# The Complexity of Evaluating Tuple Generating Dependencies*

Reinhard Pichler
Vienna University of Technology
pichler@dbai.tuwien.ac.at

Sebastian Skritek
Vienna University of Technology
skritek@dbai.tuwien.ac.at

## ABSTRACT

Dependencies have played an important role in database design for many years. More recently, they have also turned out to be central to data integration and data exchange. In this work we concentrate on tuple generating dependencies (tgds) which enforce the presence of certain tuples in a database instance if certain other tuples are already present. Previous complexity results in data integration and data exchange mainly referred to the data complexity. In this work, we study the query complexity and combined complexity of a fundamental problem related to tgds, namely checking if a given tgd is satisfied by a database instance. We also address an important variant of this problem which deals with updates (by inserts or deletes) of a database: Here we have to check if all previously satisfied tgds are still satisfied after an update. We show that the query complexity and combined complexity of these problems are much higher than the data complexity. However, we also prove sufficient conditions on the tgds to reduce this high complexity.

## Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Relational databases*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

## 1. INTRODUCTION

*Dependencies* are a classical tool in database design to express integrity constraints on databases [2, 5, 10]. Recently, renewed interest in dependencies has emerged in the vivid research areas of data integration [23, 29] and data exchange [11, 26], where dependencies are used to define *schema mappings*. Schema mappings are high-level specifications which describe the relationship between two database schemas. They are usually given in the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S}$ denotes the source schema, $\mathbf{T}$ denotes the target schema, and $\Sigma$ is a set of dependencies specifying the relationship between the source and target schema. In this paper, we restrict ourselves to *tuple generating dependencies (tgds)* which, in their original form, are first-order (FO) formulae

$$\forall \vec{x}\big(\varphi(\vec{x}) \rightarrow \exists \vec{y}\, \psi(\vec{x}, \vec{y})\big)$$

where $\varphi$ and $\psi$ are conjunctive queries (CQs) and all variables in $\vec{x}$ actually do occur in $\varphi(\vec{x})$. Intuitively, such a tgd expresses the condition that, if certain tuples are present in the database (namely those tuples which are needed to satisfy the CQ $\varphi$) then certain other tuples must be present as well (such that $\psi$ is also satisfied).

In [13], it was shown that tgds are not powerful enough to express the composition of schema mappings. The authors thus introduced *second-order (SO) tgds*, which allow existential quantification over function symbols (for details, see Section 4). Many important properties of SO tgds were proved in [13]; above all, SO tgds are closed under composition. A related extension of tgds was introduced in [16], namely *nested tgds* (see Section 5), where the existentially quantified function symbols stand for relations. In the first place, nested tgds were designed for nested relations and XML data. However, also their restriction to the relational case has been considered [36].

Many complexity results related to schema mappings and dependencies have been presented for problems like query answering in data integration or data exchange [1, 11], the data exchange problem (i.e., given an instance of the source schema, find an instance of the target schema s.t. all dependencies are fulfilled) [11], core computation in data exchange (i.e., finding a minimal solution to the data exchange problem) [12, 19], etc. The usual notion of complexity thus applied is the *data complexity*, i.e.: the schemas and dependencies are considered as fixed while only the database instances are allowed to vary. However, as more and more progress is made in the area of automatic generation and processing of schema mappings [6] we shall have to deal with sets of dependencies of ever increasing size. Hence, also the *query complexity* and the *combined complexity* (where only the dependencies resp. both, the dependencies and the data are allowed to vary) should be studied to arrive at a complete picture. In fact, this study has already been initiated in [27], where a sharp increase of the complexity of several important problems in the area of data exchange was identified when combined complexity is considered rather than data complexity, e.g.: The problem of checking if a data exchange problem has a solution rises from PTIME-membership to EXPTIME-completeness.

The goal of this paper is to continue the investigation of the query complexity and combined complexity of fundamental problems related to dependencies. More specifically, we analyze the complexity of the evaluation problem (also referred to as *model checking*) of the above mentioned three kinds of tuple generating dependencies. Note that the evaluation of tgds is a central building block of the

*This work was supported by the Austrian Science Fund (FWF), project P20704-N18 and the Vienna Science and Technology Fund (WWTF), project ICT08-032.


*ICDT 2011*, March 21–23, 2011, Uppsala, Sweden.

*chase procedure* for computing a solution to a data exchange problem [11]. Actually, as long as we only have to deal with source-to-target tgds (s-t tgds), the chase procedure could also be carried out without ever checking if the tgds are satisfied. Instead, we could compute all solutions over the source instance for the antecedent $\varphi(\vec{x})$ of every s-t tgd (i.e., we compute substitutions $\lambda$ on the variables $\vec{x}$, s.t. all atoms of $\varphi(\vec{x}\lambda)$ are in the source instance) and introduce atoms corresponding to the conclusion of the tgd in the target instance (i.e., we introduce the atoms of $\psi(\vec{x}\lambda, \vec{x}\mu)$, where $\mu$ sends the variables in $\vec{y}$ to fresh labelled nulls). Clearly, this form of s-t chase (referred to as "oblivious" chase in [24]) may possibly introduce more atoms in the target instance than needed to satisfy all s-t tgds. But in some cases, this is even desired to arrive at an intuitive definition of the semantics of query answering for queries that are more complex than just CQs, see e.g., [3, 4, 30]. In contrast, if there is no strict separation of the relation symbols occurring in the antecedents and conclusions of the tgds (like in case of target tgds or tgds in peer data management systems [22]), then the model checking of tgds may be vital to ensure termination of the chase. This is illustrated by the following example (which is taken from [11]).

EXAMPLE 1.1. Consider a schema with predicates $Dept(d, m)$ and $Emp(e, d)$, where $Dept$ contains information on departments $d$ and managers $m$; $Emp(e, d)$ contains information on employees $e$ and departments $d$. Consider the following tgds:

$\tau_1 : Dept(d, m) \rightarrow Emp(m, d)$
$\tau_2 : Emp(e, d)) \rightarrow \exists m Dept(d, m)$.

Let $J = \{Emp(bob, sales)\}$ be an instance with a single atom. Then the chase (with tgd model checking after each step) yields an instance of the form $J^* = \{Emp(bob, sales), Dept(sales, v), Emp(v, sales)\}$, where $v$ is a labelled null (which stands for the unknown manager of the sales-department).

In contrast, without tgd model checking, the chase would not terminate. Indeed, the last atom $Emp(v, sales)$ introduced in $J^*$ helps to satisfy the antecedent of $\tau_2$. Without model checking, we would thus introduce a new atom $Dept(sales, v')$, which in turn helps to satisfy the antecedent of $\tau_1$. We would thus introduce an atom $Emp(v', sales)$ which again causes tgd $\tau_2$ to fire, etc. □

Similarly, in update exchange, it is crucial to check if some tgd is violated every time a database instance has been modified [20, 28]. Of course, if the relation symbols in the antecedents and conclusions of the tgds are strictly separated and if we only allow inserts, then we could proceed by an oblivious chase as discussed above for the s-t chase. However, if we have no such separation of the relation symbols or if we also allow deletes, then tgd model checking is absolutely indispensable as the following example illustrates:

EXAMPLE 1.2. Suppose that $Emp(e, d)$ from Example 1.1 is in the source schema and $Dept(d, m)$ is in the target schema and consider only one tgd, namely $\tau_2$ (which is now an s-t tgd). Suppose that we have a source instance $I = \{Emp(bob, sales)\}$ and target instance $J = \{Dept(sales, v), Dept(hr, alice)\}$.

If a delete request is issued against $J$, it is important to check if this leads to the violation of an s-t tgd. Clearly, if the second tuple in $J$ is deleted, no tgd is violated and no action is required. In contrast, if the first tuple is deleted, then $\tau_2$ is violated. □

In Example 1.2, after deletion of the first tuple in $J$, it would be counter-intuitive to cure the database by chasing $I$ with $\tau_2$, since this would simply restore the deleted tuple. Instead, [28] proposes a "backward chase" that deletes appropriate tuples from the source instance. This backward chase procedure relies on two main ingredients: user intervention (there may exist several tuples whose deletion would cure the violation of a tgd) and tgd model checking (every deletion of a tuple may itself cause the violation of a tgd).

For our analysis of the complexity of the evaluation problem of tgds it is therefore important to study also a variant of this problem which deals with updates (by inserts or deletes) of a database: Here we have to check if all previously satisfied tgds are still satisfied after an update. We shall show that the query complexity or combined complexity of tgd model checking is dramatically higher than the data complexity: It rises from PTIME-membership to $\Pi_2 P$-completeness (for FO tgds) and from NP-completeness to NEXPTIME-completeness (for SO tgds and nested tgds). For the update variants we show that the additional knowledge of how the current database was obtained via inserts/deletes from a previous database satisfying all tgds does not help to decrease the complexity of tgd model checking. However, for all kinds of tgds studied here, we prove sufficient criteria to reduce the high complexities.

**Organization of the paper and summary of results.**

- *FO tgds.* Since FO tgds are a special case of FO formulae, the data complexity of model checking is clearly in PTIME. In Section 3, we show that this problem becomes $\Pi_2 P$-complete for query complexity and combined complexity. To search for computationally less expensive fragments, we apply the notion of treewidth to this problem. We show that the complexity can thus be pushed down to coNP-completeness. For the query complexity, we even get tractability via an appropriate criterion based on the treewidth.

- *SO tgds.* The data complexity of evaluating SO tgds has been shown to be NP-complete [13]. In Section 4, we show that the complexity rises to NEXPTIME-completeness when query complexity or combined complexity is considered. A reduction of this high complexity is obtained via an easy to check criterion based on some ordering of the universally quantified first-order variables. The benefit of this criterion is twofold. On the one hand, it yields a decrease of the complexity from NEXPTIME- to PSPACE-completeness. On the other hand, we present a reduction of SO tgds to FO formulae if this criterion is fulfilled. Hence, conditions for further decreasing the complexity can be obtained from the research on tractable fragments of FO formulae, see e.g. [14, 18].

- *Nested tgds.* Similarly to SO tgds, the data complexity of evaluating nested tgds [16] is NP-complete. Likewise, we prove in Section 5 the NEXPTIME-completeness for the query and combined complexity. Again, this high complexity is reduced to PSPACE-completeness via a criterion based on an ordering of the universally quantified first-order variables. The resulting fragment includes the nested tgds over purely relational schemas as studied in [36]. We thus prove the PSPACE-completeness also for this variant of tgds.

- *Update variants.* In Sections 3 – 5, the complexity analysis of the three kinds of tgds is extended to the update variants of the tgd evaluation problem. We shall show that in all cases the complexity of the update variant is as high as for the basic model checking problem even if the update consists in the insertion or deletion of a single tuple.

- *Technical tools.* The design of fixed-parameter algorithms has received vivid research interest in recent years [32]. A well-studied parameter in this area is the treewidth. In order to establish fragments of FO tgd model checking with lower complexity, we develop a new dynamic programming algorithm for counting the solutions of conjunctive queries with bounded treewidth (see Section 3). New techniques are required to overcome obstacles that were not present in a related problem [35]. For the complexity analysis of SO tgds in Section 4, we extend the generic PSPACE-complete problem of QBFs (quantified Boolean formulae) to SO-

QBFs (Second-Order QBFs) and show the NEXPTIME-completeness of this new class of Boolean formulae. They may now serve as a useful tool for further NEXPTIME-completeness proofs as the application to SO tgd model checking clearly illustrates.

Due to lack of space, most proofs in this paper are only sketched or even omitted. Detailed proofs of all results will be provided in the full paper.

## 2. PRELIMINARIES

In this section, we recall some basic notions and formally define the decision problems studied in this paper.

**Schemas and instances.** A *relation schema* $R_i(A_1, \ldots, A_{k_i})$ consists of a relation symbol $R_i$ of a fixed arity $k_i$ and with an assigned sequence of $k_i$ attributes $(A_1, \ldots, A_{k_i})$. By slight abuse of notation, we often identify the relation schema with the relation symbol (in particular, if the attributes are clear from the context or do not matter). A *database schema* (or simply a *schema*) $\mathbf{R} = \{R_1, \ldots, R_n\}$ is given by a finite set of relation schemas. An *instance* over a database schema $\mathbf{R}$ consists of a relation for each relation schema in $\mathbf{R}$. 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 *labelled nulls*. Two labelled 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 their combination $\langle I, J \rangle$ is an instance of the schema $\langle \mathbf{S}, \mathbf{T} \rangle$.

