Computing Cores for Data Exchange: New Algorithms and Practical Solutions

Georg Gottlob
Vienna University of Technology
gottlob@dbai.tuwien.ac.at

ABSTRACT

Data Exchange is the problem of inserting data structured under a source schema into a target schema of different structure (possibly with integrity constraints), while reflecting the source data as accurately as possible. We study computational issues related to data exchange in the setting of Fagin, Kolaitis, and Popa (PODS'03). We use the technique of hypertree decompositions to derive improved algorithms for computing the core of a relational instance with labeled nulls, a problem we show to be fixed-parameter intractable with respect to the block size of the input instances. We show that computing the core of a data exchange problem is tractable for two large and useful classes of target constraints. The first class includes functional dependencies and weakly acyclic inclusion dependencies. The second class consists of full tuple generating dependencies and arbitrary equation generating dependencies. Finally, we show that computing cores is NP-hard in presence of a system-predicate NULL(x), which is true iff x is a null value.

1. INTRODUCTION

Data Exchange Settings. A clear and comprehensive framework for data exchange in the relational context was recently developed by Fagin, Kolaitis, Miller, and Popa [13, 14], and a schema mapping tool *Clio* based on these ideas was implemented at the IBM Almaden Research Center [33, 34]. In the present paper we study computational questions concerning the data exchange problem in this framework. According to [13, 14], a *data exchange setting* $\sigma = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ consists of two relational database schemas $\mathbf{S}$ and $\mathbf{T}$, called the source and the target schema, respectively, a set Σ_{st} of logical source-to-target constraints expressing relationships between source and target data, and a set Σ_t of target constraints (see also Section 2). The associated *data exchange problem* consists in finding an appropriate target instance J for a given source instance I. If $\langle I, J \rangle$ satisfies all constraints of Σ_{st} and Σ_t, then J is called a *solution* to the data exchange problem. However, in most cases there are multiple solutions to a data exchange problem, and the goal is to compute one that best reflects the source data.

Example. Consider a source schema $\mathbf{S}$ = *(student,employee)* where the source relations

student(STUDNAME,BIRTHDATE,SSN) and
employee (EMPNAME,SSN,PHONE)

store data on students and employees of a university, respectively, and where the target schema $\mathbf{T}$ has a single relation *person(NAME,BIRTHDATE,SSN,PHONE)* which should hold the data of both students and employees. The source-to-target constraints are expressed through the following two formulas (which are tuple-generating dependencies [7] over the combined source-target schema):
d1 : $\forall x \forall y \forall z(student(x, y, z) \rightarrow (\exists u\, person(x, y, z, u)))$,
d2 : $\forall x \forall y \forall z(employee(x, y, z) \rightarrow (\exists u\, person(x, u, y, z)))$.
The target constraints Σ_t consist of the two functional dependencies $SSN \rightarrow NAME$ and $SSN \rightarrow BIRTHDATE$.

Now, consider the following source instance I, where Morris is both a student and an employee, say, a TA.

student:

STUDNAME	BIRTHDATE	SSN
Maxwell	1980	12345
Morris	1982	10022
Bolte	1979	25555

employee:

EMPNAME	SSN	PHONE
Lempel	99999	2020
Morris	10022	3030

Formally, the following relations *person1*, *person2*, and *person3* are all solutions to this data exchange problem. The variables x_i are so called *labeled null values* (short: *nulls*):

person1:

NAME	BIRTHDATE	SSN	PHONE
Maxwell	1980	12345	x_1
Morris	1982	10022	x_2
Bolte	1979	25555	x_3
Lempel	1980	99999	2020
Morris	1982	10022	3030
Tarski	1902	77777	0665

person2:

NAME	BIRTHDATE	SSN	PHONE
Maxwell	1980	12345	x_1
Morris	1982	10022	x_2
Bolte	1979	25555	x_3
Lempel	x_4	99999	2020
Morris	1982	10022	3030

person3:

NAME	BIRTHDATE	SSN	PHONE
Maxwell	1980	12345	x_1
Bolte	1979	25555	x_3
Lempel	x_4	99999	2020
Morris	1982	10022	3030

Clearly, *person1*, containing "invented" data not stemming from the source (the Tarski record and the birthdate for Lempel) has to be rejected.

