization of key dependencies. Egds are formulae of the form $\forall \vec{x}\, (\varphi(\vec{x}) \rightarrow x_i = x_j)$ where φ is a CQ over $\mathbf{T}$ and x_i, x_j are variables in $\vec{x}$.

EXAMPLE 1.5. We modify the setting from Example 1.1 and 1.2. Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma \rangle$ with $\mathbf{S} = \{C(\cdot,\cdot,\cdot)\}$ and $\mathbf{T} = \{P(\cdot,\cdot,\cdot)\}$ where C and P denote the relational schemas Course (title, course-area, prof-area) and Prof(name, prof-area, course-area). The P-relation thus contains information on the area of the professor as well as on the area(s) of the courses taught by him/her. The set Σ of st-tgds expresses the following constraints: For every course, there exists a professor who teaches courses in his/her own area of expertise and who teaches courses with this combination of course- and prof-area. Moreover, there exists a professor whose expertise matches the area of the course and vice versa. We thus define Σ as

$$\Sigma = \{C(x_1,x_2,x_3) \rightarrow (\exists y_1,y_2)\, P(y_1,y_2,y_2) \wedge P(y_1,x_2,x_3), \quad C(x_1,x_2,x_3) \rightarrow (\exists y_1) P(y_1,x_3,x_2)\}$$

This set of dependencies is minimal. However, suppose that we add the egd $P(x_1,x_2,x_3) \rightarrow x_2 = x_3$, expressing that a professor only teaches courses in his/her own area of expertise. Then $P(y_1,y_2,y_2)$ can be eliminated from the conclusion of the first st-tgd. Moreover, the first and the second st-tgd imply each other. Hence, Σ can be replaced by either Σ' or Σ'' with

$$\Sigma' = \{C(x_1,x_2,x_3) \rightarrow (\exists y_1)\, P(y_1,x_2,x_3)\} \text{ and}$$
$$\Sigma'' = \{C(x_1,x_2,x_3) \rightarrow (\exists y_1)\, P(y_1,x_3,x_2)\}.$$ □

Example 1.5 illustrates that, in the presence of target egds, our rewrite rules for the st-tgds-only case are not powerful enough. In order to deal with target egds, we will introduce a collection of further rewrite rules. In particular, one of these new rewrite rules will result in the introduction of source egds to rule out situations where two sets of st-tgds only differ on source instances which admit no target instance anyway. Indeed, in Example 1.5, Σ' and Σ'' only differ if $x_2 \neq x_3$ holds. But this is forbidden by the egd. Hence, Σ should be replaced by Σ^* with

$$\Sigma^* = \{C(x_1,x_2,x_3) \rightarrow x_2 = x_3, \quad C(x_1,x_2,x_2) \rightarrow (\exists y_1)\, P(y_1,x_2,x_2)\}.$$

In summary, we shall be able to prove that our extended set of rewrite rules again leads to a *normal form* which is *unique up to variable renaming*.

Organization of the paper and summary of results. In Section 2, we recall some basic notions. A conclusion and an outlook to future work are given in Section 7. The main results of the paper are detailed in the Sections 3 – 6, namely:

- *Optimization and normalization of sets of st-tgds.* In Section 3, we give a formal definition of the above mentioned optimality criteria for sets of st-tgds and we present rewrite rules to transform any set of st-tgds into an optimal one (i.e., minimal w.r.t. to these criteria). We shall also show that the normal form obtained by our rewrite rules is unique up to variable renaming. Moreover, we show that, if the length of each st-tgd is bounded by a constant, then this normal form can be computed in polynomial time.

- *Extension to target egds.* In Section 4, the rewrite rule system for st-tgds is then extended to schema mappings comprising target egds. Several non-trivial extensions (like the introduction of source egds) are required to arrive at a unique normal form again.

- *Semantics and evaluation of aggregate operators.* In Section 5, we discuss in detail the application of our normalization of schema mappings to the definition of a unique semantics of aggregate operators in data exchange. Moreover, we extend the tractability of aggregate queries from [1] to target egds.

- *Implementation.* In Section 6, we report on first experimental results with a prototype implementation. It is freely accessible as a web tool, see `www.dbai.tuwien.ac.at/proj/sm`.

Due to lack of space, most proofs are sketched. Full proofs of all results of Sections 3 – 5 are provided in the full paper.

2. PRELIMINARIES

A *schema* $\mathbf{R} = \{R_1, \ldots, R_n\}$ is a set of relation symbols R_i each of a fixed arity. An *instance* over a schema $\mathbf{R}$ consists of a relation for each relation symbol in $\mathbf{R}$, s.t. both have the same arity. We only consider finite instances here.

Tuples of the relations may contain two types of *terms*: *constants* and *variables*. The latter are also called *marked nulls* or *labeled nulls*. Two labeled nulls are equal iff they have the same label. For every instance J, we write $dom(J)$, $var(J)$, and $Const(J)$ to denote the set of terms, variables, and constants, respectively, of J. Clearly, $dom(J) = var(J) \cup Const(J)$ and $var(J) \cap Const(J) = \emptyset$. If we have no particular instance J in mind, we write $Const$ to denote the set of all possible constants. We write $\vec{x}$ for a tuple $(x_1, x_2, \ldots, x_n)$. However, by slight abuse of notation, we also refer to the set $\{x_1, \ldots, x_n\}$ as $\vec{x}$. Hence, we may use expressions like $x_i \in \vec{x}$ or $\vec{x} \subseteq X$, etc.

Let $\mathbf{S} = \{S_1, \ldots, S_n\}$ and $\mathbf{T} = \{T_1, \ldots, T_m\}$ be schemas with no relation symbols in common. We call $\mathbf{S}$ the *source schema* and $\mathbf{T}$ the *target schema*. We write $\langle \mathbf{S}, \mathbf{T} \rangle$ to denote the schema $\{S_1, \ldots, S_n, T_1, \ldots, T_m\}$. Instances over $\mathbf{S}$ (resp. $\mathbf{T}$) are called *source* (resp. *target*) *instances*. If I is a source instance and J a target instance, then $\langle I, J \rangle$ is an instance of the schema $\langle \mathbf{S}, \mathbf{T} \rangle$.

Homomorphisms and substitutions. Let I, I' be instances. A *homomorphism* $h\colon I \rightarrow I'$ is a mapping $dom(I) \rightarrow dom(I')$, s.t. (1) whenever $R(\vec{x}) \in I$, then $R(h(\vec{x})) \in I'$, and (2) for every constant c, $h(c) = c$. If such h exists, we write $I \rightarrow I'$. Moreover, if $I \leftrightarrow I'$ then we say that I and I' are *homomorphically equivalent*. In contrast, if $I \rightarrow I'$ but not vice versa, we say that I is *more general* than I', and I' is *more specific* than I.

If $h\colon I \rightarrow I'$ is invertible, s.t. h^{-1} is a homomorphism from I' to I, then h is called an *isomorphism*, denoted $I \cong I'$. An *endomorphism* is a homomorphism $I \rightarrow I$. An endomorphism is *proper* if it is not surjective (for finite instances, this is equivalent to being not injective), i.e., if it reduces the domain of I.

If I is an instance, and $I' \subseteq I$ is such that $I \rightarrow I'$ holds but for no other $I'' \subset I'$: $I \rightarrow I''$ (that is, I' cannot be further "shrunk" by a proper endomorphism), then I' is called a *core* of I. The core is unique up to isomorphism. Hence, we may speak about *the* core of I. Cores have the following important property: for arbitrary instances J and J', $J \leftrightarrow J'$ *iff* $core(J) \cong core(J')$.

A *substitution* σ is a mapping which sends variables to other domain elements (i.e., variables or constants). We write $\sigma = \{x_1 \leftarrow a_1, \ldots, x_n \leftarrow a_n\}$ if σ maps each x_i to a_i and σ is the identity outside $\{x_1, \ldots, x_n\}$. The application of a substitution is usually denoted in postfix notation, e.g.: $x\sigma$ denotes the image of x under σ. For an expression $\varphi(\vec{x})$ (e.g., a conjunctive query with variables in $\vec{x}$), we write $\varphi(\vec{x}\sigma)$ to denote the result of replacing every occurrence of every variable $x \in \vec{x}$ by $x\sigma$.

Schema Mappings and Data Exchange. A *schema mapping* is given by a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where $\mathbf{S}$ is the source schema, $\mathbf{T}$ is the target schema, and Σ is a set of dependencies expressing the relationship between $\mathbf{S}$ and $\mathbf{T}$ and possibly also local constraints on $\mathbf{S}$ resp. $\mathbf{T}$. The *data exchange problem* associated with $\mathcal{M}$ is the following: Given a (ground) source instance I, find a target instance J, s.t. $\langle I, J \rangle \models \Sigma$. Such a J is called a *solution for I* or, simply, a *solution* if I is clear from the context. The set of all solutions for I under $\mathcal{M}$ is denoted by $Sol^{\mathcal{M}}(I)$. If $J \in Sol^{\mathcal{M}}(I)$ is such that $J \rightarrow J'$ holds for any other solution $J' \in Sol^{\mathcal{M}}(I)$, then J is called a *universal solution*. Since the universal solutions for a source instance I are homomorphically equivalent, the core of the universal solutions for I is unique up to isomorphism. It is the smallest universal solution [11].

In the following, we will often identify a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ with the set of dependencies Σ, without explicitly mentioning the schemas, for the sake of brevity.

Equivalence of schema mappings. Different notions of equivalence of schema mappings have been recently proposed by Fagin et al. [10]. In this paper, we will only consider the strongest one, namely logical equivalence.

DEFINITION 2.1. *[10] Two schema mappings Σ and Σ' over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$ are logically equivalent (denoted as $\Sigma \equiv \Sigma'$) if, for every source instance I and target instance J, the equivalence $\langle I, J \rangle \models \Sigma \Leftrightarrow \langle I, J \rangle \models \Sigma'$ holds. In this case, the equality $Sol^{\Sigma}(I) = Sol^{\Sigma'}(I)$ holds for every source instance I.*

Embedded dependencies. *Embedded dependencies* [8] over a relational schema $\mathbf{R}$ are first-order formulae of the form $\forall \vec{x} \big(\varphi(\vec{x}\,) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})\big)$. In case of *tuple-generating dependencies* (tgds), both *antecedent* φ and *conclusion* ψ are conjunctive queries (CQs) over the relation symbols from $\mathbf{R}$ s.t. all variables in $\vec{x}$ actually do occur in $\varphi(\vec{x}\,)$. *Equality-generating dependencies* (egds) are of the form $\forall \vec{x}\, (\varphi(\vec{x}\,) \rightarrow x_i = x_j)$ with $x_i, x_j \in \vec{x}$. Throughout this paper, we shall omit the universal quantifiers: By convention, all variables occurring in the antecedent are universally quantified (over the entire formula). For a conjunctive query χ (in the antecedent or the conclusion of some dependency), we write $At(\chi)$ to denote the set of atoms of this CQ.