Let $J$, $J'$ be instances. A *homomorphism* $h\colon J \to J'$ is a mapping $dom(J) \to dom(J')$, s.t. (1) whenever $R(\vec{x}) \in J$, then $R(h(\vec{x})) \in J'$, and (2) for every constant c, $h(c) = c$. If such an $h$ exists, we write $J \to J'$. A homomorphism $h'$ is an *extension* of a homomorphism $h$ if, whenever $h(x)$ is defined, also $h'(x)$ is defined and $h'(x) = h(x)$ holds.

**Tgds.** A *first-order tuple generating dependency (FO tgd* or simply *tgd)* over a schema $\mathbf{R}$ is an FO formula $\tau = \forall \vec{x} \big(\varphi(\vec{x}) \to \exists \vec{y}\, \psi(\vec{x}, \vec{y})\big)$ where both, antecedent $\varphi$ and conclusion $\psi$ are conjunctive queries (CQs) over the relational symbols from $\mathbf{R}$ s.t. all variables in $\vec{x}$ actually do occur in $\varphi(\vec{x})$. If $\mathbf{R} = \langle \mathbf{S}, \mathbf{T} \rangle$ and $\varphi$ (resp. $\psi$) uses only relation symbols from $\mathbf{S}$ (resp. $\mathbf{T}$), then $\tau$ is called a *source-to-target tgd (s-t tgd)*.

A tgd $\tau = \forall \vec{x} \big(\varphi(\vec{x}) \to \exists \vec{y}\, \psi(\vec{x}, \vec{y})\big)$ is satisfied over some instance $J$ of $\mathbf{R}$, if the answer to $\varphi(\vec{x})$ over $J$ is a subset of the answer to $\exists \vec{y}\, \psi(\vec{x}, \vec{y})$. In terms of homomorphisms, $\tau$ is satisfied by $J$, if for every homomorphism $h\colon \varphi \to J$, there exists an extension $h'$ of $h$, s.t. $h'\colon \psi \to J$. In this case, we write $J \models \tau$.

Other types of tgds studied in this paper are *second-order tuple generating dependencies (SO tgds)* [13] and *nested tuple generating dependencies (nested tgds)* [16]. We recall their definitions in Section 4 and Section 5, respectively.

**Problem definitions.** The main theme of this paper is the *evaluation problem* (also referred to as *model checking problem*) of tgds. Formally, for a schema $\mathbf{R}$, we define the problem as follows:

| $\mathcal{L}$ **tgd model checking** | | |
|---|---|---|
| *Input:* | *TGD:* | Set $\Sigma$ of $\mathcal{L}$ tgds over schema $\mathbf{R}$. |
| | *Data:* | Instance $K$ over schema $\mathbf{R}$. |
| *Question:* | Does $K \models \Sigma$ hold? | |

Thereby $\mathcal{L} \in \{\text{FO, SO, nested}\}$. An important special case of this problem arises if we consider *source-to-target tgds* (s-t tgds), where the schema is of the form $\langle \mathbf{S}, \mathbf{T} \rangle$, s.t. the antecedents of tgds may only contain predicate symbols from $\mathbf{S}$ and the conclusions are based on $\mathbf{T}$. Actually, SO tgds and nested tgds have only been considered in source-to-target settings anyway (see [13, 16, 36]).

Tgds often occur in settings where they are used to create new tuples (by the chase) during the bootstrap phase of a (data exchange) system. But of course, checking if a set of tgds is satisfied is also important during the runtime of a system when updates may occur (e.g., in collaborative data management systems or peer data management systems). We therefore also consider the following update variants of the above model checking problem:

| $\mathcal{L}$ **tgd delete-update model checking** | | |
|---|---|---|
| *Input:* | *TGD:* | Set $\Sigma$ of $\mathcal{L}$ tgds over schema $\mathbf{R}$. |
| | *Data:* | Instance $K$ over schema $\mathbf{R}$, s.t. $K \models \Sigma$, set $U = \{t_1, \ldots, t_\ell\} \subseteq K$. |
| *Question:* | Does $K \setminus U \models \Sigma$ hold? | |

| $\mathcal{L}$ **tgd insert-update model checking** | | |
|---|---|---|
| *Input:* | *TGD:* | Set $\Sigma$ of $\mathcal{L}$ tgds over schema $\mathbf{R}$. |
| | *Data:* | Instance $K$ over schema $\mathbf{R}$, s.t. $K \models \Sigma$, set $U = \{t_1, \ldots, t_\ell\}$ of tuples. |
| *Question:* | Does $K \cup U \models \Sigma$ hold? | |

Again, $\mathcal{L} \in \{\text{FO, SO, nested}\}$. The restriction to source-to-target tgds (where the schema $\mathbf{R}$ is of the form $\langle \mathbf{S}, \mathbf{T} \rangle$) constitutes an important special case also for these update variants. Of course, in a source-to-target setting, the delete-update (resp. insert-update) model checking is only of interest if a tuple is deleted from the target instance (resp. inserted into the source instance).

When studying the complexity of the problems defined above, we arrive at the *combined complexity* [37] of the problems. In addition, we also want to study the *query complexity* of these problems, i.e.: the data is considered as fixed and the input only consists in the set of tgds. Unless explicitly stated otherwise, all complexity results in this paper refer to both, the query complexity and the combined complexity. Moreover (in case of FO-tgds), our results apply to an arbitrary schema $\mathbf{R}$ as well as to a source-to-target setting with schema $\mathbf{R} = \langle \mathbf{S}, \mathbf{T} \rangle$. Clearly, the query complexity cannot be higher than the combined complexity and the source-to-target setting cannot be more complex than the general case. Hence, in the proofs of our completeness results, we shall prove the membership for the combined complexity with an arbitrary schema and we prove the hardness for the query complexity with a schema $\langle \mathbf{S}, \mathbf{T} \rangle$.

We have already recalled that terms in a database instance can be constants or variables (labelled nulls). In data exchange, one usually assumes that labelled nulls are allowed in the target instance, while the source instance contains constants only. In our complexity analysis, we make no specific assumption on the nature of the terms. In particular, our membership results hold without any restrictions on the labelled nulls while our hardness proofs work without making use of any labelled nulls.

# 3. FIRST-ORDER TGDS

We start our complexity analysis with FO tgds. Since FO tgds are made up of two CQs and CQ evaluation is NP-complete, we cannot expect a lower complexity for the tgd evaluation. Indeed, we shall prove in this section that tgd evaluation is even one level higher in the polynomial hierarchy, i.e., $\Pi_2 P$-complete. We shall also show how this complexity can be decreased by an appropriate application of the concept of treewidth.

THEOREM 3.1. *FO tgd model checking is $\Pi_2 P$-complete (both, query and combined complexity). The problem remains $\Pi_2 P$-complete even if $\Sigma$ contains a single tgd only.*

PROOF SKETCH. The *membership* is proved by devising a $\Sigma_2 P$-algorithm for the co-problem: Let $\tau = \forall \vec{x}\big(\varphi(\vec{x}) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})\big)$ be a tgd and let $J$ be an instance. To check that $J \not\models \tau$ holds, we

(1) guess a mapping $h\colon \vec{x} \rightarrow dom(J)$,
(2) check that $h$ defines a homomorphism $\varphi(\vec{x}) \rightarrow J$, and
(3) check (by an NP-oracle) that there exists no extension $h'\colon \psi(\vec{x}, \vec{y}) \rightarrow J$ of $h$.

The *hardness* (of query complexity for the special case of source-to-target dependencies) is proved by a reduction from the $\Pi_2 P$-complete $\forall$-QSAT$_2$ problem. As fixed data $\langle I, J\rangle$, we consider

$I = \{P(1,0), P(0,1)\}$ and
$J = \{Q(1,0), Q(0,1), C(0,0,1), C(0,1,0), C(1,0,0), C(0,1,1), C(1,0,1), C(1,1,0), C(1,1,1)\}$.

Now let an arbitrary instance of $\forall$-QSAT$_2$ be given by the quantified Boolean formula $F = \forall(x_1, \ldots, x_k)\exists(y_1, \ldots, y_\ell)(C_1 \wedge \ldots \wedge C_n)$ with $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$ for $i \in \{1, \ldots, n\}$ (clearly, the restriction of the matrix to 3-CNF is w.l.o.g.). From this, we construct a single tgd $\tau = \forall \vec{x}(\varphi(\vec{x}) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y}))$ with

$\varphi(\vec{x}) = \bigwedge_{i=1}^{k} P(x_i, \bar{x}_i)$ and
$\psi(\vec{x}, \vec{y}) = \bigwedge_{i=1}^{\ell} Q(y_i, \bar{y}_i) \wedge \bigwedge_{i=1}^{n} C(l^*_{i,1}, l^*_{i,2}, l^*_{i,3})$,

where $l^*_{i,j}$ is defined as $l^*_{i,j} = x_\gamma$ if $l_{i,j} = x_\gamma$, and $l^*_{i,j} = \bar{x}_\gamma$ if $l_{i,j} = \neg x_\gamma$, resp. $l^*_{i,j} = y_\gamma$ if $l_{i,j} = y_\gamma$, and $l^*_{i,j} = \bar{y}_\gamma$ if $l_{i,j} = \neg y_\gamma$. By slight abuse of notation, we thus use $x$ and $y$ to denote both propositional variables in $F$ and FO variables in $\tau$.

Clearly, this reduction is feasible in LOGSPACE. It remains to prove that $F$ is satisfiable iff $\tau$ is satisfied in $\langle I, J\rangle$. □

THEOREM 3.2. *FO tgd delete-update model checking and FO tgd insert-update model checking are $\Pi_2 P$-complete (both, query and combined complexity). They remain $\Pi_2 P$-complete even if $\Sigma$ contains a single tgd and $U$ contains a single tuple only.*

PROOF IDEA. *Membership* for both problems follows immediately from the $\Pi_2 P$-membership of tgd model checking. The idea of the *hardness* proofs is to construct a setting where the tgd is trivially satisfied, but after insertion/removal of a single tuple the tgd is only satisfied iff some $\forall$-QSAT$_2$ formula is satisfiable. □

Recently, tgds have been recognized as a useful formalism to express integrity constraints on RDF graphs [31]. In this case, all relation symbols have arity 2. Now the question arises if the $\Pi_2 P$-completeness also holds in this restricted case. Below, we give a positive answer to this question.

THEOREM 3.3. *FO tgd model checking remains $\Pi_2 P$-complete (both, query and combined complexity) even if all relation symbols have arity 2 and $\Sigma$ contains a single tgd only.*

## 3.1 Fragments with lower complexity

We now shift our attention to the search for fragments of the FO tgd model checking problem and its update variants with lower complexity. A very natural restriction is the restriction to *full tgds*, i.e., FO tgds with no existentially quantified variables in the conclusion. It can be easily checked that in this case, all $\Pi_2 P$-complete problems considered above become coNP-complete: Indeed, the coNP-membership follows immediately from the $\Pi_2 P$-membership proof of Theorem 3.1, since the check in step (3) of that algorithm no longer requires an NP-oracle; it is in PTIME for full tgds. The coNP-hardness follows immediately from the NP-hardness of Boolean Conjunctive Query evaluation, i.e.: we just have to consider a tgd $\tau$, whose conclusion will never be fulfilled. Then $\tau$ is satisfied by a database instance $K$ iff the CQ in the antecedent of $\tau$ has no solution over $K$.

An interesting approach with a wide range of applications to intractable problems comes from *parameterized complexity* theory [9, 15]. It is based on the following observation: Many hard problems become tractable if some problem parameter is bounded by a fixed constant. One important parameter of graphs and, more generally, of finite structures is the *treewidth*, which measures the "tree-likeness" of a graph or a structure. Treewidth has also been successfully applied to conjunctive queries [7], and was even shown to be the most general graph based characterization of tractable conjunctive queries [21].