Universal Solutions. To avoid such solutions, Fagin et al. [13] define the concept of *universal solutions*. A solution J is *universal* if it can be homomorphically mapped into each other solution, i.e., if for each other solution J', there is a substitution h mapping each variable of J into a variable or constant of J' such that h applied to J is contained in J', denoted by $h(J) \subseteq J'$. Note that the solution *person1* is not universal, because every image of *person1* by a homomorohism (i.e., a variable substitution) h will still contain e.g. the constant Tarski and thus cannot be a subset of other solutions such as *person2* or *person3*. It can be seen, on the other hand that both *person2* and *person3* are universal solutions.

Fagin et al. [13] have shown that under rather general conditions on the type of dependencies allowed to occur in Σ_{st} and Σ_t, a universal solution to a data exchange problem exists, and a *canonical* universal solution can be computed in polynomial time from a source instance I by *chasing* these dependencies over the schemas $\mathbf{S}$ and $\mathbf{T}$ starting with I and an empty relation for $\mathbf{T}$. The chase procedure [7, 32] is a well known method for enforcing dependencies on instances with nulls. In particular, it was shown in [13] that if Σ_{st} consists of *tuple-generating dependencies (TGD's)* [7] (generating target tuples from source tuples), and if Σ_t consists of TGDs and *equality-generating dependencies (EGDs)* [7], and if the TGDs of Σ_t are weakly acyclic (see Section 2), then a universal solution, if one exists, can be generated in polynomial time by first chasing Σ_{st} over $\mathbf{S}$ and $\mathbf{T}$ and then chasing Σ_t over $\mathbf{T}$. In our example, where the dependencies are of the required types, this procedure yields relation *person2* as result (or an isomorphic copy of it with renamed variables). Intuitively, this is not quite the most "natural" solution, because the first Morris-tuple in *person2* is redundant and can be eliminated. Moreover, in general, a universal solution is not unique, and (logically equivalent) universal solutions can substantially differ in size. In our example, solution *person3* is clearly preferrable to *person2*. To formally capture this type of preference, Fagin et al. in [14] establish minimality as a key criterion for the quality of a universal solution and thus resort to the concept of the *core*.

The concept of Core. A *core* J of an instance I with labeled nulls is a minimal substructure of I which is also a homomorphic image of I, i.e., such that $J = I\theta$ for a uniform replacement θ of the variables of I by variables or constants. Since all cores of I are isomorphic (i.e., equal up to variable renaming), we usually speak of *the core* of an instance I and write $\underline{\text{Core}}(I)$. As shown in [14], all universal solutions to a data exchange problem have the same core (up to isomorphism). The core of any solution to a data exchange problem π is called *the core of* π and denoted by $\underline{\text{Core}}(\pi)$. In our example, the instance *person3* constitutes the core.

Cores were recently also considered in the context of the Semantic Web. In particular, it was shown that computing cores of RDF graphs can be useful for obtaining minimal representations and normal forms of such graphs [23].

Fagin et al. [13, 14] have shown that universal solutions, and in particular the core of a data exchange problem, can be used very profitably for query answering. Let Q be a union of conjunctive queries over the target schema of a data exchange problem π. The *certain answers* of Q for π consist of all tuples t such that t is ground (i.e., is made of constants only and does not contain nulls) and t occurs in *each* query answer $Q(J)$ for *each* solution J to π. These certain answers can be simply obtained from $\underline{\text{Core}}(\pi)$ by first evaluating Q over $\underline{\text{Core}}(\pi)$ and then taking the ground tuples of this result [13, 14] .

Since the core is in many contexts apparently the most satisfactory solution to a data exchange problem, it makes sense to study computational issues and the complexity of "getting to the core". This is a topic of [14] and the main topic of the present paper.