In the context of data exchange, we are mainly dealing with *source-to-target dependencies* consisting of tuple-generating dependencies (st-tgds) over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$ (the antecedent is a CQ over $\mathbf{S}$, the conclusion over $\mathbf{T}$) and *target dependencies* over $\mathbf{T}$. In the scope of this paper, target dependencies are restricted to equality-generating dependencies (referred to as "target egds"). Moreover, in Section 4, we shall also consider *source dependencies* consisting of egds over $\mathbf{S}$ (referred to as "source egds").

Dependency databases. For an embedded dependency τ with antecedent $\varphi(\vec{x})$, the *antecedent database* of τ is the set of all atoms in $\varphi(\vec{x}\lambda)$, where λ is a valuation assigning a fresh labeled null to each variable of $\vec{x}$. If τ is a tgd with conclusion $\varphi(\vec{x}, \vec{y})$, then its *conclusion database* is the set of atoms in $\psi(\vec{x}\lambda, \vec{y}\lambda)$ with $\vec{x}$ and $\vec{y}$ sent to fresh labeled nulls by λ. We call an antecedent (resp. conclusion) database *frozen*, if instead of nulls, λ assigns fresh distinct constants to the variables of the antecedent (resp. conclusion) of τ.

Chase. The data exchange problem can be solved by the *chase* [4], a sequence of steps, each enforcing a single constraint within some limited set of tuples. More precisely, let Σ contain a tgd $\tau\colon \varphi(\vec{x}) \rightarrow (\exists \vec{y})\psi(\vec{x}, \vec{y})$, s.t. $I \models \varphi(\vec{a}\,)$ for some assignment $\vec{a}$ on $\vec{x}$. Then we extend I with facts corresponding to $\psi(\vec{a}, \vec{z}\,)$, where the elements of $\vec{z}$ are fresh labeled nulls. Note that this definition of the chase differs from the definition in [9], where no new facts are added if $I \models \exists \vec{y} \psi(\vec{a}, \vec{y}\,)$ is already fulfilled. Omitting this check is referred to as *oblivious* [15] chase. It is the preferred version of chase if the result of the chase should not depend on the order in which the tgds are applied (see e.g., [2, 19, 1]).

Now suppose that Σ contains an egd $\varepsilon\colon \varphi(\vec{x}\,) \rightarrow x_i = x_j$, s.t. $I \models \varphi(\vec{a}\,)$ for some assignment $\vec{a}$ on $\vec{x}$. This egd enforces the equality $a_i = a_j$. We thus choose a null a' among $\{a_i, a_j\}$ and replace *every occurrence* of a' in I by the other term; if $a_i, a_j \in Const(I)$ and $a_i \neq a_j$, the chase halts with *failure*. We write I^{Σ} to denote the result of chasing I with the dependencies Σ.

Consider an arbitrary schema mapping $\Sigma = \Sigma_{st} \cup \Sigma_t$ where Σ_{st} is a set of source-to-target tgds and Σ_t is a set of target egds. Then the solution to a source instance I can be computed as follows: We start off with the instance $\langle I, \emptyset \rangle$, i.e., the source instance is I and the target instance is initially empty. Chasing $\langle I, \emptyset\rangle$ with Σ_{st} yields the instance $\langle I, J\rangle$, where J is called the *preuniversal instance*. This chase always succeeds since Σ_{st} contains no egds. Then J is chased with Σ_t. This chase may fail on an attempt to unify distinct constants. If the chase succeeds, we end up with $U = J^{\Sigma_t}$, which is referred to as the *canonical universal solution* $CanSol^{\Sigma}(I)$ or, simply $CanSol(I)$. Both J and U can be computed in polynomial time w.r.t. the size of the source instance [9].

3. NORMALIZATION OF ST-TGDS

In this section, we investigate ways of optimizing sets of st-tgds. In the first place, we thus formulate some natural optimality criteria. The following parameters of a set of st-tgds will be needed in the definition of such criteria:

DEFINITION 3.1. *Let Υ be a set of st-tgds. Then we define:*

- $|\Upsilon|$ *denotes the number of st-tgds in* Υ.
- $AntSize(\Upsilon) = \Sigma\{|At(\varphi(\vec{x}))| \colon \forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x},\vec{y}))$ *is an st-tgd in* $\Upsilon\}$*, i.e.,* $AntSize(\Upsilon)$ *is the total number of atoms in all antecedents of st-tgds in* Υ.
- $ConSize(\Upsilon) = \Sigma\{|At(\psi(\vec{x},\vec{y}))| \colon \forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x}, \vec{y}))$ *is an st-tgd in* $\Upsilon\}$*, i.e.,* $ConSize(\Upsilon)$ *is the total number of atoms in all conclusions of st-tgds in* Υ.
- $VarSize(\Upsilon) = \Sigma\{|\vec{y}| \colon \forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x},\vec{y}))$ *is in* $\Upsilon\}$*, i.e.,* $VarSize(\Upsilon)$ *is the total number of existentially quantified variables in all conclusions of st-tgds in* Υ.

We would naturally like to transform any set of st-tgds into an equivalent one where all the above parameters are minimal. Recall however our discussion on the splitting of st-tgds from Example 1.3. As we pointed out there, the splitting of st-tgds is comparable to normal form decomposition of relational schemas. It should clearly be applied in order to avoid anomalies like the introduction of obviously irrelevant atoms in the canonical solution as we saw in Example 1.3, where the set Σ (with two split st-tgds) was certainly preferable to Σ' even though $|\Sigma'| < |\Sigma|$ and $AntSize(\Sigma') < AntSize(\Sigma)$ hold. Note that $ConSize(\Sigma') = ConSize(\Sigma)$ in Example 1.3. Intuitively, the effect of splitting is that the atoms in the conclusion of some st-tgd are distributed over several strictly smaller st-tgds. Thus, our goal should be to find the optimal set of st-tgds (i.e., where the above mentioned parameters are minimal) among those sets of st-tgds for which no further splitting is possible. We now make precise what it means that "no further splitting" is possible and formally define optimality of a set of st-tgds.

DEFINITION 3.2. *Let Σ be a set of st-tgds. We say that Σ is* split-reduced *if there exists no Σ' equivalent to Σ, s.t. $|\Sigma'| > |\Sigma|$ but $ConSize(\Sigma') = ConSize(\Sigma)$.*

DEFINITION 3.3. *Let Σ be a set of st-tgds. We say that Σ is* optimal *if it is split-reduced and if each of the parameters $|\Sigma|$, $AntSize(\Sigma)$, $ConSize(\Sigma)$, and $VarSize(\Sigma)$ are minimal among all split-reduced sets Σ' equivalent to Σ.*

Of course, given an arbitrary set Σ of st-tgds, it is a priori not clear that an optimal set Σ' equivalent to Σ exists, since it might well be the case that some Σ' minimizes some of the parameters while another set Σ'' minimizes the other parameters. The goal of this section is to show that optimality in the above sense can always be achieved and to construct an algorithm which transforms any set Σ of st-tgds into an equivalent optimal one. To this end, we introduce a rewrite system which consists of two kinds of rewrite rules: rules which simplify each st-tgd individually and rules which are applied to the entire set of st-tgds. The following example illustrates several kinds of redundancy that a single st-tgd may contain (and which may be eliminated with our rewrite rules).

EXAMPLE 3.1. Consider the following dependency:

$$\begin{aligned}\tau\colon S(x_1,x_3) \wedge S(x_1,x_2) &\rightarrow (\exists y_1,y_2,y_3,y_4,y_5)\\ &P(x_1,y_2,y_1) \wedge R(y_1,y_3,x_2) \wedge R(2,y_3,x_2)\\ &\wedge P(x_1,y_4,2) \wedge P(x_1,y_4,y_5) \wedge Q(y_4,x_3)\end{aligned}$$

Clearly, τ is equivalent to the set $\{\tau_1,\tau_2\}$ of st-tgds:

$$\begin{aligned}\tau_1\colon S(x_1,x_3) \wedge S(x_1,x_2) &\rightarrow (\exists y_1,y_2,y_3)\\ &P(x_1,y_2,y_1) \wedge R(y_1,y_3,x_2) \wedge R(2,y_3,x_2)\\ \tau_2\colon S(x_1,x_3) \wedge S(x_1,x_2) &\rightarrow (\exists y_4,y_5)\\ &P(x_1,y_4,y_5) \wedge P(x_1,y_4,2) \wedge Q(y_4,x_3)\end{aligned}$$

Now the antecedents of τ_1 and τ_2 can be simplified:

$$\begin{aligned}\tau_1'\colon S(x_1,x_2) &\rightarrow (\exists y_1,y_2,y_3)\\ &P(x_1,y_2,y_1) \wedge R(y_1,y_3,x_2) \wedge R(2,y_3,x_2)\\ \tau_2'\colon S(x_1,x_3) &\rightarrow (\exists y_4,y_5)\\ &P(x_1,y_4,y_5) \wedge P(x_1,y_4,2) \wedge Q(y_4,x_3)\end{aligned}$$

Finally, we may also simplify the conclusion of τ_2':

$\tau_2''\colon S(x_1,x_3) \rightarrow (\exists y_4)P(x_1,y_4,2) \wedge Q(y_4,x_3)$

In total, τ is equivalent to $\{\tau_1', \tau_2''\}$. □

For the simplifications illustrated in Example 3.1, we define the rewrite rules 1 – 3 in Figure 1. Rules 1 and 2 replace an st-tgd τ by a simpler one (i.e., with fewer atoms) τ', while Rules 3 replaces τ by a set $\{\tau_1,\dots,\tau_n\}$ of simpler st-tgds. These rules make use of the following definitions of the *core* and the *components* of CQs.