We first recall the definition of the treewidth of CQs: Let $\varphi = L_1 \wedge \cdots \wedge L_n$ be a CQ with atoms $L = \{L_1, \ldots, L_n\}$ and variables $V$. A *tree decomposition* $\mathcal{T}$ of $\varphi$ is a pair $\langle T, (A_t)_{t \in T}\rangle$ where $T$ is a tree and each $A_t$ is a subset of $L \cup V$ with the following properties: (1) Every $a \in L \cup V$ is contained in some $A_t$. (2) For every variable $x \in V$ and atom $L_i$, if $x$ occurs in $L_i$ then there exists some node $t \in T$ with $\{x, L_i\} \subseteq A_t$. (3) For every $a \in L \cup V$, the set $\{t \mid a \in A_t\}$ induces a subtree of $T$.

Condition (3) is referred to as the *connectedness condition*; the sets $A_t$ are called the *bags* (or *blocks*) of $\mathcal{T}$.

The *width* of a tree decomposition $\langle T, (A_t)_{t \in T}\rangle$ is $\max\{|A_t| \mid t \in T\} - 1$. The *treewidth* of $\varphi$, denoted as $tw(\varphi)$, is the minimal width of all tree decompositions of $\varphi$. Alternatively, $tw(\varphi)$ can be defined by defining the treewidth for graphs only and by associating a graph with each CQ. Our definition of treewidth is obtained by associating the so-called *incidence graph* with each CQ. There are also other possibilities like the so-called primal graph or the dual graph. But the treewidth defined via the incidence graph yields a strictly bigger tractable fragment than any of the other notions. For a discussion, see e.g., [35].

If the treewidth of the CQs under consideration is bounded by a fixed constant, then many otherwise intractable problems become tractable, like model checking of Boolean CQs or testing if some tuple is in the answer to a non-Boolean CQ. Moreover, the set of all answers to a CQ can be computed in output-polynomial time in case of bounded treewidth.

In the remainder of this section, we investigate how the complexity of model checking of tgds is affected if we restrict the treewidth of the CQs in an FO tgd. Note that the CQ $\varphi(\vec{x})$ in the $\Pi_2 P$-hardness proof of Theorem 3.1 has treewidth 1, while $\psi(\vec{x}, \vec{y})$ in general, has unbounded treewidth. Our first question is therefore, what improvement of the complexity do we get if also the treewidth of the conclusion is restricted by a constant. It turns out that then the complexity decreases, but it is still coNP-complete.

THEOREM 3.4. *Suppose that we only consider FO tgds $\tau = \forall \vec{x}\big(\varphi(\vec{x}) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})\big)$ where both $tw(\varphi(\vec{x}))$ and $tw(\psi(\vec{x}, \vec{y}))$ are bounded by some constant. Then FO tgd model checking is*

*coNP-complete (both, query and combined complexity). It remains coNP-complete even if $\Sigma$ contains a single tgd only.*

PROOF SKETCH. For the *coNP-membership*, observe that the check in step (3) of the algorithm in the coNP-membership proof of Theorem 3.1 is now in PTIME (rather than NP-complete). This is due to the bounded treewidth of $\psi(\vec{x},\vec{y})$ and, therefore, also of $\psi(h(\vec{x}),\vec{y})$. Hence, checking $J \not\models \tau$ is in NP.

The *coNP-hardness* is shown by reduction from the VALIDITY-problem. Note that the VALIDITY problem remains coNP-hard even if the Boolean formulae are restricted to 3-DNF and each variable occurs at most three times (see [33], Proposition 9.3).

As fixed pair of database instances $\langle I, J\rangle$, we choose

$I = \{Q(1,1,1,0,0,0),\, Q(0,0,0,1,1,1)\}$ and
$J = \{R(0,0,0,0),\, R(0,0,1,0),\, R(0,1,0,0),\, R(0,1,1,0),$
$R(1,0,0,0),\, R(1,0,1,0),\, R(1,1,0,0),\, R(1,1,1,1),$
$S(1,1,1),\, S(0,1,1),\, S(1,0,1),\, S(0,0,0),\, T(1)\}.$

Now let $F = C_1 \vee \cdots \vee C_n$ with $C_i = (l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$ be an arbitrary Boolean formula in 3-DNF, s.t. each variable occurs at most three times in $F$. Let $Z = \{z_1, \ldots, z_m\}$ denote the set of variables in $F$. We construct the tgd $\tau = \forall\vec{x}(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x},\vec{y}))$ with $\vec{x} = \{x_{1,1}, x_{1,2}, x_{1,3}, \bar{x}_{1,1}, \bar{x}_{1,2}, \bar{x}_{1,3}, \ldots, x_{m,1}, x_{m,2}, x_{m,3}, \bar{x}_{m,1}, \bar{x}_{m,2}, \bar{x}_{m,3}\}$ (i.e., we introduce 6 variables for every variable $z_i \in Z$) and $\vec{y} = \{y_1, \ldots, y_n, y'_2, \ldots, y'_n\}$ (i.e., we introduce 2 variables for every implicant except for the first one) and

$\varphi(\vec{x}) = \bigwedge_{i=1}^{m} Q(x_{i,1}, x_{i,2}, x_{i,3}, \bar{x}_{i,1}, \bar{x}_{i,2}, \bar{x}_{i,3})$

$\psi(\vec{x},\vec{y}) = \bigwedge_{i=1}^{n} R(l^*_{i,1}, l^*_{i,2}, l^*_{i,3}, y_i) \wedge \bigwedge_{i=2}^{n} S(y_{i-1}, y_i, y'_i) \wedge T(y'_n)$

To define $l^*_{i,j}$ ($i \in \{1,\ldots,n\}$, $j \in \{1,2,3\}$), we transform the formula $F$ into $F'$ by replacing the first occurrence of each $z_i$ by a new variable $z_{i,1}$, and the second and third occurrence of $z_i$ (if they exist) by $z_{i,2}$ and $z_{i,3}$, respectively. We denote the resulting literals in $F'$ as $l'_{i,j}$. Then $l^*_{i,j} = x_{\gamma,s}$ if $l'_{i,j} = z_{\gamma,s}$ and $l^*_{i,j} = \bar{x}_{\gamma,s}$ if $l'_{i,j} = \neg z_{\gamma,s}$.

The intuition of this reduction is as follows: The $Q$-atoms ensure that the (at most 3) copies of each variable $z_i$ are assigned the same truth value and that $z_i$ and $\bar{z}_i$ are assigned dual values. The $R$-predicate is now 4-ary: The first 3 arguments denote the possible truth values of the literals in each implicant. The 4-th argument gives the resulting truth value of the implicant. The $S$-atoms encode the OR-operation. They are needed to "compute" the truth value of $C_1 \vee \cdots \vee C_n$ by successive binary OR-operations. The $T$-atom holds the desired truth value 1 (corresponding to true) for the evaluation of $F$.

This reduction is obviously feasible in LOGSPACE. It remains to show that $\langle I, J\rangle \models \tau$ indeed holds iff $F$ is valid. Moreover, we have to verify that the treewidth of $\varphi$ and $\psi$ has a bound that is independent of the particular formula $F$. In fact, we can even show that the treewidth of both $\varphi$ and $\psi$ is 1. □

Just as in the unbounded case, the complexity of the update-variants of the problem remains the same. The following theorem can be proven by modifying the proof of Theorem 3.4 analogously to the proof of Theorem 3.2.

THEOREM 3.5. *Assume the same setting as in Theorem 3.4. Then FO tgd delete-update model checking and FO tgd insert-update model checking are coNP-complete (both, query and combined complexity). Both problems remain coNP-complete even if $\Sigma$ contains a single tgd and $U$ contains a single tuple only.*

To reach tractability at least for the query complexity, additional restrictions are required. To this end, we define the treewidth of a tgd $\tau = \forall\vec{x}\big(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x},\vec{y})\big)$ as $tw(\tau) = tw(\varphi(\vec{x}) \wedge \psi(\vec{x},\vec{y}))$, i.e., we treat a tgd like a single CQ rather than two separated CQs. This gives us a sufficient criterion to ensure tractability of the query complexity. The combined complexity under this criterion is left as an open problem for future research.

THEOREM 3.6. *Suppose that we only consider FO tgds $\tau = \forall\vec{x}\big(\varphi(\vec{x}) \rightarrow \exists\vec{y}\ \psi(\vec{x},\vec{y})\big)$ where $tw(\tau) = tw(\varphi(\vec{x}) \wedge \psi(\vec{x},\vec{y}))$ is bounded by some constant. Then FO tgd model checking is in PTIME (query complexity only).*

The proof of this theorem is based on the following observation: It is convenient to denote FO-tgds as $\varphi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}\,\psi(\vec{x},\vec{y})$, s.t. $\vec{x}$ denotes the variables occurring both in the antecedent and conclusion, while the variables in $\vec{z}$ occur in the antecedent only. Now let $Q_1 = ans_1(\vec{x}) \leftarrow \varphi(\vec{x},\vec{z})$ and $Q_2 = ans_2(\vec{x}) \leftarrow \psi(\vec{x},\vec{y})$, i.e.: we consider both $Q_1$ and $Q_2$ as CQs with free variables $\vec{x}$. Clearly, for any instance $J$, we have $J \models \tau$ iff $Q_1(J) \subseteq Q_2(J)$, i.e.: all tuples in the answer to $Q_1$ are also contained in the answer to $Q_2$. Despite the bounded treewidth of $\tau$, $Q_i(J)$ may contain exponentially many tuples. Hence, we cannot afford to compute $Q_i(J)$ explicitly. However, for $Q'_2 = ans'_2(\vec{x}) \leftarrow \psi(\vec{x},\vec{y}) \wedge \varphi(\vec{x})$, we observe the following equivalences: $Q_1(J) \subseteq Q_2(J)$ iff $Q_1(J) = Q'_2(J)$ iff $|Q_1(J)| = |Q'_2(J)|$. Hence, model checking of FO tgds comes down to counting the number of solutions to conjunctive queries with bounded treewidth.

It remains to prove that the query complexity of the latter task is in PTIME. Indeed, this can be shown by an extension of the dynamic programming algorithm in [35] for counting the number of models of propositional formulae with bounded treewidth. It will turn out that the extension of the algorithm in [35] to CQs with no existentially quantified variables is quite straightforward. However, in order to handle also CQs without this restriction, a more sophisticated machinery will be required.

## 3.2 Efficient counting of CQ-answers

We now first present the ideas of a PTIME algorithm (similar to [35]) that computes the number of solutions of a CQ without existentially quantified variables. Then we sketch how this algorithm can be extended so as to handle also CQs with existentially quantified variables, which finally gives the proof of Theorem 3.6

For the algorithm we need the notion of a *nice tree decomposition* [25] $\langle T, (A_t)_{t\in T}, r\rangle$, where $r$ is a dedicated root node and $\langle T, (A_t)_{t\in T}\rangle$ is a tree decomposition that satisfies the following additional conditions: (1) Every node $t \in T$ has at most two children; (2) If some node $t$ has two children $t_1$ and $t_2$, then $A_t = A_{t_1} = A_{t_2}$ (*join node*); (3) If $t$ has exactly one child $t'$, then either $A_t = A_{t'} \cup \{x\}$ (*introduce node*) or $A_t = A_{t'} \setminus \{x\}$ (*forget node*), where $x$ is either a variable or an atom. Nice tree decompositions can be computed from arbitrary tree decompositions in PTIME without increasing the width [25].