Computing Cores of Arbitrary Instances. A first important question is how to compute the core of an arbitary instance J with labeled nulls. As noted in [14], computing $\underline{\text{Core}}(I)$ for an instance I is the same as minimizing a conjunctive query, which is a well-known NP-hard problem [9]. Thus, computing $\underline{\text{Core}}(I)$ is NP-hard. However, Fagin et. al [14] identified as an important parameter, the block size of I, whose boundedness implies the tractability of the core computation problem. Let us briefly define the notion of block size. The connection graph $G(I)$ of an instance I is the undirected graph whose vertices are the nulls (=variables) of I and which contains an edge $\{x, y\}$ if x and y jointly appear in a tuple (=atom) of I. Each connected component of $G(I)$ is called a *block* of I. The maximum size of a block of I is termed the *block size* of I and is denoted by *blocksize*(I) . Fagin et. al [14] showed the following:

PROPOSITION 1.1 ([14]). *The core of an instance I of size n having blocksize$(I) \leq b$ can be computed in time $O(n^{b+3})$.*

Fagin et al. [14] ask whether there are better algorithms for computing $\underline{\text{Core}}(I)$ for an instance I. This is one of the problems we address in the present paper.

Computing Cores of Data Exchange Problems. The next question is how to compute cores of data exchange problems, more precisely, of universal instances of data exchange problems. Such universal instances are produced from ground (i.e., null-value free) relations through the application of dependencies and may thus have certain properties that may lead to polynomial core computation algorithms. We always assume a *fixed* data exchange setting σ. The problem is then to compute $\underline{\text{Core}}(\pi)$ for the data exchange problem $\pi = \langle \sigma, I \rangle$ where I is a (variable) source instance. Since σ is fixed, the size n of π can be identified with the size $\|I\|$ of the source instance I. Fagin et al. [14] proved the following result:

PROPOSITION 1.2. *Let $\sigma = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ be a fixed data exchange setting. If Σ_{st} consists of source-to-target TGDs and Σ_t consists of EGDs, then the core of $\langle \sigma, I \rangle$ can be computed in polynomial time from a source instance I.*

The key finding for establishing Proposition 1.2 was the observation that under the premises of this proposition, the block size of the canonical solution of a data exchange problem is bounded by a constant b, and thus, by Proposition 1.1, the core can be computed in polynomial time.

Of course it would be interesting to know whether the core computation problem remains tractable when Σ_t contains TGDs in addition to EGDs (assuming that the chase terminates), or at least practical subclasses of TGDs. A more specific question asked in [14] is whether the core computation problem is tractable for EGDs and *full* TGDs, that is, TGDs which create no new null values (see Section 2).

New Results. The present paper takes the above open problems as starting point for an investigation into the complexity of core computation in the context of data exchange. We first study core computation from arbitrary instances and then turn our attention to instances that arise in the context of data exchange problems. The following is a short summary of our results:

▶ In Section 3 we improve the bound of $O(n^{b+3})$ of Proposition 1.1 to $O(n^{\lfloor b/2 \rfloor+2})$, thus halfing the exponent. We show this by using a reduction of core computation to the conjunctive query containment problem and using an algorithm for query containment that decides whether a query Q contains a query Q' in time $O(n^{\lfloor v/2 \rfloor+1})$, where v is the number of variables in Q. The algorithm uses recently introduced methods of hypertree decompositions [18] and exploits the fact that the hypertree-width of each conjunctive query having v variables is bounded by $\lfloor v/2 \rfloor + 1$.

As an intermediate result, which may be if independent interest, we establish that each conjunctive query Q with v variables can be answered in time $O(\|Q\|)+O(|Q|\times a^{\lfloor v/2 \rfloor+1})$ over a database whose largest relation has size a.

▶ In Section 4 we show that the problem of computing the core of an instance J is fixed-parameter intractable [11] w.r.t. the block size b of J, and this is even so if J is the canonical solution of a data exchange problem. This shows that there is little hope to eliminate the parameter b from the exponent of the upper bound of core computation.