DEFINITION 3.4. *Let $\chi(\vec{u},\vec{v})$ be a CQ with variables in $\vec{u} \cup \vec{v}$ and let $\mathcal{A}$ denote the structure consisting of the atoms $At(\chi(\vec{u},\vec{v}))$, s.t. the variables $\vec{u}$ are considered as constants and the variables $\vec{v}$ as labeled nulls. Let $\mathcal{A}'$ denote the core of $\mathcal{A}$ with $\mathcal{A}' \subseteq \mathcal{A}$, i.e., there exists a substitution $\sigma\colon \vec{v} \rightarrow Const \cup \vec{u} \cup \vec{v}$ s.t. $At(\chi(\vec{u}, \vec{v}\sigma)) = \mathcal{A}' \subseteq At(\chi(\vec{u},\vec{v}))$. Then we define the* core *of $\chi(\vec{u},\vec{v})$ as the CQ $\chi(\vec{u},\vec{v}\sigma)$.*

DEFINITION 3.5. *Let $\chi(\vec{u},\vec{v})$ be a CQ with variables in $\vec{u} \cup \vec{v}$. We set up the* dual graph *$\mathcal{G}(\tau)$ as follows: The atoms of $\chi(\vec{u},\vec{v})$ are the vertices of $\mathcal{G}(\tau)$. Two vertices are connected if the corresponding atoms have at least one variable from $\vec{v}$ in common. Let $\{C_1,\dots,C_n\}$ denote the connected components of $\mathcal{G}(\tau)$. Moreover, for every $i \in \{1,\dots,n\}$, let $\vec{v}_i$ with $\emptyset \subseteq \vec{v}_i \subseteq \vec{v}$ denote those variables from $\vec{v}$, which actually occur in C_i and let $\chi_i(\vec{u},\vec{v}_i)$ denote the CQ consisting of the atoms in C_i. Then we define the* components *of $\chi(\vec{u},\vec{v})$ as the set $\{\chi_1(\vec{u},\vec{v}_1),\dots,\chi_n(\vec{u},\vec{v}_n)\}$.*

The splitting rule (i.e., Rule 3 in Figure 1) was already applied in Example 1.3. Rule 2 involving core computation of the antecedent was applied in Example 1.1, when we reduced $L(x_1,x_2,x_3) \wedge L(x_4,3,x_5) \wedge P(x_5,x_6)$ to its core $L(x_4,3,x_5) \wedge P(x_5,x_6)$. Likewise, in Example 3.1, the simplification of τ_1 and τ_2 to τ_1' and τ_2' is due to Rule 2. Analogously, Rule 1 involving core computation of the conclusion allowed us to reduce $L(y_1,y_2,y_3) \wedge L(x_1,3,y_4) \wedge P(y_4,x_2)$ in Example 1.2 to $L(x_1,3,y_4) \wedge P(y_4,x_2)$. In Example 3.1, Rule 1 was applied when we replaced τ_2' by τ_2''.

The following example illustrates that additional rules are required in order to remove an st-tgd or a part of an st-tgd whose redundancy is due to the presence of other st-tgds.

EXAMPLE 3.2. Consider the set $\Sigma = \{\tau_1', \tau_2'', \tau_3\}$, where τ_1' and τ_2'' are the st-tgds resulting from the simplification steps in Example 3.1 and τ_3 is given below:

Rewrite rules to simplify a set of st-tgds

Rule 1 (Core of the conclusion, see Definition 3.4).
$\tau\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y}) \implies$
$\tau'\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y}\sigma)$,
s.t. $\psi(\vec{x},\vec{y}\sigma)$ is the core of $\psi(\vec{x},\vec{y})$.

Rule 2 (Core of the antecedent, see Definition 3.4).
$\tau\colon \varphi(\vec{x}_1,\vec{x}_2) \rightarrow (\exists\vec{y})\psi(\vec{x}_1,\vec{y}) \implies$
$\tau'\colon \varphi(\vec{x}_1,\vec{x}_2\sigma) \rightarrow (\exists\vec{y})\psi(\vec{x}_1,\vec{y})$,
s.t. $\varphi(\vec{x}_1,\vec{x}_2\sigma)$ is the core of $\varphi(\vec{x}_1,\vec{x}_2)$.

Rule 3 (Splitting, see Definition 3.5).
$\tau\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y}) \implies \{\tau_1,\dots,\tau_n\}$, s.t.
$\{\chi(\vec{x},\vec{y}_1),\dots,\chi(\vec{x},\vec{y}_n)\}$ are the components of $\psi(\vec{x},\vec{y})$
and $\tau_i\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y}_i)\chi(\vec{x},\vec{y}_i)$ for $i \in \{1,\dots,n\}$.

Rule 4 (Implication of an st-tgd).
$\Sigma \implies \Sigma \setminus \{\tau\}$
if $\Sigma \setminus \{\tau\} \models \tau$.

Rule 5 (Implication of atoms in the conclusion).
$\Sigma \implies (\Sigma \setminus \{\tau\}) \cup \{\tau'\}$
if $\tau\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y})$
and $\tau'\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y}')\psi'(\vec{x},\vec{y}')$,
s.t. $At(\psi'(\vec{x},\vec{y}')) \subset At(\psi(\vec{x},\vec{y}))$
and $(\Sigma \setminus \{\tau\}) \cup \{\tau'\} \models \tau$.

Figure 1: Redundancy elimination from a set of st-tgds.

$\tau_1'\colon S(x_1,x_2) \rightarrow (\exists y_1,y_2,y_3)$
$\qquad P(x_1,y_2,y_1) \wedge R(y_1,y_3,x_2) \wedge R(2,y_3,x_2)$

$\tau_2''\colon S(x_1,x_3) \rightarrow (\exists y_4)P(x_1,y_4,2) \wedge Q(y_4,x_3)$

$\tau_3\colon S(2,x) \rightarrow (\exists y)R(2,y,x)$

The tgd τ_3 generates only a part of the atoms that τ_1' does, and fires in strictly fewer cases than τ_1'. Hence, τ_3 may be deleted. Moreover, considering the combined effect of the rules τ_1' and τ_2'', which fire on exactly the same tuples, and a substitution $\{y_1 \leftarrow 2, y_2 \leftarrow y_4\}$, we notice that the first two atoms in the conclusion of τ_1' are in fact redundant. It is indeed possible to reduce τ_1' to

$\tau_1''\colon S(x_1,x_2) \rightarrow (\exists y_3)R(2,y_3,x_2)$.

In total, Σ may be replaced by $\Sigma' = \{\tau_1'',\tau_2''\}$. $\square$

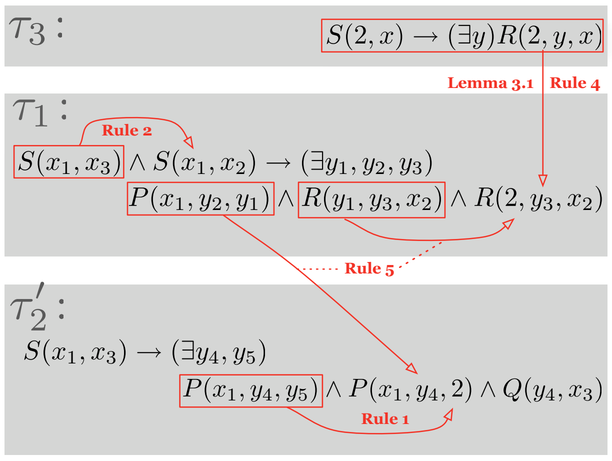
Rules 4 and 5 in Figure 1 allow us to eliminate such redundancies from a set Σ of st-tgds: By Rule 4, we may delete an st-tgd τ from Σ, if τ is implied by the others, like τ_3 in Example 3.2. Rule 5 allows us to replace a rule τ by a strictly smaller rule (with fewer atoms in the conclusion) if τ is implied by τ' together with the remaining st-tgds in Σ (cf. the replacement of τ_1' with τ_1'' in Example 3.2 above). Figure 2 illustrates the elimination of redundant atoms through Rules 1, 2, 4 and 5 in a set $\Sigma = \{\tau_1,\tau_2',\tau_3\}$ of tgds from Examples 3.1 and 3.2.

In principle, the implication of a tgd by a set of dependencies can be tested by a procedural criterion based on the chase [4]. For our purposes, the following, declarative criterion is more convenient.

LEMMA 3.1. *Consider an st-tgd $\tau\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y})$ and a set Σ of st-tgds. Then $\Sigma \models \tau$ holds iff there exist (not necessarily distinct) st-tgds $\tau_1,\dots,\tau_k$ in Σ, s.t. all st-tgds $\tau,\tau_1,\dots,\tau_k$ are pairwise variable disjoint and the following conditions hold:*

(a) For every $i \in \{1,\dots,k\}$, there exists a substitution $\lambda_i\colon \vec{x}_i \rightarrow Const \cup \vec{x}$, s.t. $At(\varphi_i(\vec{x}_i\lambda_i)) \subseteq At(\varphi(\vec{x}))$.

(b) There exists a substitution $\mu\colon \vec{y} \rightarrow Const \cup \vec{x} \cup \bigcup_{i=1}^{k}\vec{y}_i$, s.t. $At(\psi(\vec{x},\vec{y}\mu)) \subseteq \bigcup_{i=1}^{k} At(\psi_i(\vec{x}_i\lambda_i,\vec{y}_i))$.



τ_3 :
$S(2,x) \rightarrow (\exists y)R(2,y,x)$
Lemma 3.1
Rule 4
τ_1 :
Rule 2
$S(x_1,x_3) \wedge S(x_1,x_2) \rightarrow (\exists y_1,y_2,y_3)$
$P(x_1,y_2,y_1) \wedge R(y_1,y_3,x_2) \wedge R(2,y_3,x_2)$
Rule 5
τ_2' :
$S(x_1,x_3) \rightarrow (\exists y_4,y_5)$
$P(x_1,y_4,y_5) \wedge P(x_1,y_4,2) \wedge Q(y_4,x_3)$
Rule 1

Figure 2: Tgd optimization. Rectangles mark eliminated atoms, arrows show justifications for elimination.

Note that Rule 5 generalizes Rule 1 and, in principle, also Rule 4. Indeed, if we restrict Σ in Rule 5 to the singleton $\Sigma = \{\tau\}$, then the replacement of τ by τ' means that we reduce $\psi(\vec{x},\vec{y})$ to its core. Moreover, Rule 5 allows us to eliminate all atoms from the conclusion of τ iff τ may be deleted via Rule 4. Clearly, the deletion of the conclusion of τ essentially comes down to the deletion of τ itself. The correctness of Rules 1 – 5 is easily established.

LEMMA 3.2. *The Rules 1 – 5 in Figure 1 are correct, i.e.: Let Σ be a set of st-tgds and $\tau \in \Sigma$. Suppose that Σ is transformed into Σ' by applying one of the Rules 1 – 5 to τ, i.e.:*

– τ is replaced by a single st-tgd τ' (via Rule 1,2,5),
– τ is replaced by several st-tgds $\tau_1,\dots,\tau_n$ (via Rule 3),
– or τ is deleted (via Rule 4).