In the following, we denote with $T_t$ the subtree of a tree decomposition $T$ rooted at $t \in T$, with $A^a_t$ (resp. $A^v_t$) the set of labels in $A_t$ representing atoms (resp. variables), $V_t = \bigcup_{t'\in T_t} A^v_{t'}$ and $At_t = \bigcup_{t'\in T_t} A^a_{t'}$. Now suppose that we want to compute the number of different solutions of some CQ $\varphi$ over some instance $I$, and let $T$ be a nice tree decomposition of width $w-1$ of $\varphi$. We do this in a bottom up traversal of the tree. Obviously, if we could store for each node $t \in T$ all assignments $\mu\colon V_t \rightarrow dom(I)$ s.t. for every atom $C_i \in \varphi\colon \mu(C_i) \in I$, we would just have to count the number of different mappings $\mu$ in the root node of $T$. However, as there may be exponentially many (w.r.t. the size of the query) such mappings, this is not allowed if we aim at tractability w.r.t. the combined complexity. Therefore, we only store a smaller "footprint" of

every such $\mu$ at each node $t$, consisting of the variable assignment $\mu$ restricted to $A_t^v$, and, in addition, for every $C_i \in A_t^a$ the set $I_i$ of those atoms from $I$ to which $C_i$ can be mapped by any extension of $\mu$. For the formalization of these sets $I_i$, we introduce an operator $\hat{\pi}$ as follows: For $\mu$ and $C_i$ as above, we define:

$$\hat{\pi}(\mu(C_i)) = \{Q \in I \mid Q = R(s'_1, \ldots, s'_\gamma), C_i = R(s_1, \ldots, s_\gamma) \text{ and for all } s_j \in \{s_1, \ldots, s_\gamma\} \cap V_t \colon \mu(s_j) = s'_j\}$$

We further define the footprint of $\mu$, denoted by $fpr(\mu)$, as $fpr(\mu) = (\alpha, I_1, \ldots, I_\ell)$, where $\alpha$ is a variable assignment on $A_t^v$, $\ell = |A_t^a|$, and $I_i \subseteq I$ for every $i \in \{1, \ldots, \ell\}$, and the following conditions (1) and (2) hold: (1) $\alpha$ is the restriction of $\mu$ to the variables in $A_t^v$ and (2) for all $C_i \in A_t^a$, $I_i = \hat{\pi}(\mu(C_i))$.

Clearly, given $\mu$, $fpr(\mu)$ is uniquely defined for a given node $t \in T$ and can be easily computed. It is, however, easy to see that different assignments $\mu$ may possess the same footprint, i.e. $\mu \neq \mu'$ but $fpr(\mu) = fpr(\mu')$. Given a footprint $\theta = (\alpha_x, I_1, \ldots, I_\ell)$ and a node $t \in T$, we therefore define $N(t, \alpha, I_1, \ldots, I_\ell)$ as the set of all mappings $\mu \colon V_t \to dom(I)$ s.t. (1) $fpr(\mu) = \theta$ and (2) for all $C_i \in At_t$ s.t. $Var(C_i) \subseteq V_t$, it holds that $\mu(C_i) \in I$.

Then, because of (2), if $r$ denotes the root node of $T$, it follows immediately that $\bigcup_{(\alpha, I_1, \ldots, I_\ell)} N(r, \alpha, I_1, \ldots, I_\ell)$ contains exactly all the solutions to $\varphi$. Again, for complexity reasons, it is not allowed to store any such set $N(\eta)$ with $\eta = (t, \alpha, I_1, \ldots, I_\ell)$ explicitly. Instead, as we are only interested in the number of solutions, it suffices to store $\eta$ together with the cardinality of $N(\eta)$. Hence we can use footprints to identify sets of assignments $V_t \to dom(I)$, and can store efficiently how many different assignments have a certain footprint. For the correctness of the algorithm, it is crucial that no assignment $\mu$ is counted twice. This is guaranteed by the following lemma, which follows immediately from the fact that $fpr(\mu)$ is uniquely defined.

LEMMA 3.1. *Given* $(t, \alpha, (I_1, \ldots, I_\ell))$ *and* $(t, \alpha', (I'_1, \ldots, I'_\ell))$ *with* $\alpha \neq \alpha'$ *or* $I_i \neq I'_i$ *for any* $i \in \{1, \ldots, \ell\}$, *then* $N(t, \alpha, (I_1, \ldots, I_\ell)) \cap N(t, \alpha', (I'_1, \ldots, I'_\ell)) = \emptyset$

We store all the $\eta = (t, \alpha, I_1, \ldots, I_\ell)$ together with the cardinality of $N(\eta)$ in a table $M_t$ consisting of $k = |A_t^v|$ columns encoding the variable assignment $\alpha$, $\ell = |A_t^a|$ columns containing $(I_1, \ldots, I_\ell)$ and one column storing the cardinality of $N(\eta)$.

We restrict our data structure $M_t$ to contain only rows where $I_1 \neq \emptyset, \ldots, I_\ell \neq \emptyset$. In the rules given below, if some row is created with $I_i = \emptyset$, this row is immediately removed from $M_t$ without explicitly mentioning this removal. This is justified by the fact that none of the assignments $\mu$ encoded by such rows can ever be extended to a variable assignment such that $\varphi$ is satisfied. To show that the size of each $M_t$ is only exponential in the treewidth (but not in the size of the query $\varphi$ and the instance $I$), we need the following lemmas:

LEMMA 3.2. *Let* $N(t, \alpha, (I_1, \ldots, I_\ell)) \neq \emptyset$ *and* $N(t, \alpha, (I'_1, \ldots, I'_\ell)) \neq \emptyset$. *Then for all* $i \in \{1, \ldots, \ell\}$, *either* $I_i = I'_i$ *or* $I_i \cap I'_i = \emptyset$

PROOF. Let $i$ be arbitrary and assume that $I_i \neq I'_i$. We have to show that then $I_i \cap I'_i = \emptyset$. Towards this goal, let $\mu \in N(t, \alpha, (I_1, \ldots, I_\ell))$ and $\mu' \in N(t, \alpha, (I'_1, \ldots, I'_\ell))$. As by assumption $I_i \neq I'_i$, $\mu$ and $\mu'$ differ on $Var(C_i)$. Therefore let $x \in Var(C_i)$ s.t. $\mu(x) \neq \mu'(x)$. By the definition of $\hat{\pi}$, it follows immediately that $I_i \cap I'_i = \emptyset$. □

LEMMA 3.3. *Given an instance* $I$, *CQ* $\varphi$, *and tree decomposition* $T$ *of width* $w - 1$ *of* $\varphi$, *the table* $M_t$ *stored for each* $t \in T$ *has at most* $O(|dom(I)|^w * |I|^w)$ *rows.*

PROOF. For every $t$, there can be at most $|dom(I)|^w$ different values of $\alpha$, since $A_t$ contains at most $w$ variables. Moreover, the number of different values of $(I_1, \ldots, I_\ell)$ is bounded by $|I|^w$ for the following reason: $\ell = |A_t^a| \leq w$. Moreover, for every $i$, we have: $I_i \neq \emptyset$, $I_i \cap I'_i = \emptyset$ for any two sets $I_i$, $I'_i$ with $I_i \neq I'_i$, and $I_i \subseteq I$. Hence, each $I_i$ may take at most $|I|$ different values. □

Next we describe how to fill $M_t$ for every $t \in T$ in a single bottom-up traversal of the tree decomposition.

*Leaf Node.* Let $t$ be a leaf node with the variables $x_1, \ldots, x_k$ and the atoms $C_1, \ldots, C_\ell$. Then for every variable assignment $\alpha$ on $x_1, \ldots, x_k$, $M_t$ contains a row with the variable assignment $\alpha$, $I_i = \hat{\pi}(\alpha(C_i))$, and the counter in each row is 1.

*Introduce Node (Atom).* Let $t$ be an introduce node for atom $C_{\ell+1}$, and $t'$ the child node of $t$. Then copy every row $\rho$ from $M_{t'}$ into $M_t$ and set $I_{\ell+1} = \hat{\pi}(\alpha(C_{\ell+1}))$. The counter of the row remains unchanged. The idea is that due to the connectedness condition, we know that the extensions $\mu$ of $\alpha$ to $V_t$ cannot affect any variable in $C_{\ell+1}$ except those mapped by $\alpha$.

*Introduce Node (Variable).* Let $t$ be an introduce node for variable $x_{k+1}$, and $t'$ the child node of $t$. Then for every row $\rho \in M_{t'}$, add $|dom(I)|$ rows to $M_t$, one for every possible extension of $\alpha_{t'}$ to $\alpha$, i.e. for every possible assignment for $x_{k+1}$. Every $I'_i$ is filtered according to $\alpha(x_{r+1})$, i.e. $I_i = \{A \in I'_i | C_i = R(s_1, \ldots, s_\gamma)$ and $A = R(s'_1, \ldots, s'_\gamma)$ and $s'_j = \alpha(x_{k+1})$ for all $j$ s.t. $s_j = x_{k+1}\}$. The value for the counter is copied.

*Forget Node (Atom).* Let $t$ be a forget node for atom $C_\ell$. Merge all rows from $M_{t'}$ in the child node $t'$ of $t$ that only differ by $I_\ell$. For every resulting row in $M_t$, the value of the counter is the sum over the counters of all merged rows.

*Forget Node (Variable).* Let $t$ be a forget node for variable $x_k$. Merge all rows from $M_{t'}$ in the child node $t'$ of $t$ that only differ by $x_k$. For every resulting row in $M_t$, the value of the counter is the sum over the counters of all merged rows.

*Join Node.* Let $t$ be a join node with two children $t'$ and $t''$. For each pair of rows $\rho' \in M_{t'}$ and $\rho'' \in M_{t''}$: If $\alpha' = \alpha''$, then add a row $\rho$ to $M_t$ with $\alpha = \alpha' = \alpha''$ and $I_i = I'_i \cap I''_i$ (for all $i \in \{1, \ldots, \ell\}$). The counters of $\rho'$ and $\rho''$ are multiplied to retrieve the counter for $\rho$. Finally all equal rows are merged and their counter values are summed up.

*Root Node.* Sum up the counters of all rows that do not contain empty sets. This gives the number of possible assignments to all variables in $\varphi$.

Analogously to the dynamic programming algorithm in [35] for counting the models of a propositional formula in CNF with bounded treewidth, we get the upper bound $O(|T| * (|dom(I)|^w * |I|^w)^2)$ for filling in $M_t$ for each node $t$ of the tree decomposition $T$. The reason for this is that each table $M_t$ (with $t \in T$) has at most $O(|dom(I)|^w * |I|^w)$ rows and computing $M_t$ from the table at the child node(s) can be done in time which is at most quadratic in the size of these tables. For instance, in case of a join node, we can compute $M_t$ by a nested loop over all rows in the tables $M_{t'}$ and $M_{t''}$ of the child nodes $t'$ and $t''$ of $t$.

So far, we have only considered the case where the CQ $\varphi$ contains only free variables. For the proof of Theorem 3.6 however, an algorithm is required for arbitrary CQs of the form $\varphi = \exists \vec{y} \psi(\vec{x}, \vec{y})$, to count the number of different variable assignments (solutions) on $\vec{x}$ only. It therefore remains to extend the above algorithm s.t. the resulting algorithm is still tractable w.r.t. the size of the query. Note that, if we were only interested in the decision problem (does there exists a solution?), we could still use the same data structure and

rules as in the previous case. However, to count the number of different assignments on $\vec{x}$, this algorithm does not give the correct results, as it cannot distinguish whether two truth assignments differ on $\vec{x}$ or on $\vec{y}$. To this end, we have to adapt the data structure stored at each node of the tree decomposition.

PROOF SKETCH OF THEOREM 3.6. As we are no longer interested in the number of different assignments on all variables, but only on the free ones, from now on we consider at each $t \in T$ mappings $\mu\colon V_t \to dom(I)$ as $\mu = \mu_x \cup \mu_y$ where $\mu_x\colon V_t \cap \vec{x} \to dom(I)$ and $\mu_y\colon V_t \cap \vec{y} \to dom(I)$. Therefore, we also slightly change the definition of a footprint of $\mu$ on some $t \in T$. We now consider a function $fpr(\mu_x, \mu_y) = (\alpha_x, \alpha_y, I_1, \ldots, I_\ell)$, where, as before, (1) $\alpha_x$ is the restriction of $\mu_x$ to the variables in $A_t^v$, (2) $\alpha_y$ is the restriction of $\mu_y$ to the variables in $A_t^v$, and (3) for every $C_i \in A_t^a$, $I_i = \hat{\pi}(\mu(C_i))$ (for $\mu = \mu_x \cup \mu_y$).