▶ In Section 5 we define the *simple TGDs* as a class of TGDs whose left sides underly some restrictions. Simple TGDs are, however, more expressive than the well-known inclusion dependencies [8]. We prove that the core computation problem is tractable for data exchange settings where Σ_{st} consists, as usual, of source-to-target TGDs, and where the set Σ_t of target constraints consists of a set of weakly acyclic simple TGDs. In particular, it follows that the problem is tractable if Σ_t contains functional dependencies and acyclic (actually, even weakly acyclic) inclusion dependencies. Note that functional and inclusion dependencies are considered by many as the most important dependencies cf. [8, 6, 28, 29]. Together, they can be used, e.g., to express foreign key constraints and other referential integrity conditions. We further show in Section 5 that the hypertree width of an instance is essentially preserved while chasing weakly acyclic simple TGDs, which leads to improved methods for dealing with TGDs. Hence, we believe to have identified a relevant practical tractable framework, which widens the range of applicability of the approach of Fagin et al. [14].

▶ In Section 6 we show that computing cores of data exchange problems whose target constraits consists of full TGDs and arbitrary EGDs is tractable. This solves an open problem of [14]. We start by showing the result for full TGDs alone. The proof uses a new algebraic interchange lemma stating that a full-TGD chase commutes with a homomorphism in case the homomorphism is idempotent. The result is extended to cover full TGDs plus EGDs by a new method of simulating EGDs by full TGDs. This tractable class includes both join dependencies (and thus, in particular, multivalued dependencies) and functional dependencies.

▶ Finally, in Section 7, we study a slight extension of the data exchange framework defined by Fagin et al. [13, 14]. We assume that there exists a predefined system predicate $NULL(x)$ which evaluates to true only if x is (bound to) a null value, in short, which can be used to distinguish nulls from non-nulls. Obviously, to make sense, such a predicate is not to be considered part of the source or target schema, and is not to be altered by homomorphisms or dependencies (EGDs or TGDs). Considering such a $NULL$ predicate is not unrealistic, since similar features exist in classical databases. We show that the core computation problem in such an extended setting is NP hard, even when the target dependencies consist of a single acyclic full TGD. What this shows at least is that one has to be careful when deviating from the framework proposed in [13, 14]. Adding a simple polynomially computable predicate such as $NULL(x)$ may dramatically enhance the expressive power of the constraints so to make the core computation problem intractable.

Future work. In the present paper we show that computating cores is tractable for two important classes of data exchange problems involving TGDs. We suspect that there are larger tractable classes, and hope to determine them. Another important issue is query answering. It was shown that the positive results concerning the computation of the certain answers for unions of plain conjunctive queries (undoubtedly the most important class of queries) do not carry over to more complex types of queries [14], not even if acceptable methods of query rewriting are used [5]. We believe that more work is needed to extend the current data exchange framework to deal with more complex queries.

2. PRELIMINARIES AND NOTATION

We adhere to the usual terminology of database research [1, 39] and to the concepts and notions on data exchange used in [13, 14], many of which were already introduced in Section 1. In this section, we introduce additional notation, conventions, and definitions.

We use the expressions *variable* and *labeled null value* as synonyms. An *instance* is a finite relation over constants and variables. Each *tuple* $\langle t_1, \ldots, t_r \rangle$ of a relation r is identified with the logical *atom* $r(t_1, \ldots, t_r)$. If ξ is an object (e.g., an atom, tuple, formula, instance etc.) in which variables and/or constants occur, then $var(\xi)$ denotes the set of variables in ξ, and $const(\xi)$ the set of constants in ξ. We define $dom(\xi) = const(\xi) \cup var(\xi)$. If ξ contains no variables, then ξ is said to be *ground*. As in [13], we assume that source instances of a data exchange problem are ground. We adopt the RAM model for our complexity bounds. The cardinality of a set S is denoted by $|S|$, while the *size* of an object ξ is denoted by $\|\xi\|$. For an instance K and a set of variables $B \subseteq var(K)$, denote by $K[B]$ all non-ground atoms A of K such that in $var(A) \subseteq B$. If K is an instance and x a variable, then $atoms(x, K)$ denotes the set of all atoms of K in which x appears as argument.

Let K and M be instances. A maping $f : dom(K) \to dom(M)$ is *legal* if for each constant $c \in const(K)$, $f(c) = c$. We denote the set of legal mappings from K to M by $legal(K, M)$. A legal mapping $h : dom(K) \longrightarrow dom(M)$ is a *homomorphism* if $\forall A \in K : h(A) \in M$. An endomorphism of M is a homomorphism from M to M. The set of all endomorpohisms of M is denoted by $end(M)$.