Then Σ and Σ' are equivalent.

The following notion of a "proper instance" of an st-tgd plays an important role for proving that our Rules 1 – 5 lead to a unique normal form. A proper instance of an st-tgd τ is obtained from τ by leaving the antecedent unchanged and by eliminating at least one of the existentially quantified variables in the conclusion of τ.

DEFINITION 3.6. *Let $\tau\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y})$ be an st-tgd. We call an st-tgd τ' a* proper instance *of τ, if there exists a strict subset $\vec{y}' \subset \vec{y}$ and a substitution $\sigma\colon \vec{y} \rightarrow Const \cup \vec{x} \cup \vec{y}'$, s.t. τ' is of the form $\tau'\colon \varphi(\vec{x}) \rightarrow (\exists\vec{y}')\psi(\vec{x},\vec{y}\sigma)$.*

EXAMPLE 3.3. In the following three tgds, each next tgd is a proper instance of the previous ones:

$\tau_1\colon S(x_1,x_2) \rightarrow (\exists y_1,y_2)Q(x_1,y_1,y_2)$

$\tau_2\colon S(x_1,x_2) \rightarrow (\exists y_1)Q(x_1,y_1,y_1)$

$\tau_3\colon S(x_1,x_2) \rightarrow Q(x_1,x_2,x_2)$

Moreover, we observe that $\tau_2 \models \tau_1$ and $\tau_3 \models \tau_2$ holds. $\square$

The importance of "proper instances" to our investigations comes from the following properties:

LEMMA 3.3. *Let τ and τ' be st-tgds, s.t. τ' is a proper instance of τ. Then the following properties hold:*

(1) $\tau' \models \tau$.

(2) Suppose that τ is reduced w.r.t. Rule 1. Then $\tau \not\models \tau'$.

LEMMA 3.4. *Let τ be an st-tgd and let Σ be a set of st-tgds. Moreover, suppose that τ is reduced w.r.t. Rules 1 – 3.*

If $\Sigma \models \tau$, then one of the following two conditions is fulfilled: Either there exists a single st-tgd $\tau_0 \in \Sigma$, s.t. $\tau_0 \models \tau$ or there exists a proper instance τ' of τ, s.t. $\Sigma \models \tau'$.

PROOF. (Idea) The intuition underlying this lemma is that we cannot stitch together the conclusion of a reduced st-tgd τ (which consists of a single connected component!) by two other st-tgds. Two st-tgds produce two connected components, which first have to be created out of the conclusion of τ by instantiating at least one variable in $\vec{y}$ to a constant resp. a variable in $\vec{x}$. □

LEMMA 3.5. *Suppose that an st-tgd $\tau \in \Sigma$ is reduced w.r.t. Rules 1 – 3 and that τ cannot be deleted via Rule 4. If there exists a proper instance τ' of τ, s.t. $\Sigma \models \tau'$ holds, then there exists an st-tgd τ'', s.t. τ may be replaced by τ'' via Rule 5.*

PROOF. (Sketch) By $\Sigma \models \tau'$, there exist $\tau_1, \ldots, \tau_k$ in Σ, s.t. conditions (a) and (b) of Lemma 3.1 are fulfilled. Let $\lambda_1, \ldots, \lambda_k$ denote the substitutions in condition (a). By (b), there exsits a substitution μ, s.t. $\tau'\colon \varphi(\vec{x}) \rightarrow (\exists \vec{y}\,')\psi'(\vec{x}, \vec{y}\,')$ and $At(\psi'(\vec{x}, \vec{y}\,'\mu)) \subseteq \bigcup_{i=1}^{k} At(\psi_i(\vec{x}_i\lambda_i, \vec{y}_i))$.

Note that $\Sigma \setminus \{\tau\} \not\models \tau'$ since, otherwise also $\Sigma \setminus \{\tau\} \models \tau$ by Lemma 3.3, part (1) and, therefore, τ could be deleted by Rule 4. Hence, at least one of the τ_i coincides with τ (up to variable renaming). Then the CQ $\psi''(\vec{x}, \vec{y}\,'')$ of $\tau''\colon \varphi(\vec{x}) \rightarrow (\exists \vec{y}\,'')\psi''(\vec{x}, \vec{y}\,'')$ is obtained as the conjunction of those atoms $A(\vec{x}, \vec{y}) \in At(\psi(\vec{x}, \vec{y}))$, such that $A(\vec{x}\lambda_i, \vec{y}) \in At(\psi'(\vec{x}, \vec{y}\,'\mu)) \cap At(\psi_i(\vec{x}_i\lambda_i, \vec{y}_i))$ for some τ_i that coincides with τ. Clearly, we have $At(\psi''(\vec{x}, \vec{y}\,'')) \subset At(\psi(\vec{x}, \vec{y}))$ since, otherwise, $\tau \models \tau'$, which contradicts Lemma 3.3, part (2). Hence, τ may be replaced by τ'' via Rule 5. □

We now define a normal form of st-tgds via the rewrite rules of this section. We will then show that this normal form is unique up to isomorphism in the sense defined below.

DEFINITION 3.7. *Let Σ be a set of st-tgds and let Σ' be the result of applying the Rules 1 – 5 of Figure 1 exhaustively to Σ. Then we call Σ' the* normal form *of Σ.*

DEFINITION 3.8. *Let τ_1 and τ_2 be tgds with $\tau_1\colon \varphi_1(\vec{x}_1) \rightarrow (\exists \vec{y}_1)\psi_1(\vec{x}_1, \vec{y}_1)$ and $\tau_2\colon \varphi_2(\vec{x}_2) \rightarrow (\exists \vec{y}_2)\psi_2(\vec{x}_2, \vec{y}_2)$. We say that τ_1 and τ_2 are* isomorphic *if τ_2 is obtained from τ_1 via variable renamings $\eta\colon \vec{x}_1 \rightarrow \vec{x}_2$ and $\vartheta\colon \vec{y}_1 \rightarrow \vec{y}_2$.*

Let Σ_1 and Σ_2 be two sets of tgds. We say that Σ_1 and Σ_2 are isomorphic *if $|\Sigma_1| = |\Sigma_2|$, every $\tau_1 \in \Sigma_1$ is isomorphic to precisely one $\tau_2 \in \Sigma_2$ and every $\tau_2 \in \Sigma_2$ is isomorphic to precisely one $\tau_1 \in \Sigma_1$.*

We start by showing for two single st-tgds τ_1 and τ_2 that logical equivalence and isomorphism coincide provided that the st-tgds are reduced via our rewrite rules. This result will then be extended to sets Σ_1 and Σ_2 of st-tgds.

LEMMA 3.6. *Let τ_1 and τ_2 be two st-tgds and suppose that τ_1 and τ_2 are reduced w.r.t. Rules 1 – 3. Then τ_1 and τ_2 are isomorphic, iff τ_1 and τ_2 are equivalent.*

PROOF. (Sketch) The "$\Rightarrow$"-direction follows immediately from Lemma 3.1. For the "$\Leftarrow$"-direction, let τ_1 and τ_2 be equivalent st-tgds with $\tau_1\colon \varphi_1(\vec{x}_1, \vec{x}_2) \rightarrow (\exists \vec{y})\psi_1(\vec{x}_1, \vec{y})$ and $\tau_2\colon \varphi_2(\vec{u}_1, \vec{u}_2) \rightarrow (\exists \vec{v})\psi(\vec{u}_1, \vec{v})$.

By Lemma 3.1, there exist substitutions λ and ρ, s.t.

$\lambda\colon \vec{x}_1 \cup \vec{x}_2 \rightarrow \mathit{Const} \cup \vec{u}_1 \cup \vec{u}_2$, and

$\rho\colon \vec{u}_1 \cup \vec{u}_2 \rightarrow \mathit{Const} \cup \vec{x}_1 \cup \vec{x}_2$, such that

$At(\varphi_1(\vec{x}_1\lambda, \vec{x}_2\lambda)) \subseteq At(\varphi_2(\vec{u}_1, \vec{u}_2))$ and

$At(\varphi_2(\vec{u}_1\rho, \vec{u}_2\rho)) \subseteq At(\varphi_1(\vec{x}_1, \vec{x}_2))$.

By exploiting the equivalence of τ_1 and τ_2 and the fact that these st-tgds are reduced w.r.t. Rule 2, we can show that the antecedents of τ_1 and τ_2 are isomorphic (i.e., the above inclusions are in fact equalities). Moreover, by exploiting that the st-tgds are reduced w.r.t. Rule 1, we may conclude that τ_1 and τ_2 are isomorphic. □

THEOREM 3.1. *Let Σ_1 and Σ_2 be equivalent sets of st-tgds, i.e., $\Sigma_1 \models \Sigma_2$ and $\Sigma_2 \models \Sigma_1$. Let Σ_1' and Σ_2' denote the normal form of Σ_1 resp. Σ_2. Then Σ_1' and Σ_2' are isomorphic.*

PROOF. (Idea) Clearly, for an arbitrary $\tau_1 \in \Sigma_1$ we have $\Sigma_2 \models \tau_1$. By making use of Lemma 3.4 and 3.5, we can show that τ_1 must be implied by a single st-tgd $\tau_2 \in \Sigma_2$. The basic idea of this proof is the following: Suppose to the contrary that τ_1 is implied by two or more st-tgds from Σ_2. Since also $\Sigma_1 \models \Sigma_2$ holds, we can construct two or more st-tgds $\tau, \tau', \ldots$ in Σ_1 which imply τ_1. But then, by Lemma 3.4, these st-tgds also imply a proper instance of τ_1 and, therefore, by Lemma 3.5, Rule 5 is applicable to τ_1, which is a contradiction. Likewise, every $\tau_2 \in \Sigma_2$ is implied by exactly one st-tgd in Σ_1. It remains to show that every $\tau_1 \in \Sigma_1$ is equivalent to some $\tau_2 \in \Sigma_2$ and vice versa. Finally, by Lemma 3.6, logical equivalence and isomorphism of single st-tgds τ_1 and τ_2 coincide if the st-tgds are reduced w.r.t. Rules 1 – 3. □

We now consider the complexity of computing the normal form of a set of st-tgds. Of course, the application of any of the Rules 1, 2, 4, and 5 is NP-hard, since they involve CQ answering. However, below we show that if the length of each st-tgd (i.e., the number of atoms) is bounded by a constant, then the normal form according to Definition 3.7 can be obtained in polynomial time.