Again, different mappings $\mu$ may create the same footprint. However, from the information in the footprint, we cannot distinguish whether these mappings differ on the free or the existential variables. But only if they differ on the free variables, they correspond to different solutions. The problem here is that the same $\mu_x$ may (in combination with different $\mu_y$) create different footprints. Hence, Lemma 3.1 does no longer hold. We therefore group different assignments $\mu$ by $\mu_x$. Because of this we cannot use the footprints as identifiers for the sets of assignments managed at every node of the tree, but instead have to use *sets of footprints*. This results in the following datastructure to store the relevant information at each node $t$ of $T$:

Let $|\vec{x}| = k_1$, $|\vec{y}| = k_2$, and $M = |dom(I)|^{k_2}$. Now for a set of footprints $\theta_1, \ldots, \theta_m$ at some $t \in T$, where all $\theta_i$ share the same $\alpha_x$, we define the *combined footprint* $(\alpha_x, \mathcal{I})$, where $\mathcal{I} = \{(\alpha_y^1, I_1^1, \ldots, I_\ell^1), \ldots, (\alpha_y^m, I_1^m, \ldots, I_\ell^m)\}$. Given such a combined footprint $(\alpha_x, \mathcal{I})$ at node $t$, we define $N(t, \alpha_x, \mathcal{I})$ as the set of mappings $\mu_x\colon V_t \cap \vec{x} \to dom(I)$ such that

(a) for every $j \in \{1, \ldots, |\mathcal{I}|\}$, there exists a $\mu_y\colon V_t \cap \vec{y} \to dom(I)$ s.t. (1) $fpr(\mu_x, \mu_y) = (\alpha_x, \alpha_y^j, I_1^j, \ldots, I_\ell^j)$ and (2) for all $C_i \in At_t$ s.t. $Var(C_i) \subseteq V_t\colon \mu(C_i) \in I$ (where $\mu = \mu_x \cup \mu_y$)

(b) for all $\hat{\mu}_y$ s.t. $\mu_x \cup \hat{\mu}_y$ satisfies (a2) (i.e. for all $C_i \in At_t$ s.t. $Var(C_i) \subseteq V_t\colon \mu(C_i) \subseteq I$, for $\mu = \mu_x \cup \hat{\mu}_y$), there exists a $j$ s.t. $fpr(\mu_x, \hat{\mu}_y) = (\alpha_x, \alpha_y^j, I_1^j, \ldots, I_\ell^j)$.

By considering these sets $\mathcal{I}$ of combinations of $(\alpha_y^j, I_1^j, \ldots I_\ell^j)$ rather than a single footprint $(\alpha_y, I_1, \ldots I_\ell)$, the following property (corresponding to Lemma 3.1) holds:

LEMMA 3.4. *Given $(t, \alpha_x, \mathcal{I})$ and $(t, \alpha_x', \mathcal{I}')$ with $\alpha_x \neq \alpha_x'$ or $\mathcal{I} \neq \mathcal{I}'$ then $N\big(t, \alpha_x, \mathcal{I}\big) \cap N\big(t, \alpha_x', \mathcal{I}'\big) = \emptyset$.*

Hence, analogously to the $\exists$-free case treated above, we can count the number of solutions to a CQ $\varphi = \exists\vec{y}\psi(\vec{x}, \vec{y})$ by filling in tables $M_t$ at each node $t \in T$ consisting of columns encoding the variable assignment $\alpha_x$, the set $\mathcal{I}$, and the cardinality of $N\big(t, \alpha_x, \mathcal{I}\big)$. This can be done by rules analogously to the rules described above for the various types of nodes of a nice tree decomposition. Analogously to Lemma 3.3, the number of rows in each table $M_t$ is bounded by $O(2^{|dom(I)|^w \cdot |I|^w})$ since we now have to deal with *sets of* combinations $(\alpha_y^j, I_1^j, \ldots I_\ell^j)$. However, for the query complexity, $|I|$ is considered as constant and, therefore, the size of each table $M_t$ is polynomially bounded w.r.t. the size of $\varphi$. Thus, also the computation of each $M_t$ is feasible in polynomial time w.r.t. the size of $\varphi$. □

# 4. SECOND-ORDER TGDS

In [13], *second-order tuple generating dependencies (SO tgds)* were introduced since FO tgds are not powerful enough to express the composition of schema mappings defined by FO s-t tgds. In contrast, SO tgds are closed under composition. Moreover, any set of FO s-t tgds can be transformed into an equivalent set of SO tgds.

An SO tgd over source and target schema $\langle \mathbf{S}, \mathbf{T} \rangle$ is a formula of the form

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

where (1) each member of $\vec{f}$ is a function symbol, (2) each $\varphi_i$ is a conjunction of atoms $S(y_1, \ldots, y_k)$ ($S \in \mathbf{S}$, $y_j \in \vec{x}_i$) and equalities of the form $t = t'$ ($t, t'$ are terms based on $\vec{x}_i$ and $\vec{f}$), (3) each $\psi_i$ is a conjunction of atoms $T(t_1, \ldots, t_l)$ ($T \in \mathbf{T}$, $t_1, \ldots, t_l$ are terms based on $\vec{x}_i$ and $\vec{f}$), and (4) every $y \in \vec{x}_i$ must appear in some atom in $\varphi_i$.

The semantics of SO tgds over instances $\langle I, J \rangle$ is defined over structures $\langle U; I, J \rangle$, where the *universe* $U$ includes the active domain and is "sufficiently large" (in [13], it was discovered that it is not necessary to consider arbitrarily large universes, but it suffices to consider a universe of polynomial size). An SO tgd $\sigma = \exists \vec{f} \sigma'$ (where $\sigma'$ is a first-order formula) is satisfied by $\langle U; I, J \rangle$, if there exists a set of functions $\mathbf{f}^0$ ($\forall f_i^0 \in \mathbf{f}^0\colon f_i^0\colon U^{k_i} \to U$) s.t. $\langle U; I, J \rangle \models \sigma'[\vec{f} \to \mathbf{f}^0]$.

EXAMPLE 4.1 ([13]). *Consider the schemas $\mathbf{S}_1 = \{T(n, c)\}$, $\mathbf{S}_2 = \{T'(n, c), S(n, s)\}$, and $\mathbf{S}_3 = \{E(s, c)\}$, where $T(n, c)$ and $T'(n, c)$ mean that a student with name $n$* takes *course $c$, $S(n, s)$ associates to each* student *name $n$ a student id $s$, and $E(s, c)$ means that a student with id $s$ is* enrolled *on course $c$. Let two mappings from $\mathbf{S}_1$ to $\mathbf{S}_2$ and from $\mathbf{S}_2$ to $\mathbf{S}_3$ be defined by the following sets of FO s-t tgds:*

$$\Sigma_{12} = \{(\forall n, c)\big(T(n, c) \to T'(n, c)\big), \\ (\forall n, c)\big(T(n, c) \to (\exists s) S(n, s)\big)\} \text{ and}$$
$$\Sigma_{23} = \{(\forall n, s, c)\big(S(n, s) \wedge T'(n, c) \to E(s, c)\big)\}.$$

*Fagin et al. showed that the composition of these two mappings cannot be expressed by FO tgds. To see this, consider an instance of $\mathbf{S}_1$ with atoms $T(n, c_1), \ldots, T(n, c_m)$ (i.e., the same student $n$ takes several courses). Then the atoms $E(s, c_1), \ldots, E(s, c_m)$ in the corresponding instance of $\mathbf{S}_3$ must have the same student id $s$. FO tgds are too weak to enforce the equality of the first component in all atoms $E(s, c_i)$. In particular, the FO tgd $(\forall n, c)\big(T(n, c) \to (\exists s) E'(s, c)\big)$ fails to enforce this equality. In contrast, the SO tgd in $\Sigma_{13} = \{(\exists f)(\forall n, c)\big(T(n, c) \to E(f(n), c)\big)\}$ does the trick.*

In this section, we want to pinpoint the query complexity and combined complexity of model checking of SO tgds. We shall thus prove the NEXPTIME-completeness of this problem. To establish this result, we introduce the notion of SO-QBFs, i.e., quantified Boolean formulae (QBFs) extended by existentially quantified second-order function variables. We shall first show the NEXPTIME-completeness of SO-QBFs and then prove the hardness of model checking of SO tgds by a reduction from SO-QBFs.

## 4.1 Second-Order QBFs

QBFs are formulae of the form $F = \forall \vec{x}_1 \exists \vec{x}_2 \forall \vec{x}_3 \exists \vec{x}_4 \ldots Q_n \vec{x}_n$ $\varphi(\vec{x}_1, \ldots, \vec{x}_n)$, s.t. $\varphi(\vec{x}_1, \ldots, \vec{x}_n)$ is a propositional formula over the Boolean algebra $\mathbb{B} = (\{0, 1\}; \wedge, \vee, \neg, \to, \leftrightarrow)$. W.l.o.g., we assume that the first quantifier is universal; the last quantifier $Q_n$ is either existential or universal. The Boolean variables are interpreted over $\{0, 1\}$; the quantifiers and connectives have the usual meaning. Model checking of QBFs (i.e., checking if a given QBF $\psi$ evaluates to 1) is the classical PSPACE-complete problem [33].

We extend QBFs to SO-QBFs by allowing existentially quantified function symbols at the beginning of the quantifier prefix and

functional terms as atoms, i.e., SO-QBFs are of the form $F = \exists\vec{f}\forall\vec{x}_1\exists\vec{x}_2\forall\vec{x}_3\ldots Q_n\vec{x}_n\varphi(\vec{f},\vec{x}_1,\ldots,\vec{x}_n)$, where $\vec{f}$ is a set of function symbols. The atoms in $\varphi(\vec{f},\vec{x}_1,\ldots,\vec{x}_n)$ may be Boolean variables, Boolean constants 0,1, or *functional terms* of the form $f_i(t_1,\ldots,t_{k_i})$, where $f_i \in \vec{f}$, $k_i$ denotes the arity of $f_i$ and $t_1,\ldots,t_{k_i}$ are Boolean variables or Boolean constants 0,1. The Boolean variables, quantifiers and connectives are interpreted as usal. Each $k_i$-ary function symbol $f_i \in \vec{f}$ is interpreted by a function $f_i^0\colon \{0,1\}^{k_i} \to \{0,1\}$. Below we show that model checking of SO-QBFs (i.e., checking if a given SO-QBF $\psi$ evaluates to 1) is significantly harder than model checking of QBFs.

THEOREM 4.1. *Model checking of SO-QBFs is NEXPTIME-complete. It remains NEXPTIME-complete even for formulae $F = \exists\vec{f}\forall\vec{x}\varphi(\vec{f},\vec{x})$ where $\varphi(\vec{f},\vec{x})$ is in 3-CNF and contains no constants 0,1.*

PROOF SKETCH. The *membership* is established via the following non-deterministic algorithm: We first guess the functions, that is, an exponentially big function table for each function symbol. Then the SO-QBF essentially reduces to a QBF, where the value of each functional term in any truth assignment is obtained by a simple lookup into the corresponding function table. Hence, the SO-QBF evaluation requires only polynomial space w.r.t. the size of the input formula (in addition to the function tables) and is thus feasible in exponential time.

The *hardness* is shown by encoding the computation of a NEXPTIME-Turing machine into an SO-QBF similarly to the proof of the Cook-Levin Theorem [33]. The exponentially higher complexity than in case of SAT is due to the existentially quantified function symbols, which allow us to encode a successor relation. We can then binary encode exponentially many time instants and tape positions by vectors $X$ and $Y$ of Boolean variables. Moreover, we can use functional terms to express the configuration of the Turing machine, and to enforce only valid transitions between two configurations. Below, we sketch the reduction.