Recall the notion of *blocks* of variables introduced in [13], and already explained in the introduction. The set of all blocks of K is denoted by $blocks(K)$. Let $x \in var(K)$ then $block(x, K)$ denotes the block of K containing x. Let $V \subseteq var(K)$, then $blocks(V, K) = \bigcup_{x \in V}\{block(x, K)\}$, i.e., $blocks(V, K)$ is the set of all blocks of K that contain at least one variable from V. If B is a block of K, then $atoms(B, K)$ denotes the set of atoms $A \in K$, such that $var(A) \subseteq B$. Such a set $atoms(B, K)$ is called an *atom-block* of K. The *block size* of an instance K, denoted by $blocksize(K)$ is the maximum number of variables appearing in a block of K. The *block width* $blockwidth(K)$ of K is the maximum number of atoms appearing in an atom-block of K.

Throughout the paper, we assume that the set Σ_{st} of a data exchange problem $\sigma = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$ consists of TGDs of the form $\forall \mathbf{x}(\phi_{\mathbf{S}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$, where $\phi_{\mathbf{S}}(\mathbf{x})$ is a conjunction of atomic formulas over $\mathbf{S}$ with variables $\mathbf{x}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{T}$ whose variables are among those in $\mathbf{x}$ and $\mathbf{y}$. Such TGDs are also referred to as *source-to-target TGDs.* Each target constraint from Σ_t is either a TGD on $\mathbf{T}$, i.e. a formula of the form $\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y}))$, or an EGD, i.e., an equality-generating dependency [7] of the form $\forall \mathbf{x}(\phi_{\mathbf{T}}(\mathbf{x}) \rightarrow (x_1 = x_2))$. In these dependencies, $\phi_{\mathbf{T}}$ and $\psi_{\mathbf{T}}(\mathbf{x}, \mathbf{y})$ are conjunctions of atomic formulas over $\mathbf{T}$, and x_1, x_2 are among the variables in $\mathbf{x}$. As in [13, 14], we may drop the universal quantifiers in front of a dependency.

A TGD is *full* if no existential quantifier occurs in it. The relation graph $RG(\Sigma)$ of a set Σ of TGDs is the directed graph whose vertices are the relations mentioned in Σ and which contains an edge $R \rightarrow S$ if relation R appears in the lhs of a TGD of Σ and S in the rhs of the same TGD. A set Σ of TGDs is *acyclic* if $RG(\Sigma)$ is acyclic.

A similar, but more general concept is the notion of a *weakly acyclic set of TGDs* [14]. Let Σ be a set of TGDs over a fixed schema. The *dependency graph* $DG(\Sigma)$ of Σ has as vertices all *positions* of Σ, i.e., all pairs (R, A), where R is a relation symbol of the schema and A an attribute of R. The set of edges is defined as follows: for every TGD $\phi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ and for every x that occurs in ψ:

For every occurrence of x in Φ in position (R, A_i)

- **(a)** For every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$ (if it does not already exist).
- **(b)** In addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a *special edge* $(R, A_i) \Rightarrow (T, C_k)$ (if it does not already exist).

Σ is *weakly acyclic* if the dependency graph has no cycle going through a special edge.

The *chase* procedure "enforces" EGDs by unifying labeled nulls with other labeled nulls or with constants, and TGDs by creating new target tuples where necessary. If a new tuple is created via a TGD, the existentially quantified variables in the corresponding atom in the rhs of the TGD are reflected by new labeled nulls in the new tuple. We refer the reader to [13] for a formal definition of the *chase* procedure in the context of data exchange. Chasing a set Σ of dependencies on an instance K may produce various results, depending on the order of application of the dependencies. We denote the result by $CHASE_{\Sigma}(K)$, assuming some arbitrary, but fixed order. Note that the result of chasing a set of *full TGDs* on an instance K is order-independent and thus unique.