THEOREM 3.2. *Suppose that the length (i.e., the number of atoms) of the st-tgds under consideration is bounded by some constant b. Then there exists an algorithm which reduces an arbitrary set Σ of st-tgds to normal form in polynomial time w.r.t. the total size $||\Sigma||$ of (an appropriate representation of) Σ.*

PROOF. (SKETCH) First, the total number of applications of each rule is bounded by the total number of atoms in all st-tgds in Σ. Indeed, Rule 4 deletes an st-tgd. The Rules 1, 2, and 5 delete at least one atom from an st-tgd. Rule 3 splits the conclusion of an st-tgd in 2 or more parts. Hence, also the total number of applications of Rule 3 is bounded by the total number of atoms in Σ. Finally, the application of each rule is feasible in polynomial time since the most expensive part of these rules is the CQ answering where the length of the CQs is bounded by the number of atoms in a single st-tgd. □

The restriction on the number of atoms in each st-tgd is used in the above proof only in order to show that each rule application is feasible in polynomial time. The argument, that the total number of rule applications is bounded by the total number of atoms in all st-tgds in Σ applies to any set Σ of st-tgds. We thus get:

COROLLARY 3.1. *The rewrite rule system consisting of Rules 1 – 5 is terminating, i.e., Given an arbitrary set Σ of st-tgds, the nondeterministic, exhaustive application of the Rules 1 – 5 terminates.*

It can be shown that the unique normal form produced by our rewrite rules is indeed *optimal*.

THEOREM 3.3. *Let Σ be a set of st-tgds, s.t. Σ is in normal form. Then Σ is optimal according to Definition 3.3.*

We conclude this section by two remarks on the splitting rule:

(1) The purpose of the splitting rule is to enable a further simplification of the antecedents of the resulting st-tgds. Of course, it may happen that no further simplification is possible, e.g.: Let $\Sigma = \{R(x,y) \wedge R(y,z) \rightarrow S(x,z) \wedge T(z,x)\}$. Splitting yields $\Sigma' = \{R(x,y) \wedge R(y,z) \rightarrow S(x,z); R(x,y) \wedge R(y,z) \rightarrow T(z,x)\}$, which cannot be further simplified. In cases like this, one may either "undo" the splitting or simply keep track of st-tgds with identical (possibly up to variable renaming) antecedents in order to avoid multiple evaluation of the same antecedent by the chase.

(2) Definition 3.2 gives a "semantical" definition of "split reduced" while the splitting rule is a "syntactical" criterion. The following lemma establishes the close connection between them.

LEMMA 3.7. *Let Σ be a split-reduced set of STDs and let Σ^* denote the normal form of Σ. Then, for every possible sequence of rewrite rule applications, this normal form Σ^* is obtained from Σ without ever applying Rule 3 (i.e., splitting).*

4. EXTENSION TO TARGET EGDS

We now extend our rewrite rule system to schema mappings with both st-tgds and target egds. Again, we will be able to show that the resulting normal form is unique up to variable renaming and that it has similar optimality properties as in the st-tgd-only case.

An important complication introduced by the egds has already been hinted at in Section 1, namely the equivalence of two sets of st-tgds may be affected by the presence of egds:

EXAMPLE 4.1. (Example 1.5 slightly extended).

$$\begin{aligned}\Sigma_{st} = \{&C(x_1,x_2,x_3) \rightarrow \\ &\quad (\exists y_1,y_2)\ P(y_1,y_2,y_2) \wedge P(y_1,x_2,x_3) \\ &C(x_1,x_2,x_3) \rightarrow (\exists y_1) P(y_1,x_3,x_2) \\ &C(x_1,x_2,x_2) \rightarrow Q(x_1)\}\end{aligned}$$

$$\begin{aligned}\Sigma'_{st} = \{&C(x_1,x_2,x_3) \rightarrow (\exists y_1)\ P(y_1,x_2,x_3) \\ &C(x_1,x_2,x_3) \rightarrow Q(x_1)\}\end{aligned}$$

$$\Sigma_t\ = \{P(x_1,x_2,x_3) \rightarrow x_2 = x_3\}$$

We have $\Sigma_{st} \cup \Sigma_t \equiv \Sigma'_{st} \cup \Sigma_t$. Moreover, both Σ_{st} and Σ'_{st} are in normal form w.r.t. the Rules 1 – 5 from Section 3. However, $\Sigma_{st} \not\equiv \Sigma'_{st}$ holds. □

In contrast, the equivalence of two sets of target egds is not influenced by the presence of st-tgds, as the following lemma shows.

LEMMA 4.1. *Suppose $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ be two logically equivalent sets of st-tgds and target egds. Then, Σ_t and Υ_t are equivalent.*

PROOF. W.l.o.g. assume that there exists an $\varepsilon : \varphi(\vec{x}) \rightarrow \sigma(\vec{x}) \in \Upsilon_t$ s.t. $\Sigma_t \not\models \varepsilon$. That is, the set $L = At(\varphi(\vec{x}))^{\Sigma_t}$ of atoms of the antecedent of ε chased with Σ_t does not satisfy ε. However, it does satisfy Σ_t. Now, consider the pair of instances $\langle \emptyset, L\rangle$. Since $L \models \Sigma_t$, $\langle \emptyset, L\rangle \models \Sigma$ and $\langle \emptyset, L\rangle \not\models \Upsilon$, which is a contradiction. □

Recall that we are only considering *logical* equivalence of dependencies here. The study of weaker notions of equivalence [10] which only take attainable target instances into account (which is not the case for L in the above proof) are left for future work.

Our primary goal is the definition of a unique normal form of the st-tgds also in the presence of target egds. The first step towards this goal is to incorporate the effects of egds into st-tgds. As we have already seen in Section 1, this may require the introduction of source egds. Source instances contain no variables. Hence, there will be no source chase. The source egds are only meant to capture the failure conditions which cannot be detected otherwise after the rewriting of the st-tgds.

```
Procedure PROPAGATE
Input: A set of st-tgds and target egds $\Sigma = \Sigma_{st} \cup \Sigma_t$
Output: Sets of source egds $\Sigma_s$ and rewritten st-tgds $\Sigma^*_{st}$
1. Set $\Sigma_s = \Sigma^*_{st} = \emptyset$;
2. for each st-tgd $\tau$: $\varphi(\vec{x}) \rightarrow (\exists\vec{y})\psi(\vec{x},\vec{y})$ in $\Sigma_{st}$ do
  /* (a) initialization of source and target instance */
    $I_S := At(\varphi(\vec{x}))$;
    $I_T := \emptyset$;
    $I := I_S \cup I_T$;
  /* (b) chase with $\Sigma = \Sigma_{st} \cup \Sigma_t$ */
    $J := I^{\Sigma}$;
  /* (c) transform st-tgd $\tau$ into $\tau'$ */
    let $J = J_S \cup J_T$, s.t. $J_S$ is an instance over $\mathbf{S}$
        and $J_T$ is an instance over $\mathbf{T}$;
    let $J^* = core(J_T)$, where the variables that occur
        in $J_S$ are considered as constants.
    $\tau' := \big(\bigwedge_{A \in J_S} A\big) \rightarrow (\exists\vec{y}) \bigwedge_{B \in J^*} B$;
    $\Sigma^*_{st} := \Sigma^*_{st} \cup \{\tau'\}$;
  /* (d) generate source egds */
    Compute a substitution $\lambda$ s.t. $At(\varphi(\vec{x}\lambda)) = J_S$;
    for each pair of variables $x_j, x_k \in \vec{x}$ do
        if $x_j\lambda = x_k\lambda$ then
            $\Sigma_s := \Sigma_s \cup \{\varphi(\vec{x}) \rightarrow x_j = x_k\}$;
  end for;
```

Figure 3: Procedure Propagate.

In Figure 3, we present the procedure PROPAGATE, which propagates the effect of the target egds into the st-tgds and thereby possibly generates source egds. The idea of this procedure is that, for every st-tgd τ, we identify all egds that will "fire" whenever τ does. Moreover, we want that all equalities enforced by these egds should already be enforced in the st-tgd. Note that the chase in step 2.(b) is not the usual chase in data exchange. Here we chase all of I (consisting of both a source and a target instance) with Σ in order to propagate backwards the effect of the target egds. We are assuming that the st-tgds contain only variables. Hence, the chase in step 2.(b) will never fail – it only equates variables. Of course, if st-tgds were also allowed to contain constants, then the chase in step 2.(b) could fail. In this case, we would simply delete τ and generate a source dependency of the form $\varphi(\vec{x}) \rightarrow \bot$ to indicate that the st-tgd τ must never fire.

EXAMPLE 4.2. We now apply the PROPAGATE procedure to $\Sigma = \Sigma_{st} \cup \Sigma_t$ from Example 4.1. We start the loop with τ: $C(x_1,x_2,x_3) \rightarrow (\exists y_1,y_2)\ P(y_1,y_2,y_2) \wedge P(y_1,x_2,x_3)$.

(a) $I := \{C(x_1,x_2,x_3)\}$.

(b) Chasing I with Σ_{st} yields $I' = \{C(x_1,x_2,x_3), P(y'_1,y'_2,y'_2), P(y'_1,x_2,x_3), P(y''_1,x_3,x_2)\}$. We apply the egd in Σ_t to get $I'' = \{C(x_1,x_2,x_2), P(y'_1,y'_2,y'_2), P(y'_1,x_2,x_2), P(y''_1,x_2,x_2)\}$. The third st-tgd in Σ_{st} is now applicable and we get $J = I^{\Sigma} = \{C(x_1, x_2,x_2), P(y'_1,y'_2,y'_2), P(y'_1,x_2,x_2), P(y''_1,x_2,x_2), Q(x_1)\}$.

(c) We have $J = J_S \cup J_T$ with $J_S = \{C(x_1,x_2,x_2)\}$ and $J_T = \{P(y'_1,y'_2,y'_2), P(y'_1,x_2,x_2), P(y''_1,x_2,x_2), Q(x_1)\}$. Core computation of J_T yields $J^* = \{P(y'_1,x_2,x_2), Q(x_1)\}$. Hence, τ is transformed into $\tau'\colon C(x_1,x_2,x_2) \rightarrow \exists y'_1\ P(y'_1,x_2,x_2) \wedge Q(x_1)$.

(d) We compute the substitution $\lambda = \{x_3 \leftarrow x_2\}$, which maps the only atom $C(x_1,x_2,x_3)$ in $\varphi(\vec{x}\,)$ into $J_S = \{C(x_1,x_2,x_2)\}$. We thus get one source egd $C(x_1,x_2,x_3) \rightarrow x_2 = x_3$.