Let $M$ denote a non-deterministic Turing machine (TM) which decides some NEXPTIME-problem $L$. W.l.o.g., we may assume that, on any input $I$, the TM $M$ halts after exactly $2^m - 1$ steps, with $m = n^k$ and $n = |I|$. We define the following SO-QBF to simulate the computation of $M$ on input $I$:

**Functions.** In our construction, we will use the following functions, with the corresponding intended meaning (upper case letters denote vectors of Boolean variables, while lower case letters stand for single Boolean variables. Superscripts denote the arity of a function or the size of the vector, respectively. If omitted, vectors have size $m$, and the arity of the function is clear from its arguments.): For every $i \in \{1,\ldots,m\}$, $\mathit{last}^i(X^i)$, $\mathit{succ}^i(X^i,Y^i)$, where $\mathit{first}^i(X^i)$ and $\mathit{last}^i(X^i)$ denote that $X^i$ is the first resp. last $i$-ary bit vector (i.e.: $(0,\ldots,0)$ resp. $(1,\ldots,1)$, and $\mathit{succ}^i(X^i,Y^i)$ expresses that the $i$-ary bit-vector $Y^i$ is the successor of $X^i$. We encode the lexicographical ordering on $\{0,1\}^m$ by the "less or equal" relation $\mathit{leq\_than}(X,Y)$ and the "less" relation $\mathit{le\_than}(X,Y)$. The terms $\mathit{state}_s(X)$, $\mathit{cursor}(X,Y)$, and $\mathit{symbol}_\sigma(X,Y)$ express that, at time instant $X$, the machine $M$ is in state $s$, the cursor is at position $Y$, and the tape cell $Y$ contains the symbol $\sigma$.

**Initial configuration.** Suppose that the Turing machine is run on the input string $w = \sigma_1'\ldots\sigma_n'$. Moreover, let $b_0,\ldots,b_n$ denote the binary representations of the numbers $0,\ldots,n$. Then the initial configuration of the Turing machine is encoded by the following subformulae: $\forall X(\mathit{first}(X) \to \mathit{cursor}(X,X))$ (the cursor is located over the first tape cell), $\forall X(\mathit{first}(X) \to \mathit{state}_{s_0}(X))$ (the machine is in the initial state), $\forall X(\mathit{first}(X) \to \mathit{symbol}_\triangleright(X,b_0) \wedge \mathit{symbol}_{\sigma_1'}(X,b_1)) \wedge \cdots \wedge \mathit{symbol}_{\sigma_n'}(X,b_n))$ (the first $n+1$ cells contain the left tape delimiter $\triangleright$ followed by the input string $w$), and $\forall X\forall Y(\mathit{first}(X) \wedge \mathit{le\_than}(b_n,Y) \to \mathit{symbol}_\sqcup(X,Y))$ (the remaining tape cells are empty).

**Acceptance.** W.l.o.g. we assume that $M$ halts after $2^m - 1$ steps in an accepting state. Let $s_q$ denote the accepting state. Then acceptance is encoded by $\varphi_{acc} = \forall X(\mathit{last}(X) \to \mathit{state}_{s_q}(X))$.

**Valid configurations of $M$.** To ensure that the encoding represents only valid configurations of $M$, we define subformulae which encode that, at every time instant, the TM is in exactly one state, the cursor is at exactly one position and every tape cell contains exactly one symbol. For example, the latter is expressed by one subformula $\forall X\forall Y(\mathit{symbol}_{\sigma_j}(X,Y) \leftrightarrow (\neg\mathit{symbol}_{\sigma_0}(X,Y) \wedge \cdots \wedge \neg\mathit{symbol}_{\sigma_{j-1}}(X,Y) \wedge \neg\mathit{symbol}_{\sigma_{j+1}}(X,Y) \wedge \cdots \wedge \neg\mathit{symbol}_{\sigma_p}(X,Y)))$ for every symbol $\sigma_j$ from the alphabet.

**Inertia rules.** To ensure that changes on the tape occur only at the cursor position, we add for every symbol $\sigma$ from the alphabet the subformula $\forall X\forall X'\forall Y\forall Y'(\mathit{cursor}(X,Y) \wedge \mathit{symbol}_\sigma(X,Y') \wedge \neg(Y = Y') \wedge \mathit{succ}(X,X') \to \mathit{symbol}_\sigma(X',Y'))$.

**Transition rules.** For each pair $(s,\sigma)$ of a state $s$ and a symbol $\sigma$ let $(s,\sigma,s_1',\sigma_1',d_1),\ldots,(s,\sigma,s_k',\sigma_k',d_k)$ denote all possible transitions of the Turing machine. We encode these transitions by the formula $\forall X\forall X'\forall Y\forall Y'\forall Y''$ $((\mathit{state}_s(X) \wedge \mathit{cursor}(X,Y) \wedge \mathit{symbol}_\sigma(X,Y) \wedge \mathit{succ}(X,X') \wedge (\mathit{last}(Y) \vee \mathit{succ}(Y,Y')) \wedge (\mathit{first}(Y) \vee \mathit{succ}(Y'',Y))) \to$
$((\mathit{symbol}_{\sigma_1'}(X',Y) \wedge \mathit{state}_{s_1'}(X') \wedge \mathit{cursor}(X',Y_1^*)) \vee \cdots \vee (\mathit{symbol}_{\sigma_k'}(X',Y) \wedge \mathit{state}_{s_k'}(X') \wedge \mathit{cursor}(X',Y_k^*))))$,
where $Y_i^* = Y$ if $d_i = 0$, $Y^* = Y'$ if $d_i = +1$, and $Y^* = Y''$ if $d_i = -1$.

**Helper functions.** In addition, we need several subformulae to ensure that the functions $\mathit{first}$, $\mathit{last}$, $\mathit{succ}$, $\mathit{le\_than}$, and $\mathit{leq\_than}$ have their desired meanings.

**Summary.** The SQ-QBF encoding the Turing machine consists of the conjunction of all these subformulae, and the SO quantification goes over all the above mentioned function symbols. The transformation into 3-CNF and the elimination of Boolean constants is routine. The desired form $F = \exists\vec{f}\forall\vec{x}\varphi(\vec{f},\vec{x})$ is obtained by a kind of Skolemization, i.e., suppose that the quantifier prefix starts with $\exists\vec{f}\forall\vec{x}_1\exists(y_1,\ldots,y_m)\forall\vec{x}_2$. Then the existentially quantified Boolean variables $y_1,\ldots,y_m$ are replaced by functional terms $g_1(\vec{x}_1),\ldots,g_m(\vec{x}_1)$ and the quantifier prefix is transformed into $\exists\vec{f}\exists(g_1,\ldots,g_m)\forall\vec{x}_1\forall\vec{x}_2$. □

## 4.2 Complexity of SO Tgds

We now use SO-QBFs to establish the complexity of SO tgds.

THEOREM 4.2. *SO tgd model checking is NEXPTIME-complete (both, query and combined complexity). It remains NEXPTIME-complete even if $\Sigma$ contains a single SO tgd consisting of a single implication only.*

PROOF SKETCH. For the *membership*, we proceed analogously to Theorem 4.1: We first guess the functions (i.e., an exponentially big function table for each function symbol). The size of these function tables is indeed single-exponential due to the aforementioned result from [13] that it suffices to consider a polynomially big universe. Then the SO tgd essentially reduces to a first-order formula, which can be evaluated in polynomial space and hence in exponential time (w.r.t. the size of the input formula).

The *hardness*-proof is by reduction from SO-QBFs. We define $\langle U; I, J\rangle$ as $U = \{0,1\}$, $I = \{P(0,1),\ P(1,0)\}$, and $J =$

$\{Q(0,1), Q(1,0)\} \cup \{C(\alpha,\beta,\gamma) \mid \alpha,\beta,\gamma \in \{0,1\}\} \backslash \{C(0,0,0)\}$. Let $F = \exists(f_1,\dots,f_k)\forall(x_1,\dots,x_\ell)\varphi(\vec{f},\vec{x})$ be an arbitrary SO-QBF with $\varphi(\vec{f},\vec{x}) = C_1 \wedge \dots \wedge C_n$ and $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$, where each $l_{i,j}$ is a (negated or unnegated) variable or functional term. We construct the SO tgd $\tau = \exists\vec{f}\exists\vec{f}'(\forall\vec{x}\forall\vec{x}'(\varphi \rightarrow \psi))$ with
$\vec{f}' = \{f' \mid f \in \vec{f}\}$, $\vec{x}' = \{x' \mid x \in \vec{x}\}$,
$\varphi = \bigwedge_{i=1}^{\ell} P(x_i, x_i')$, and
$\psi = \psi_1 \wedge \psi_2$ with
$\psi_1 = \big(\bigwedge_{i=1}^{n} C(l_{i,1}^*, l_{i,2}^*, l_{i,3}^*)\big)$ and
$\psi_2 = \big(\bigwedge_{f_i(\vec{t})\in\psi_1 \vee f_i'(\vec{t})\in\psi_1} Q(f_i(\vec{t}), f_i'(\vec{t}))\big)$, where

$$l_{i,j}^* = \begin{cases} x_\gamma & \text{if } l_{i,j} = x_\gamma \\ x_\gamma' & \text{if } l_{i,j} = \neg x_\gamma \\ f_\gamma(x_{\gamma,1},\dots,x_{\gamma,\alpha}) & \text{if } l_{i,j} = f_\gamma(x_{\gamma,1},\dots,x_{\gamma,\alpha}) \\ f_\gamma'(x_{\gamma,1},\dots,x_{\gamma,\alpha}) & \text{if } l_{i,j} = \neg f_\gamma(x_{\gamma,1},\dots,x_{\gamma,\alpha}) \end{cases}$$

Clearly, this reduction is in LOGSPACE. □

As in the FO case, the update variants of the problem contain the full hardness of the basic problem of model checking.

THEOREM 4.3. *SO tgd delete-update model checking and SO tgd insert-update model checking are NEXPTIME-complete (both, query and combined complexity). They remain NEXPTIME-complete even if $\Sigma$ contains a single SO tgd with a single implication and the update $U'$ contains a single tuple only.*

PROOF IDEA. *Membership* follows directly from Theorem 4.2. The *hardness* of both, SO tgd delete-update model checking and insert-update model checking is shown by the same modifications of the proof of Theorem 4.2 as in the FO case. □

To achieve a lower complexity, we define the fragment SO$^{ord}$ of SO tgds as follows: W.l.o.g., SO tgds contain no nesting of functional terms, i.e., any term of the form $f(\dots, g(\vec{t}), \dots)$ can be replaced by a term of the form $f(\dots, z, \dots)$ and an additional equality $z = g(\vec{t})$ for a fresh variable $z$, see [13]. Hence, terms in SO tgds are either constants or (first-order) variables or functional terms of the form $f(t_1,\dots,t_k)$, where $f$ is a second-order variable and the $t_i$'s are either constants or (first-order) variables. An SO tgd $\tau$ is called an SO$^{ord}$ tgd if it is rectified (i.e., every variable is bound by only one quantifier) and there exists an ordering $\vec{X} = (x_{i_1},\dots,x_{i_n})$ of the first-order variables in $\tau$, s.t. (1) the variables occurring in each functional term in $\tau$ form a prefix of $X$, i.e., the set of variables occurring in a functional term is of the form $\{x_{i_1},\dots,x_{i_\alpha}\}$ with $\alpha \leq n$ and, moreover, (2) for every pair of functional terms $f(s_1,\dots,s_k)$ and $f(t_1,\dots,t_k)$ in $\tau$ with identical function symbol, either these two terms differ on some constant position (i.e., $s_i \neq t_i$ for some $i$, s.t. $s_i, t_i$ are constants) or these two terms coincide on all variable positions (i.e., $s_i = t_i$ for all $i$, s.t. $s_i, t_i$ are variables). We show that SO$^{ord}$ tgds can be efficiently transformed into FO formulae. Thus, the complexity of model checking reduces to PSPACE.

THEOREM 4.4. *SO$^{ord}$ tgds can be transformed into first-order formulae by a PTIME-transformation.*

PROOF. Let $\vec{X} = (x_1,\dots,x_n)$ denote the ordering of the first-order variables in $\tau$, s.t. the variables occurring in each functional term in $\tau$ form a prefix of $X$. W.l.o.g., we may arrange the universal quantifiers in this order. Then the transformation into an FO formula can be achieved by applying "de-Skolemization", which is a common technique in second-order quantifier elimination [8, 17], i.e., rather than replacing an existentially quantified variable $y$ in the scope of universally quantified variables $\{x_1,\dots,x_\alpha\}$ by a functional term $g(x_1,\dots,x_\alpha)$, we replace a functional term $f(x_{i_1},\dots,x_{i_\alpha})$ (where $\{x_{i_1},\dots,x_{i_\alpha}\} = \{x_1,\dots,x_\alpha\}$ holds) by an existentially quantified FO variable $y$ in the scope of $\forall(x_1, \dots, x_\alpha)$. □

Note that the SO tgd $\tau = (\exists f)(\forall n, c)\big(T(n,c) \rightarrow E(f(n),c)\big)$ in Example 4.1 is an SO$^{ord}$ tgd and, thus, Theorem 4.4 is applicable to $\tau$. Indeed, as already mentioned in [13], $\tau$ is equivalent to the FO formula $\tau' = \forall n \exists y \forall c\big(T(n,c) \rightarrow E(y,c)\big)$.