3. CORE COMPUTATION USING HYPERTREE DECOMPOSITIONS

Logically, a database instance I with nulls represents an existentially quantified conjunction of atoms (=tuples) whose arguments are constants or variables. Boolean Conjunctive queries (BCQs) are defined the same way. The difference is just in the use: An instance *affirms* that a conjunction of atoms is true, a BCQ *asks* whether a conjunction of atoms is true in a database. For the aims of this section this difference is not essential, and we will thus identify instances with Boolean conjunctive queries. Via this trivial identification it is clear that, as already noted in [14], computing the core of instance I is the same as minimizing the query I. A query or instance I *contains* a query or instance J, denoted by $I \triangleright J$, if for each database db the result $J(db)$ of evaluating J over db is set-theoretically contained in the result $I(db)$ of evaluating I over db. As shown in [9], $I \triangleright J$ iff there exists a homomorphism from I to J.

The following algorithm computes the core $\underline{\text{Core}}(I)$ of an instance I by reducing this task to $|var(I)|$ calls of a query containment check by exploiting the blocks of I.

Algorithm 3.1. CORECOMP
Input: *An instance I.*
Output: *The core of I (up to isomorphism).*

```
1. Identify the blocks of $I$, and let $B(x)$ be the block of $x$
   in $I$ for each variable $x \in var(I)$.
2. $K := I$;
3. FOR each $x \in var(I)$ DO
   IF $x \in var(K)$ AND $K[B(x)] \triangleright (K - atoms(x, K))$
   THEN $K := K - atoms(x, K)$;
4. Output $K$
```

The correctness of the CORECOMP algorithm follows from results in [14]. Lemma 3.2, whose proof is available in [16], asserts this correctness and states a complexity bound for CORECOMP modulo containment checks.

Lemma 3.2. *The* CORECOMP *algorithm is correct and runs in time $O(n^2) + |var(I)| \times C_{\triangleright}$, where $C_{\triangleright}$ is the maximum cost for a test $K[B(x)] \triangleright (K - atoms(x, K))$ performed by* CORECOMP.

We are thus looking for a good algorithm for query containment checks $K[B(x)] \triangleright (K - atoms(x, K))$, where the performance is measured in terms of the block size of $B(x)$, i.e., in terms of the variables of the query $K[B(x)]$. While there are several papers describing methods for conjunctive query evaluation, query containment, and query minimization, (e.g. [9, 3, 4, 36, 10, 30]), some of which also discuss polynomial special cases, to our best knowledge, none of these papers presents or suggests an algorithm which, for queries of b variables has a worst case behaviour better than $O(n^b)$, and whose use would thus significantly improve the worst case bound of $O(n^{b+3})$ for computing cores of instances having maximum block size b. However,in [17], in the context of automated theorem proving, an algorithm *Division into Components (DC)* for clause subsumption was presented (a problem algorithmically equivalent to conjunctive query containment, cf. [19]), which can solve the subsumption test and thus query containment in time $O(pol(n) \times n^{\lfloor b/2 \rfloor + 1})$ for some polynomial pol. This shows that significantly better bounds than the bound $O(n^{b+3})$ stated in Proposition 1.1 can be obtained for core computation. While the DC algorithm has been successfully used in practice, it has a somewhat large polynomial overhead ($pol(n)$) and is not perfectly suited for deriving our desired upper bounds. Rather than using DC, we will thus use methods based on the more recent concept of *hypertree decomposition* [18], also studied in [20, 21, 22]. For the purposes of the present paper, it is sufficient to formally define a *restricted* version of hypertree decomposition.

Definition 3.3. *Let Q be a conjunctive query.*

- *A* hypertree *for Q is a tree whose vertices are sets of query atoms. These vertices are often referred to as* hypernodes.

- A restricted hypertree decomposition (rhd) *for Q is a hypertree T for Q such that:*
  - *Each query atom is contained in at least one hypernode of T.*
  - *For each variable $x \in var(Q)$, the subgraph of T induced by those hypernodes of T containing x is connected.*
- *The* width *of a rhd T is defined as $max_{v \in T}|v|$.*
- *The restricted hypertree width $rhw(Q)$ of Q is the minimum width over all rhds of Q.*

The more general concept of *hypertree decomposition* [18] allows partial atoms to occur at hypernodes, too. A formal definition is given in [18, 16]. For the associated notion of *hypertree width $hw(Q)$* of a query Q, it holds that $hw(Q) \leq rhw(Q)$, and in some cases $hw(Q) < rhw(Q)$ [18].