After the first iteration of the loop, we thus have
$\Sigma^*_{st} = \{C(x_1,x_2,x_2) \rightarrow \exists y'_1\ P(y'_1,x_2,x_2) \wedge Q(x_1)\}$ and
$\Sigma_s = \{C(x_1,x_2,x_3) \rightarrow x_2 = x_3\}$. The remaining two iterations only yield isomorphic st-tgds and source egds. □

The PROPAGATE procedure never increases the antecedents of the st-tgds. Hence, the cost of the join-operations when computing a canonical solution is not affected. On the other hand, the size of the conclusions is normally increased by this procedure. Note however that all atoms thus accumulated in the conclusion of some st-tgd τ would be generated in a target instance anyway, whenever τ fires. Moreover, we will ultimately delete redundant atoms in the conclusion of all st-tgds via Rules 1 and 5 from the previous section. However, for the time being, it is important to have all these atoms present. This ensures that dependencies resulting from the PROPAGATE procedure possess the following essential properties.

LEMMA 4.2. *Consider a set $\Sigma = \Sigma_{st} \cup \Sigma_t$ of st-tgds Σ_{st} and target egds Σ_t. Moreover, let the result of* PROPAGATE(Σ_{st}, Σ_t) *be denoted by $(\Sigma_s, \Sigma^*_{st})$. Then, the following conditions hold:*

(1) For every st-tgd $\tau \in \Sigma^*_{st}$: the chase of the frozen antecedent database of τ with $\Sigma_{st} \cup \Sigma_t$ is successful.

(2) For every source instance I: if $I \not\models \Sigma_s$ then the chase of I with $\Sigma_{st} \cup \Sigma_t$ fails.

PROOF. (1) After the successful completion of the chase in step 2.(b) of the PROPAGATE procedure, all necessary unifications in the antecedent relations have been performed. Hence, the instance $\langle At(\varphi(\vec{x}\,)),\ At(\psi(\vec{x},\vec{y}\,))\,\rangle$ for an st-tgd $\varphi(\vec{x}\,) \rightarrow (\exists\vec{y}\,)\psi(\vec{x},\vec{y}\,)$ in Σ^*_{st} satisfies both Σ_{st} and Σ_t. Freezing the variables in $At(\varphi(\vec{x}\,))$ (i.e., considering them as constants) makes no difference.

(2) An inspection of steps 2.(b) and (d) of the PROPAGATE procedure reveals that Σ_s enforces only those equalities which are implied by $\Sigma_{st} \cup \Sigma_t$. Therefore, a violation of Σ_s means that also $\Sigma_{st} \cup \Sigma_t$ is violated. □

LEMMA 4.3. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$, and let $(\Sigma_s, \Sigma^*_{st})$ denote the result of* PROPAGATE(Σ_{st}, Σ_t). *Moreover, let $\tau \in \Sigma^*_{st}$ with $\tau\colon \varphi(\vec{x}) \rightarrow \exists(\vec{y})\ \psi(\vec{x},\vec{y})$, and let I be a source instance with $I \subseteq At(\varphi(\vec{x}))$, i.e., I is obtained as an arbitrary* subset *of the frozen antecedent database $At(\varphi(\vec{x}))$ of τ.*

Then the chase of I both with $\Sigma = \Sigma_{st} \cup \Sigma_t$ and with $\Sigma^* = \Sigma_s \cup \Sigma^*_{st} \cup \Sigma_t$ succeeds. Moreover, $core(I^{\Sigma}) = core(I^{\Sigma^*})$.

PROOF. By condition (1) of Lemma 4.2, the chase with Σ and with Σ^* succeeds on the frozen antecedent database $At(\varphi(\vec{x}))$ of τ. Hence, the chase clearly succeeds also for any subset of $At(\varphi(\vec{x}))$. The equality $core(J^{\Sigma}) = core(J^{\Sigma^*})$ immediately follows from the correctness of the PROPAGATE procedure, see Lemma 4.5. □

By the above lemmas, the PROPAGATE procedure incorporates some of the effects of the target egds into the st-tgds. Nevertheless, as the following examples shows, additional measures are needed for a unique normal form of the st-tgds in the presence of egds:

EXAMPLE 4.3. Let $\Sigma_1 = \{\varepsilon, \tau_1\}$ and $\Sigma_2 = \{\varepsilon, \tau_2\}$ with

$\varepsilon\colon R(x_1,x_2,x_3) \wedge R(x_1,x_4,x_5) \rightarrow x_3 = x_5$
$\tau_1\colon S(x_1,x_2) \wedge S(x_1,x_3) \rightarrow \exists y\, R(x_1,x_2,y) \wedge R(x_1,x_3,y)$
$\tau_2\colon S(x_1,x_2) \rightarrow \exists y\, R(x_1,x_2,y)$

Applying PROPAGATE to $\Sigma_1 = \{\varepsilon, \tau_1\}$ or $\Sigma_2 = \{\varepsilon, \tau_2\}$ neither produces source egds nor changes the τ_i's. However, it is easy to verify that Σ_1 and Σ_2 are equivalent. We only show that $\Sigma_2 \models \tau_1$. Let $\langle S, T\rangle$ be an arbitrary pair of source and target instance with $\langle S, T\rangle \models \Sigma_2$, s.t. the antecedent of τ_1 is satisfied by atoms $S(a,b)$ and $S(a,c)$ in S. Chasing these atoms with τ_2 yields $R(a,b,y)$ and $R(a,c,y')$. The egd ε is applicable to these atoms and enforces the equality $y = y'$. Hence, T contains atoms $R(a,b,y)$ and $R(a,c,y)$, thus satisfying the conclusion of τ_1. □

Rewrite Rules in the Presence of Egds

Rule E1 (General implication).
$\Sigma \Longrightarrow \Sigma \setminus \{\delta\}$
if $\Sigma \setminus \{\delta\} \models \delta$.

Rule E2 (Restriction of an antecedent to subsets).
$\Sigma \Longrightarrow (\Sigma \setminus \{\tau\}) \cup \{\tau_1, \ldots, \tau_n\}$
if $\tau\colon \varphi(\vec{x}\,) \rightarrow (\exists\vec{y}\,)\psi(\vec{x},\vec{y}\,)$
and $(\Sigma \setminus \{\tau\}) \cup \{\tau_1, \ldots, \tau_n\} \models \tau$
and for each $i \in \{1, \ldots, n\}$
$\tau_i\colon \varphi_i(\vec{x}_i,) \rightarrow (\exists\vec{y_i})\psi_i(\vec{x}_i,\vec{y}_i)$,
s.t. $\emptyset \subset At(\varphi_i(\vec{x}_i)) \subset At(\varphi(\vec{x}))$
and $\psi_i(\vec{x}_i,\vec{y}_i) = core(At(\varphi_i(\vec{x}_i))^{\Sigma})$.

Figure 4: Rewrite rules in the presence of egds.

In Figure 4, we define two more rewrite rules. Rule E1 allows the deletion of implied dependencies analogously to Rule 4 of Section 3. However, since we are now dealing with st-tgds and egds, the declarative implication criterion from Lemma 3.1 no longer works. Instead, we take the chase-based procedure by Beeri and Vardi [4], applicable to any embedded dependencies that cannot cause an infinite chase (which is clearly the case when all tgds are st-tgds).

LEMMA 4.4. *[4] Let Σ be a set of acyclic tgds and egds and let δ be either a tgd or an egd. Let $\varphi(\vec{x}\,)$ denote the antecedent of δ and let T denote the database obtained by chasing $At(\varphi(\vec{x}\,))$ with Σ. The variables in $\vec{x}$ are considered as labeled nulls. Then $\Sigma \models \delta$ iff $T \models \delta$ holds.*

Rule E2 generalizes the Rule 2 from Section 3 in that it aims at the restriction of the antecedent $\varphi(\vec{x}\,)$ of an st-tgd to one or more strict subsets of $At(\varphi(\vec{x}\,))$. Let $\varphi_i(\vec{x}_i)$ be such a conjunction of atoms with $At(\varphi_i(\vec{x}_i)) \subset At(\varphi(\vec{x}\,))$. Then we compute the corresponding conclusion by chasing the frozen antecedent database $At(\varphi_i(\vec{x}_i))$ with Σ followed by the core computation (precisely as in Rule 1). By Lemma 4.3, the chase succeeds and, by construction, each of the resulting rules τ_i is implied by Σ. Moreover, the termination of applications of Rule E2 is due to a simple multiset argument on the strictly shrinking number of atoms in the antecedent. In total, we state the following correctness property of the transformations defined in this Section:

LEMMA 4.5. *The* PROPAGATE *procedure as well as the rewrite rules E1 and E2 in Figure 4 are correct, i.e.:*

(1) Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and let the result of PROPAGATE(Σ_{st}, Σ_t) *be $(\Sigma_s, \Sigma^*_{st})$. Then $\Sigma \equiv \Sigma'$ for $\Sigma' = \Sigma_s \cup \Sigma^*_{st} \cup \Sigma_t$.*

(2) Let Σ be a set of dependencies and let Σ' be the result of applying one of the rules E1 or E2 to Σ. Then $\Sigma \equiv \Sigma'$.

PROOF. (Sketch) (1) PROPAGATE leaves Σ_t unchanged. Moreover, Lemma 4.2, part (2), implies $\Sigma \models \Sigma_s$. It thus remains to show $\Sigma \models \tau'$ for evey $\tau' \in \Sigma^*_{st}$ and $\Sigma' \models \tau$ for evey $\tau \in \Sigma_{st}$. These relationships are proved by inspecting the loop in PROPAGATE procedure (in particular, step 2.b) and checking that the implication criterion of [4] recalled in Lemma 4.4 is fulfilled.

(2) The correctness of Rule E1 is analogous to the correctness of Rule 1 in Lemma 3.1. Now suppose that Σ' is the result of an application of Rule E2. The implication $\tau \models \tau_i$ holds by construction, for every $i \in \{1, \dots, n\}$. Hence, we clearly have $\Sigma \models \Sigma'$. Conversely, we also have $\Sigma' \models \Sigma$, since this condition is part of the definition of Rule E2. □

The PROPAGATE procedure plus the rules E1 and E2 are indeed all ingredients we need to transform the st-tgds in two equivalent sets $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ in such a way that resulting sets Σ^*_{st} and Υ^*_{st} of st-tgds are equivalent.