THEOREM 4.5. *The SO$^{ord}$ tgd model checking problem is* PSPACE*-complete (both, query and combined complexity). The problem remains* PSPACE*-complete even if $\Sigma$ contains a single tgd with a single implication only.*

PROOF. The *membership* is an immediate consequence of Theorem 4.4. The *hardness* can be shown by defining SO$^{ord}$-QBFs as a fragment of QBFs (in the same way as SO$^{ord}$ tgds) and by establishing the following chain of reductions: From QBFs to SO$^{ord}$-QBFs and from SO$^{ord}$-QBFs to SO$^{ord}$ tgds. The latter reduction is precisely the one from Theorem 4.2. The first reduction is done using Skolemization: Let $y_i$ be an existentially quantified variable and suppose that $y_i$ is in the scope of the universally quantified variables $\vec{x}$. Then we introduce a new function symbol $g_i$ and replace every occurrence of $y_i$ by the functional term $g_i(\vec{x})$. Obviously, the resulting SO-QBF falls into the set of SO$^{ord}$-QBFs. □

In the above proof, the reduction from SO$^{ord}$-QBFs to SO tgds is the same as in the proof of Theorem 4.2. Hence, the PSPACE-hardness of the update variants of the problem follows immediately by applying the same ideas as in the proof of Theorem 4.3.

THEOREM 4.6. *The problems SO$^{ord}$ tgd insert-update model checking and SO$^{ord}$ tgd delete-update model checking are PSPACE-complete (both, query and combined complexity). The problems remain PSPACE-complete even if $\Sigma$ contains a single SO tgd with a single implication and the update $U'$ contains a single tuple only.*

Finally, we show that the fragment of SO$^{ord}$ tgds can be efficiently identified.

THEOREM 4.7. *It can be checked in PTIME if an arbitrary SO tgd is an SO$^{ord}$ tgd.*

PROOF SKETCH. Let $t_1,\dots,t_m$ denote the functional terms occurring in an arbitrary SO tgd $\tau$. For $i \in \{1,\dots,m\}$, let $X_i$ denote the set of variables occurring in $t_i$. Now let $X_{j_1},\dots,X_{j_m}$ be an ordering of these variable sets $X_i$ by ascending cardinality (ties are broken by random). Then we first check if $X_{j_1} \subseteq \dots \subseteq X_{j_m}$ holds. If this is the case, then we can define an ordering $\vec{X}$ on the variables in $\tau$ by first arranging the variables of $X_{j_1}$ in arbitrary order, then the variables in $X_{j_2} \setminus X_{j_1}$, …, then $X_{j_m} \setminus X_{j_{m-1}}$. It remains to check that the conditions (1) and (2) of SO$^{ord}$ are thus fulfilled. □

# 5. NESTED TGDS

Nested tgds were introduced in [16] to allow for more structure preserving transformations of data from a source into a target instance. As they were defined over a hierarchical schema, before giving the definition of nested tgds, we first adopt the nested relational model presented in [34] to our needs.

We extend the notion of relation schemas $R(A_1,\dots,A_n)$ by allowing two different types of attributes $A_i$, namely atomic and relation attributes. Thereby the latter ones are again relation schemas.

An instance still consists of a set of atoms, with each atom now containing at each position either an atomic value or a set of atoms (referred to as subrelation) depending on the type of the corresponding attribute. By slight abuse of notation we simply write $R$ to denote the relation schema $R(A_1, \ldots, A_n)$. We define the *nesting depth* $\delta(R)$ of relation schema $R$ as $\delta(R) = 1$ if all attributes of $R$ are atomic and $\delta(R) = 1 + \max(\{\delta(A_i) \mid A_i \text{ is a relation attribute of } R\})$ otherwise.

Now let $I$ and $J$ be two instances of relation schema $R$. We recursively define a *hierarchical homomorphism* $I \to J$ as a mapping $h\colon I \to J$ with the following properties: (1) Let $\delta(R) = 1$. Then $h$ is a hierarchical homomorphism iff $h$ is a homomorphism $I \to J$. (2) Now let $\delta(R) > 1$. W.l.o.g. we assume $R$ to contain $k + l$ attributes, where the first $k \geq 0$ attributes of $R$ have atomic type and the last $l \geq 1$ attributes are relation attributes. Then $h$ is a hierarchical homomorphism iff for every atom $t = R(t_1, \ldots, t_{k+l}) \in I$ there exists an atom $t' = R(t'_1, \ldots, t'_{k+l}) \in J$ s.t. $h(t_1) = t'_1, \ldots, h(t_k) = t'_k$ and, for every $j \in \{1, \ldots, l\}$, $h\colon t_{k+j} \to t'_{k+j}$ is a hierarchical homomorphism.

A *nested tgd* is a logical formula $\exists \vec{F}(\chi_{1,1})$ where $\chi_{1,1}$ is defined over the hierarchical relational model inductively as

$\chi_{1,1} = \forall \vec{x}_1(\varphi_1(\vec{x}_1) \to \exists \vec{y}_1(\psi_1(\vec{x}_1, \vec{y}_1) \wedge \chi_{2,1} \wedge \cdots \wedge \chi_{2,m_2}))$,

s.t. for every $i$ and $j$, $\chi_{i,j}$ is defined as

$\chi_{i,j} = \forall \vec{x}_i(\varphi_i(\vec{x}_1, \ldots, \vec{x}_i) \to$
$\exists \vec{y}_i(\psi_i(\vec{x}_1, \ldots, \vec{x}_i, \vec{y}_1, \ldots, \vec{y}_i) \wedge \chi_{i+1,1} \wedge \cdots \wedge \chi_{i+1,m_{i+1}}))$.

Thereby, every $\varphi_i$ is a conjunction of source atoms, while $\psi_i$ is a conjunction of target atoms; variables in $\vec{x}_i$ and $\vec{y}_i$ can be of atomic type (atomic variables) or relational type (relation variables), and all variables from $\vec{x}_i$ (resp. $\vec{y}_i$) must occur in $\varphi_i$ (resp. $\psi_i$). A source atom is an atom whose leading symbol is either a relation symbol from the source schema or a relation variable that was bound in some source atom. Thereby a relation variable is bound if it occurs in the arguments of an atom with either a predicate symbol or a bound relation variable as leading symbol. Target atoms are defined analogously. Hence relation variables cannot only occur in the arguments of an atom, but can also be used as leading symbol of an atom. Note that it follows from the above definition, that relation variables can only occur in nested tgds over schemas with nesting depth greater than 1. $\vec{F}$ is a set of function variables which are used to map combinations of atomic values to relations. Nested tgds allow terms built with these function variables (using only universally quantified atomic variables as arguments) to be used in any place where an existential relation variable may stand. Therefore nested tgds over schemas of nesting depth 1 cannot contain any second-order term. We refer to $\chi_{i,j}$ as a *submapping* at level $i$. By slight abuse of notation, we reuse the same symbols ($\vec{x}_i$, $\vec{y}_i$, $\chi_{i,j}$) in different submappings at the same level, although they can take different values in every such submapping. Similar to SO tgds, a nested tgd $\exists \vec{F}(\chi)$ is satisfied if for some set $\vec{F}^0$ of concrete functions $\langle I, J\rangle \models \chi[\vec{F} \to \vec{F}^0]$ holds (under the above definition of a hierarchical homomorphism).

The following example demonstrates the previous definitions. Thereby we first use the imperative syntax from [16], and then rephrase the schemas and tgds in a more logic-style notation.

EXAMPLE 5.1 ([16]). *Consider two relation schemas Proj and Dept, where Proj is a source relation schema and Dept a target relation schema. Suppose that they are defined as follows:*

*Proj*: *set of* $[dN, pN, Emps$: *set of* $[eN]]$ *and*
*Dept*: *set of* $[dN, Emps_t$: *setof* $[eN, Proj_t$: *set of* $[pN]]]$,

*where d stands for "department", e for "employee", p for "project", and N for "name". In a logic-style notation, the above relation schemas can be written as* $Proj(dN, pN, Emps(eN))$ *and* $Dept(dN, Emps_t(eN, Proj_t(pN)))$.

*Intuitively, in the source schema, employees are grouped by projects and departments. In the target schema, the information is first grouped by departments and then by employees. Consider the following nested tgd* $\tau$ *over the above schemas:*

$\underline{for}\ p\ \underline{in}\ Proj \Rightarrow \underline{exists}\ d'\ \underline{in}\ Dept$
$\underline{where}\ d'.dN = p.dN \wedge d'.Emps_t = E[p.dN] \wedge$
$(\ \underline{for}\ e\ \underline{in}\ p.Emps \Rightarrow \underline{exists}\ e'\ \underline{in}\ d'.Emps_t,\ p'\ \underline{in}\ e'.Proj_t$
$\underline{where}\ p'.pN = p.pN \wedge e'.eN = e.eN \wedge$
$e'.Proj_t = P[p.dN, e.eN]),$

*where* $E$ *and* $P$ *are functions returning sets, and* $E[x]$ *and* $P[x]$ *denote the function value for argument* $x$. *In the logic-style notation, the same mapping can be written as*

$\exists E, P\ ((\forall d, p, E_s)(Proj(d, p, E_s) \to (Dept(d, E[d]) \wedge$
$((\forall e)(E_s(e) \to (E[d](e, P[d, e]) \wedge P[d, e](p)))))))$

*Now let the instances* $I$ *and* $J$ *be defined as* $I = \{Proj('d1', 'p1', \{Emps('e1')\}), Proj('d1', 'p2', \{Emps('e1')\})\}$ *and* $J = \{Dept('d1', \{Emps_t('e1', \{Proj_t('p1')\})\}), Dept('d1', \{Emps_t('e1', \{Proj_t('p2')\})\})\}$. *Then* $\langle I, J\rangle \models \tau$ *clearly holds.*

*Note that the use of the functions* $E$ *and* $P$ *requires both projects to be contained in the same subrelation in the target instance, while e.g. for* $J' = \{Dept('d1', \{Emps_t('e1', \{Proj_t('p1')\})\}), Dept('d1', \{Emps_t('e1', \{Proj_t('p2')\})\})\}$, *we have* $\langle I, J'\rangle \not\models \tau$. *In other words, the set functions allow us to express grouping.*

As can be seen from the example, the functions allowed in nested tgds can be used to restructure the source data by expressing grouping conditions. Moreover, every existential relation variable can be replaced by a default Skolem function, that is by introducing a new function symbol and by replacing every occurrence of the variable by a functional term, built from the new function symbol and taking as arguments all universally quantified variables the replaced relation variable was in the scope of. If the resulting tgd contains only such default Skolem functions, this process and the resulting tgd are called *default Skolemization* in [16].

Nested tgds have also been considered in [36], but there no hierarchical schema was assumed, but only relations of depth 1. When applying the above definition (that captures the definition in [16]) to schemas of nesting depth 1 only, we arrive at the definition presented in [36], except that in the latter every $\psi_i$ is allowed to be missing (i.e. to consist of no atoms). However, this is a purely technical issue and has no impact on the complexity as we will see below. In contrast, the restriction of schemas to nesting depth 1 (in connection with disallowing the use of second-order functions) clearly does make a big difference. We show below that this restriction significantly decreases the complexity of model checking.

Towards the complexity of nested tgd model checking, we first observe that applying standard techniques for encoding object relational databases as purely relational ones allows us to replace set functions by atomic functions and to avoid nested relations. To this end we recursively introduce for every relation attribute with schema $R(A_1, \ldots, A_n)$ a new top level relation schema $R(id, A_1, \ldots, A_n)$. For every subrelation we create a unique identifier, replace the subrelation by this identifier and move the content of the subrelation into the corresponding new top level relation, where $R.id$ contains the identifier of the original subrelation. To transform nested tgds, we additionally replace all set functions by atomic functions determining the unique identifier values, and relation variables are replaced by atomic variables for the corresponding identifiers. We denote the result of this operation as *flattened*$(X)$, where $X$ is a hierarchical schema, an instance, or a (nested) tgd.

EXAMPLE 5.2. *Recall the schemas, tgd, and instances from Example 5.1. The flattened correspondence of the target schema is* $\{Dept(dN, eid),\ Emps_t(eid, eN, pid),\ Proj_t(pid, pN)\}$.

*The flattened tgd is*

$$\exists f_e, f_p\ ((\forall d, p, eid)(Proj(d, p, eid) \rightarrow (Dept(d, f_e(d)) \wedge ((\forall e)(Emps(eid, e) \rightarrow (Emps_t(f_e(d), e, f_p(d, e)) \wedge Proj_t(f_p(d, e), p)))))).$$

*The flattened target instance is* $\{Dept('d1', eid1),\ Emps_t(eid1, 'e1', pid1),\ Proj_t(pid1, 'p1'),\ Proj_t(pid1, 'p2')\}$.

It can be verified that $\langle I, J\rangle \models \tau$ iff $\langle \mathit{flattened}(I), \mathit{flattened}(J)\rangle \models \mathit{flattened}(\tau)$, and, obviously, $\mathit{flattened}(X)$ can be computed efficiently. The flattening will be useful to prove the membership part of the following complexity result:

THEOREM 5.1. *The nested tgd model checking problem over a hierarchical schema is NEXPTIME-complete (both, query and combined complexity). The problem remains NEXPTIME-complete even if* $\Sigma$ *contains only a single nested tgd.*

PROOF IDEA. It was already noticed in [16] that over purely relational schemas, every nested tgd can be transformed into an equivalent SO tgd. Hence, we can transform nested tgds by first applying the flattening and then transforming the flattened tgds into a corresponding SO tgd. Hence, the NEXPTIME-membership of nested tgds follows immediately from the NEXPTIME-membership of SO tgds (see Theorem 4.2).

The *hardness* is shown by reduction from SO-QBFs. The idea is the same as in the reduction from SO-QBFs to SO tgds, except that we have to express function variables in the SO-QBF by functions over subrelations, as nested tgds do not support functions over atomic values. □

Analogously to FO tgds and SO tgds, also for nested tgds, the update variants have the full complexity of model checking:

THEOREM 5.2. *Nested tgd insert-update model checking and nested tgd delete-update model checking are NEXPTIME-complete. They remain NEXPTIME-complete even if* $\Sigma$ *contains a single nested tgd and* $U$ *contains a single atom only.*

PROOF IDEA. *Membership* follows directly from Theorem 4.2, while the *hardness* proof of the two update variants is based on ideas similar to the FO and SO case. □

Analogously to SO tgds, we want to identify a fragment of nested tgds, based on the structure of the functional terms, for which model checking has a lower complexity. W.l.o.g. we assume a nested tgd $\tau$ to be flattened. However, unlike for SO tgds, we cannot define the fragment of nested$^{ord}$ tgds immediately on $\mathit{flattened}(\tau)$, as it may still contain existential variables, because of which we are not allowed to arbitrarily change the order of the universally quantified variables. Therefore we first have to replace every existentially quantified variable by a new Skolem function. Then, analogously to SO$^{ord}$, we define the nested$^{ord}$ tgds as those nested tgds where there exists an ordering $\vec{X}$ on the universally quantified variables of $\tau$, such that after flattening and Skolemization, (1) the variables occurring in each functional term form a prefix of $\vec{X}$ (2) for every pair of functional terms $f(\vec{s})$ and $f(\vec{t})$ with identical function symbol, either these two terms differ on some constant position or they coincide on all variable positions.

THEOREM 5.3. *Nested$^{ord}$ tgds can be transformed into first-order formulae in PTIME.*

PROOF IDEA. The general idea of this transformation is, given some nested tgd $\tau$, first to obtain $\mathit{flattened}(\tau)$. Then, by Skolemization, all remaining existential variables are removed. Finally, basic equivalence rules of FO logic allow us to "unfold" the nesting levels, resulting in a "flat" SO tgd. □

We want to stress that the fragment of nested$^{ord}$ tgds includes the special cases of default Skolemization as mentioned in [16] and the case of nested tgds without function variables over purely relational schemas as defined in [36]. Below we show that model checking is PSPACE-complete in all these cases.

THEOREM 5.4. *The nested$^{ord}$ tgd model checking problem is PSPACE-complete (both, query and combined complexity). The problem remains PSPACE-complete if* $\Sigma$ *contains a single nested$^{ord}$ tgd only, and even for the special cases of default Skolemization (as defined in [16]) and the case of nested tgds without function variables over purely relational schemas (according to [36]).*

PROOF SKETCH. *Membership* follows immediately from Theorem 5.3. Note that the fact that in the nested tgds defined in [36] the $\psi_i$ may be empty has no effect on the membership result, as it still allows for rewriting into an FO formula.

We show the PSPACE-*hardness* for nested tgds without function variables over a purely relational schema. Obviously, this is a special case of the other two classes of nested tgds mentioned in the theorem. The proof is done by reduction from the well-known PSPACE-complete problem QSAT and works similarly to the $\Pi_2 P$-hardness proof for the FO tgd model checking problem. We use the same source and target schemas and instances as in the proof of Theorem 3.1. Let an arbitrary instance of QSAT be given by the QBF $\forall\vec{x}_1 \exists\vec{y}_1 \ldots \forall\vec{x}_m \exists\vec{y}_m(\varphi)$. W.l.o.g. we assume that the innermost quantifier is "$\exists$" and $\varphi$ is in 3-CNF, i.e., $\varphi = C_1 \wedge \cdots \wedge C_n$ with $C_i = (l_{i,1} \vee l_{i,2} \vee l_{i,3})$). We define the following nested tgd $\tau$ with

$$\tau = \forall\vec{x}_1, \vec{x}'_1(\textstyle\bigwedge_{i=1}^{|\vec{x}_1|} P(x_{1,i}, x'_{1,i}) \rightarrow (\exists\vec{y}_1, \vec{y}'_1\ (\textstyle\bigwedge_{i=1}^{|\vec{y}_1|} Q(y_{1,i}, y'_{1,i}) \wedge (\tau_2)))),$$

where (for $i \in \{2, \ldots, m-1\}$)

$$\tau_i = \forall\vec{x}_i, \vec{x}'_i(\textstyle\bigwedge_{j_i=1}^{|\vec{x}_i|} P(x_{i,j_i}, x'_{i,j_i}) \rightarrow (\exists\vec{y}_i, \vec{y}'_i(\textstyle\bigwedge_{j_i=1}^{|\vec{y}_i|}(Q(y_{i,j_i}, y'_{i,j_i}) \wedge (\tau_{i+1}))))),$$ and

$$\tau_m = \forall\vec{x}_m, \vec{x}'_m(\textstyle\bigwedge_{i=1}^{|\vec{x}_m|} P(x_{m,i}, x'_{m,i}) \rightarrow (\exists\vec{y}_m, \vec{y}'_m(\textstyle\bigwedge_{i=1}^{|\vec{y}_m|} Q(y_{m,i}, y'_{m,i}) \wedge \textstyle\bigwedge_{i=1}^{n} C(l^*_{i,1}, l^*_{i,2}, l^*_{i,3})))).$$

Thereby all further definitions ($x'_i$, $l^*_{i,j}, \ldots$) are as in the proof of Theorem 3.1. □

Again, the update variants of the problem have the full complexity of the general model checking problem.

THEOREM 5.5. *Nested$^{ord}$ tgd insert-update model checking and nested$^{ord}$ tgd delete-update model checking are PSPACE-complete (both, query and combined complexity). They remain PSPACE-complete even if* $\Sigma$ *contains a single nested$^{ord}$ tgd and* $U$ *contains a single atom only.*

PROOF IDEA. *Membership* follows directly from Theorem 5.4. The *hardness* of the two problems is shown by adapting the proof ideas from the FO and SO cases to the reduction in the proof of Theorem 5.4. □

# 6. CONCLUSION

In this paper, we have studied the complexity of the model checking problem of three kinds of tgds (FO, SO, and nested tgds). The data complexity of these problems has been known to be tractable

(for FO tgds) and NP-complete (for SO tgds and nested tgds). For the combined complexity and the query complexity of these problems, we have proved completeness in the classes $\Pi_2 P$ (for FO tgds) and NEXPTIME (for SO and nested tgds), respectively. Moreover, we have shown that the update variants of these problems (i.e., a database which previously satisfied all tgds is updated by deleting or inserting a tuple) retain the full complexity. However, we have also identified sufficient criteria to reduce these high complexities. Technically, we have developed a new fixed-parameter algorithm for counting the solutions of CQs with bounded treewidth and we have introduced a new class of Boolean formulae, namely SO-QBFs (Second-Order QBFs). The latter may play as generic NEXPTIME-complete problem a similar role to QBFs in PSPACE.

Future work in this area should pursue two lines of research: On the one hand, the search for fragments of the model checking problem with lower complexity (ideally, tractable fragments) should be continued. On the other hand, the investigation of the combined complexity and the query complexity (which was initiated in [27] and continued here) should be extended to many more fundamental problems in the area of schema mappings and data exchange. It is to be expected that these complexities are much higher than the data complexity, which has been mainly studied so far. But then, it is even more important to understand the source of the high combined complexity and query complexity and to search for fragments with significantly lower complexity.

# 7. REFERENCES


[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proc. PODS'98*, pages 254–263. ACM Press, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*. Addison-Wesley, 1995.

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

[4] 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.

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

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

[7] C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[8] W. Conradie, V. Goranko, and D. Vakarelov. Algorithmic correspondence and completeness in modal logic. I. The core algorithm SQEMA. *Log. Methods in Comp. Sci.*, 2(1), 2006.

[9] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, New York, 1999.

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

[11] 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.

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

[13] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

[14] J. Flum, M. Frick, and M. Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. Springer, 2006.

[16] A. Fuxman, M. A. Hernández, C. T. H. Ho, R. J. Miller, P. Papotti, and L. Popa. Nested mappings: Schema mapping reloaded. In *Proc. VLDB'06*, pages 67–78, 2006.

[17] D. Gabbay, R. Schmidt, and A. Szalas. *Second-Order Quantifier Elimination: Foundations, Computational Aspects and Applications*, volume 12 of *Studies in Logic*. College Publications, 1995.

[18] G. Gottlob, N. Leone, and F. Scarcello. Robbers, marshals, and guards: game theoretic and logical characterizations of hypertree width. *J. Comput. Syst. Sci.*, 66(4):775–808, 2003.

[19] G. Gottlob and A. Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.

[20] T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen. Update exchange with mappings and provenance. In *Proc. VLDB'07*, pages 675–686. VLDB Endowment, 2007.

[21] M. Grohe, T. Schwentick, and L. Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.

[22] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. ICDE'03*, page 505. IEEE Computer Society, 2003.

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

[24] 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.

[25] T. Kloks. *Treewidth: Computations and Approximations*. Springer, Berlin, 1994.

[26] P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proc. PODS'05*, pages 61–75. ACM, 2005.

[27] P. G. Kolaitis, J. Panttaja, and W. C. Tan. The complexity of data exchange. In *Proc. PODS'06*, pages 30–39. ACM, 2006.

[28] L. Kot and C. Koch. Cooperative update exchange in the Youtopia system. *PVLDB*, 2(1):193–204, 2009.

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

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

[31] M. Meier. Towards rule-based minimization of RDF graphs under constraints. In *Proc. RR'08*, volume 5341 of *LNCS*, pages 89–103. Springer, 2008.

[32] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, New York, 2006.

[33] C. H. Papadimitriou. *Computational complexity*. Addison Wesley, 1994.

[34] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *Proc. VLDB'02*, pages 598–609, 2002.

[35] M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

[36] B. ten Cate and P. G. Kolaitis. Structural characterizations of schema-mapping languages. In *Proc. ICDT'09*, volume 361 of *ACM Int. Conf. Proc. Series*, pages 63–72. ACM, 2009.

[37] M. Y. Vardi. The complexity of relational query languages. In *Proc. STOC'82*, pages 137–146. ACM, 1982.