THEOREM 4.1. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ be two logically equivalent sets consisting of st-tgds and target egds. Moreover, let $(\Sigma_s, \Sigma'_{st}) =$ PROPAGATE(Σ_{st}, Σ_t) and $(\Upsilon_s, \Upsilon'_{st}) =$ PROPAGATE$(\Upsilon_{st}, \Upsilon_t)$, and suppose that Σ^*_{st} and Υ^*_{st} have been obtained by further reducing Σ'_{st} and Υ'_{st} respectively w.r.t. the rules E1 and E2. Then Σ^*_{st} and Υ^*_{st} are equivalent.*

PROOF. We have to show that $\Upsilon^*_{st} \models \sigma$ for every $\sigma \in \Sigma^*_{st}$ (and, symmetrically, $\Sigma^*_{st} \models \tau$ for every $\tau \in \Upsilon^*_{st}$). For an arbitrary $\sigma \in \Sigma^*_{st}$, let $\bar{\Sigma}_\sigma$ denote the set of st-tgds whose antecedents are the proper subsets of σ and whose conclusions are obtained by chasing the corresponding antecedent database with $\Sigma^*_{st} \cup \Sigma_t$ (i.e., we get st-tgds analogous to the τ_i's in Rule E2). Then it can be shown that there exists a subset $T_\sigma \subseteq \Upsilon^*_{st}$, s.t. $T_\sigma \cup \Sigma_s \cup \bar{\Sigma}_\sigma \cup \Sigma_t \cup \Sigma^*_{st} \setminus \{\sigma\} \models \sigma$ and every $\tau \in T_\sigma$ fulfills the following properties:

1. The antecedent $\varphi_\tau(\vec{x}_\tau)$ of τ is homomorphically equivalent to the antecedent $\varphi_\sigma(\vec{x}_\sigma)$ of σ;
2. there exists a substitution λ, such that $\varphi_\tau(\vec{x}_\tau \lambda) = \varphi_\sigma(\vec{x}_\sigma)$. That is, the antecedent of τ can be mapped onto the entire antecedent of σ;
3. τ is not equivalent to any dependency in $\Sigma_{st} \setminus \{\sigma\}$.

By symmetry, for every $\tau \in \Upsilon^*_{st}$ there exists a $S_\tau \subseteq \Sigma^*_{st}$ with the analogous properties. It can be further shown that all these sets T_σ and S_τ are singletons and there exists a one-to-one correspondence between them in the following sense: $\tau \in T_\sigma$ iff $\sigma \in S_\tau$. From this, we can ultimately conclude that $\tau \models \sigma$ and $\sigma \models \tau$, which clearly implies $\Upsilon^*_{st} \models \sigma$ and $\Sigma^*_{st} \models \tau$. □

Intuitively, the effect of the PROPAGATE procedure followed by the Rules E1 and E2 is to fully incorporate the effect of the egds on the antecedent and conclusion of the st-tgds into the st-tgds themselves. We can then apply all the rewrite rules from Section 3 to Σ^*_{st} and Υ^*_{st} to get further simplified, isomorphic sets of st-tgds. In summary, we get the following normal form:

DEFINITION 4.1. *Consider a set $\Sigma = \Sigma_{st} \cup \Sigma_t$ of st-tgds Σ_{st} and target egds Σ_t and let the result of* PROPAGATE(Σ_{st}, Σ_t) *be denoted by (Σ_s, Σ'_{st}) Moreover, let Σ^*_{st} denote the set of st-tgds resulting from Σ'_{st} by exhaustive application of the rules E1, E2 as well as the rules 1–5 from Section 3 and let Σ^*_s denote the result of exhaustive reduction of Σ_s via rule E1. Then we call $\langle \Sigma^*_s, \Sigma^*_{st}, \Sigma_t \rangle$ the* normal form *of Σ.*

THEOREM 4.2. *Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ and $\Upsilon = \Upsilon_{st} \cup \Upsilon_t$ be equivalent sets consisting of st-tgds and target egds and let $\langle \Sigma^*_s, \Sigma^*_{st}, \Sigma_t \rangle$ and $\langle \Upsilon^*_s, \Upsilon^*_{st}, \Upsilon_t \rangle$ be the corresponding normal forms. Then the equivalences $\Sigma^*_{st} \equiv \Upsilon^*_{st}$ and $\Sigma_t \equiv \Upsilon_t$ hold. Moreover, Σ^*_{st} and Υ^*_{st} are isomorphic.*

PROOF. (1) The equivalence $\Sigma_t \equiv \Upsilon_t$ was shown in Lemma 4.1. The equivalence $\Sigma^*_{st} \equiv \Upsilon^*_{st}$ is due to Theorem 4.1 and the correctness of the Rules 1 – 5 proved in Lemma 3.2. Moreover, it follows immediately from Theorems 4.1 and 3.1 that Σ^*_{st} and Υ^*_{st} are isomorphic. □

A normal form of the (source or target) egds is not important for our purposes since we will show in Theorem 5.1 that the unique (up to isomorphism) canonical solution only depends on the normalization of the st-tgds – the equivalence of the source egds and the syntactic representation of the egds are irrelevant.

To sum up, our normal form according to Definition 4.1 guarantees that the st-tgds are unique up to isomorphism. Moreover, they are optimal according to the optimality criteria from Section 3 among all st-tgds where the effect of the egds has been fully incorporated, i.e., if two sets $\Sigma_{st} \cup \Sigma_t$ and $\Sigma'_{st} \cup \Sigma_t$ are equivalent then Σ_{st} and Σ'_{st} are equivalent as well. However, if we lift the restriction of incorporating the egds into the st-tgds then additional possibilities of optimization may arise. The PROPAGATE procedure seems advantageous in any case since it applies to an st-tgd τ all egds and st-tgds that inevitably follow the application of τ on any source instance. The situation with Rule E2 is more complex. In Example 4.3, we have seen that Rule E2 may help to eliminate atoms (in this case, both in the antecedent and the conclusion) whose redundancy cannot be detected by any of the Rules 1 – 5. However, there may also be situations where the total number of atoms in antecedent and conclusion increase when Rule E2 is applied (it is however guaranteed that every st-tgd thus generated is strictly smaller than the original one). Exploring the potential of further optimization is left for future work.

5. AGGREGATE QUERIES

We now study the semantics and evaluation of aggregate queries in data exchange, i.e., queries of the form `SELECT` f `FROM` R, where f is an aggregate operator $\mathsf{min}(R.A)$, $\mathsf{max}(R.A)$, $\mathsf{count}(R.A)$, $\mathsf{count}(*)$, $\mathsf{sum}(R.A)$, or $\mathsf{avg}(R.A)$, and where R is a target relation symbol or, more generally, a conjunctive query over the target schema and A is an attribute of R. For this purpose, we first recall some basic notions on query answering in data exchange as well as some fundamental results on aggregate queries from [1].

Certain Answers. Though any target database satisfying the schema mapping and local constraints is called a "solution", a random choice of a candidate for materializing a target database is not satisfactory: query answering in data exchange cannot be reduced to evaluating queries against random solutions. The widely accepted approach is based on the notion of *certain answers*:

DEFINITION 5.1. *Let Σ be a schema mapping over the schema $\langle \mathbf{S}, \mathbf{T} \rangle$, and let I be an instance over $\mathbf{S}$. Then, the certain answer for a query q over $\mathbf{T}$ and for the source instance I are*

$$certain(q, I, \mathcal{W}(I)) = \bigcap \{q(J) | J \in \mathcal{W}(I)\},$$

where $\mathcal{W}(I)$ is the set of possible worlds *for I and Σ.*

Several proposals can be found in the literature [9, 13, 19, 20] as to which solutions should be taken as possible worlds $\mathcal{W}(I)$. Typical examples are the set of all solutions, the set of universal solutions, the core of the universal solutions, or the CWA-solutions. For conjunctive queries, all these proposals lead to identical results.

Aggregate Certain Answers. Afrati and Kolaitis [1] initiated the study of the semantics of aggregate queries in data exchange. They adopted the notion of *aggregate certain answers* for inconsistent databases of Arenas et al. [3] to data exchange:

DEFINITION 5.2. *[1] Let query q be of the form* SELECT f FROM R*, where R is a target relation symbol or, more generally, a first-order query over the target schema* $\mathbf{T}$*, and f is one of the aggregate operators* $\mathsf{min}(R.A)$*,* $\mathsf{max}(R.A)$*,* $\mathsf{count}(R.A)$*,* $\mathsf{count}(*)$*,* $\mathsf{sum}(R.A)$*, or* $\mathsf{avg}(R.A)$ *for some attribute A of R. For all aggregate operators but* $\mathsf{count}(*)$*, tuples with a null value in attribute $R.A$ are ignored in the computation.*

- *Value r is a possible answer of q w.r.t. I and $\mathcal{W}(I)$ if there is an instance $J \in \mathcal{W}(I)$ such that $f(q)(J) = r$.*
- *$poss(f(q), I, \mathcal{W}(I))$ denotes the set of all possible answers of the aggregate query $f(q)$ w.r.t. I and $\mathcal{W}(I)$.*
- *The aggregate certain answer agg-certain$(f, I, \mathcal{W}(I))$ of the aggregate query $f(q)$ w.r.t. I and $\mathcal{W}(I)$ is the interval*

  $$[\,\mathsf{glb}(poss(f(q), I, \mathcal{W}(I))),\ \mathsf{lub}(poss(f(q), I, \mathcal{W}(I)))\,],$$

  where glb *and* lub *stand, respectively, for the greatest lower bound and the least upper bound.*

Semantics of aggregate queries via endomorphic images. A key issue in defining the semantics of queries in data exchange is to define which set of possible worlds should be considered. In [1], Afrati and Kolaitis showed that all previously considered sets of possible worlds yield a trivial semantics of aggregate queries. They therefore introduced a new approach via the *endomorphic images of the canonical solution*. Let $Endom(I, \mathcal{M})$ denote the endomorphic images of the canonical universal solution $J^* = CanSol(I)$, i.e.: $J \in Endom(I, \mathcal{M})$ if there exists an endomorphism $h\colon J^* \to J^*$, s.t. $J = h(J^*)$. It was shown in [1] that choosing $\mathcal{W}(I) = Endom(I, \mathcal{M})$ leads to an interesting and non-trivial semantics of aggregate queries. However, in general, the semantics definition depends on the concrete syntactic representation of the st-tgds.

EXAMPLE 5.1. Consider two schema mappings $\mathcal{M}_1 = \langle \mathbf{S}, \mathbf{T}, \Sigma_1 \rangle$ and $\mathcal{M}_2 = \langle \mathbf{S}, \mathbf{T}, \Sigma_2 \rangle$, with source schema $\mathbf{S} = \{P\}$, target schema $\mathbf{T} = \{R\}$, and the following st-tgds:

$\Sigma_1 = \{P(x) \to (\exists y) R(1, x, y)\}$ and
$\Sigma_2 = \{P(x) \to (\exists y_1 \dots y_n) R(1, x, y_1) \wedge \cdots \wedge R(1, x, y_n)\}$

Clearly, Σ_1 and Σ_2 are logically equivalent. However, for the source instance $I = \{P(a)\}$, $\mathcal{M}_1$ and $\mathcal{M}_2$ yield two different canonical solutions $J_1 = \{R(1, a, y)\}$ and $J_2 = \{R(1, a, y_1), \dots, R(1, a, y_n)\}$. Let A denote the name of the first attribute of R. Then all of the three aggregate queries $\mathsf{count}(R.A)$, $\mathsf{count}(*)$, and $\mathsf{sum}(R.A)$ have the range semantics $[1, 1]$ in $\mathcal{M}_1$ and $[1, n]$ in $\mathcal{M}_2$, i.e.: $\mathcal{M}_1$ admits only one possible world and the three aggregate queries evaluate to 1 in this world. In contrast, $\mathcal{M}_2$ gives rise to many possible worlds with $\{R(1, a, y)\}$ being the smallest and $\{R(1, a, y_1), \dots, R(1, a, y_n)\}$ being the biggest. Thus, the three aggregate queries may take values between 1 and n. $\square$

In order to eliminate the dependence on the concrete syntactic representation of the st-tgds, we have defined a new normal form of st-tgds in Definition 4.1. Below, we show that we thus get a unique canonical solution also in the presence of target egds.

THEOREM 5.1. *Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a schema mapping and let $\Sigma_s^* \cup \Sigma_{st}^* \cup \Sigma_t$ be the normal form of $\Sigma_{st} \cup \Sigma_t$. Moreover, let I be a source instance and J^* the canonical solution for I under $\mathcal{M}$ obtained via an oblivious chase with Σ_{st}^* followed by a chase with Σ_t in arbitrary order. Then J^* is unique up to isomorphism. We denote J^* as $CanSol^*(I)$.*

PROOF. (Sketch) By Theorem 4.2, the normal form of the st-tgds is unique up to isomorphism. Hence, also the result of the oblivious chase with the st-tgds is unique up to isomorphism. Finally, also the chase with equivalent sets of egds produces isomorphic canonical universal instances. This property is proved by induction on the length of one of the chase sequences. $\square$

To obtain a unique range semantics of the aggregate functions min, max, count, $\mathsf{count}(*)$, sum, and avg, we therefore propose to follow the approach of [1], with the only difference that we take the unique target instance $CanSol^*(I)$ from Theorem 5.1.

We conclude this section by extending also the tractability results from [1] to schema mappings with target egds.

THEOREM 5.2. *Let $\mathcal{M} = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t \rangle$ be a schema mapping, let R be a CQ over $\mathbf{T}$ or a relation symbol in $\mathbf{T}$, let A be an attribute in R and f an aggregate operator* $\mathsf{min}(R.A)$*,* $\mathsf{max}(R.A)$*,* $\mathsf{count}(R.A)$*,* $\mathsf{count}(*)$*,* $\mathsf{sum}(R.A)$*, or* $\mathsf{avg}(R.A)$*.*

The problem of aggregate query evaluation *(i.e., given a source instance I, compute agg-certain$(f, I, Endom(I, J^*))$ with $J^* = CanSol^*(I)$) is in PTIME in each of the following cases:*

(1) R is a CQ and $f \in \{$ $\mathsf{min}(R.A)$*,* $\mathsf{max}(R.A)$*,* $\mathsf{count}(R.A)$*,* $\mathsf{count}(*)$*$\}$.*

(2) R is a CQ and $f =$ $\mathsf{sum}(R.A)$ *and A is an attribute with non-negative values only.*

(3) R is a target relation symbol and $f =$ $\mathsf{avg}(R.A)$*.*

PROOF. (Sketch) Case (1) and (2) are immediate since the considerations in [1] are not affected by the egds. For case (3), the algorithm of [1] can be extended to schema mappings with egds by applying the *Rigidity Lemma* of [11]. $\square$

6. IMPLEMENTATION

We have performed a preliminary implementation of the optimization and normalization algorithms presented in Sections 3 and 4. The chart in Figure 5 shows typical running times of the algorithm on randomly generated mappings of varying size.

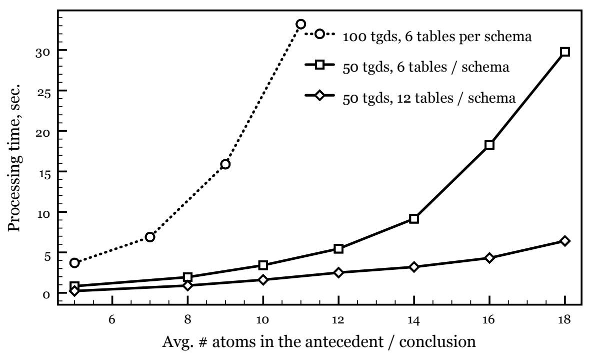


Figure 5: Performance of the optimization

In the experiments illustrated in Figure 5, random mappings consisting of 50 resp. 100 tgds with varying average number of atoms in the antecedent and conclusion were optimized. Intel MacBook with 2 GHz clock speed and 2 GB RAM was used as a test station.

The current design of the optimization tool is rather straightforward, and thus the reported performance must be seen as a baseline. We have evaluated the algorithm in several scenarios: Figure 5 presents the case where mappings are unrealistically "dense" (e.g., 100 different st-tgds relate the schemas with only 6 tables each). Such situations are disadvantageous, e.g., for the Rules 4 and 5, as for every tgd with sufficiently large antecedent, there are with high probability other tgds with "compatible" antecedents which thus may potentially imply some of its conclusion atoms. Moreover, starting from a certain size, tgds necessarily include self-joins which affects the performance of all transformation rules.

It is currently possible to optimize the mappings of up to 100 st-tgds with an average of 15 atoms in the antecedent resp. conclusion over such "dense" schemas — see the leftmost curve in Figure 5. Doubling the number of relations in the schema (and, hence, reducing the "density") resulted in a considerable performance improvement (cf. the two solid curves).

Target egds have proven to be a significant source of complexity for the algorithm. Indeed, the PROPAGATE procedure yields dependencies with a large number of atoms in the conclusion, which, due to the effect of egds, become connected via shared variables. Thus, on large "dense" schema mappings, the Rules E1 and E2 require evaluation of conjunctive queries with several hundreds of atoms, pushing the running times up the order of several minutes on the mappings with 100 egds and tgds.

However, with a more realistic correspondence between the number of dependences and the size of the schemas, a significantly better performance was achieved. For instance, with the ratio of 3 st-tgds per relation, our normalization tool could tackle mappings of up to 500 tgds with an average of 15 atoms in the antecedent resp. conclusion within 30 seconds (not shown in the chart). Similar running times were attained with a "data integration" scenario featuring a small target schema (6 relations), and a larger source schema in which the number of relations increased proportionally to the number of st-tgds, which had an average of 15 atoms in the antecedent and an average of 3 atoms in the conclusion. The same trend also holds in the presence of target egds. Thus, the normalization of up to 100 egds and 200 tgds with an average of 15 atoms per antecedent resp. conclusion was feasible within one minute.

To summarize, even without fine tuning, the presented algorithms allow to optimize schema mappings consisting of hundreds of dependencies. At the same time, there is certainly a large room for improvements left, e.g., well-known optimization techniques for evaluation and minimization of conjunctive queries [17] suggest themselves, especially for normalizing schema mappings with egds.

7. CONCLUSION

In this paper, we have initiated the study of a theory of schema mapping optimization. We have thus formulated several natural optimality criteria and we have presented a rewrite rule system for transforming any set of st-tgds into an equivalent optimal one. We have also shown that the optimal form is unique up to variable renaming. The rewrite rule system was then extended to schema mappings including target egds where we again managed to prove the uniqueness of the normal form. The normalization of schema mappings was finally applied to aggregate queries in data exchange. A prototype implementation is freely available on the web.

As future work, we envisage several extensions of our results: Above all, the case of schema mappings including target egds has to be further investigated. We have presented a rewrite rule system which carries out several natural simplifications and which yields a unique normal form. However, in contrast to the st-tgds-only case, a detailed analysis of the properties of our normal form in the presence of egds is missing. In particular, the ultimate goal of proving the optimality w.r.t. the criteria defined for the st-tgds-only case (and possibly modifying or extending these criteria in the presence of egds) is a challenging goal for future work.

We have considered two sets of dependencies as equivalent if they are *logically equivalent*. As pointed out in [10], weaker notions of equivalence such as "data exchange equivalence" and "conjunctive query equivalence" may sometimes be more appropriate. We want to extend our normal forms to these forms of equivalence.

8. REFERENCES

[1] F. N. Afrati and P. G. Kolaitis. Answering aggregate queries in data exchange. In *Proc. PODS'08*, pages 129–138. 2008.

[2] M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *Proc. PODS'04*, pages 229–240. ACM, 2004.

[3] M. Arenas, L. E. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 3(296):405–434, 2003.

[4] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

[5] P. A. Bernstein, T. J. Green, S. Melnik, and A. Nash. Implementing mapping composition. *VLDB J.*, 17(2):333–353, 2008.

[6] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *Proc. SIGMOD'07*, pages 1–12. ACM, 2007.

[7] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. STOC'77*, pages 77–90. ACM Press, 1977.

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

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

[10] R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a theory of schema-mapping optimization. In *Proc. PODS'08*, pages 33–42. ACM, 2008.

[11] R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: getting to the core. *ACM Trans. Dat. Syst.*, 30(1):174–210, 2005.

[12] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *Proc. VLDB'06*, pages 9–16. ACM, 2006.

[13] A. Hernich and N. Schweikardt. CWA-solutions for data exchange settings with target dependencies. In *Proc. PODS'07*, pages 113–122. ACM, 2007.

[14] T. Imielinski and W. L. Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[15] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[16] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. PODS'05*, pages 61–75.

[17] I. K. Kunen and D. Suciu. A scalable algorithm for query minimization. Tech. report, University of Washington, 2002.

[18] M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.

[19] L. Libkin. Data exchange and incomplete information. In *Proc. PODS'06*, pages 60–69. ACM Press, 2006.

[20] L. Libkin and C. Sirangelo. Data exchange and schema mappings in open and closed worlds. In *Proc. PODS'08*, pages 139–148. ACM, 2008.

[21] Y. Sagiv and M. Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.