The next proposition follows from results in [18] (or also [10, 21]) and from the fact that joins and semijoins can be performed in linear time on a RAM (cf. the Appendix of [15]).

PROPOSITION 3.4. *Let T be a given rhd or hd of width k for a query Q, then then Q can be evaluated over a database db in time $O(t \times a^k)$, where a is the size of the largest relation in db, and t is the number of hypernodes of T.*

THEOREM 3.5. *For each conjunctive query Q having b variables, $hw(Q) \leq rhw(Q) \leq \lfloor b/2 \rfloor + 1$.*

PROOF. It suffices to show the bound for rhw. Construct a set $R \subseteq Q$ as follows. Let R initially be the empty set and repeat as long as possible the following: Choose "greedily" an arbitrary atom A from $Q - R$ where A contains two variables x and y such that $x \notin var(R)$ and $y \notin var(R)$ and add A to R. Observe that R consists of at most $\lfloor b/2 \rfloor$ atoms, given that the number $|var(R)|$ of vertices (=variables) of R is at least twice the number of atoms in R. Note that in case all atoms have arity 2, R is just a *maximal matching* of the query graph. Observe that, by construction of R, every atom $A \in Q - R$ has at most one variable not matched by R, i.e., $|var(A) - var(R)| \leq 1$.

For each variable $x \in var(Q) - var(R)$, let $S(x) = \{A \in Q | x \in var(A)\}$ be the set of all atoms of Q containing x. Note that for $x \neq y$ it holds that $S(x) \cap S(y) = \emptyset$. A rhd T for Q is constructed as follows:

- The root of T is the set R. For each atom $A \in Q - R$ such that $var(A) \subseteq var(R)$, we attach a separate child $\{A\}$ to R; this child will be a leaf in T.
- For each variable $x \in var(Q) - var(R)$, we construct a hypernode $H(x)$ by picking an arbitrary atom $A(x) \in S(x)$ and letting $H(X) = R \cup \{A(x)\}$. Note that $|H(x)| \leq \lfloor b/2 \rfloor + 1$. For each $x \in var(Q) - var(R)$, we attach $H(x)$ as a child of R.
- The third level of T is constructed as follows. For each $x \in var(Q) - var(R)$ and for each atom $A \in S(x) - A(x)$, i.e., for each remaining atom not yet covered containing x, we create a singleton hypernode $\{A\}$ and attach it as a child to $H(x)$.

The conditions of Def. 3.3 for a rhd are clearly satisfied. Thus T constitutes a *rhd* of width $\leq \lfloor b/2 \rfloor + 1$. □

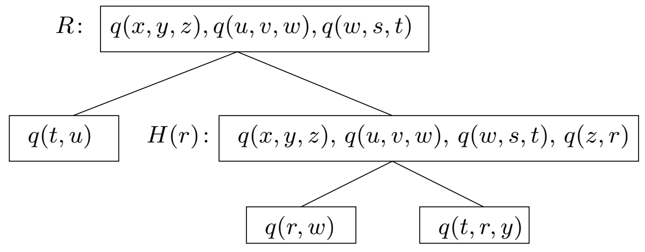
R: q(x,y,z), q(u,v,w), q(w,s,t)
q(t,u)
H(r): q(x,y,z), q(u,v,w), q(w,s,t), q(z,r)
q(r,w)
q(t,r,y)

Figure 1: Decomposition of query $Q1$

EXAMPLE 3.6. *To illustrate the proof of Theorem 3.5, consider the query $Q1 : q(x,y,z) \wedge q(z,r) \wedge q(u,v,w) \wedge q(t,u) \wedge q(r,w) \wedge q(w,s,t) \wedge q(t,r,y)$. A restricted hypertree decomposition of $Q1$ generated by the method described in the proof of Theorem 3.5 is depicted in Figure 1.*

The same result holds if, instead of hypertree width, we use the related concept of *query width (qw)*[10]. However, for the better-known concept of *treewidth (tw)* [37, 27, 10], no similar upper bound can be achieved. In fact, a query Q having n variables can have $tw(Q) = n - 1$. In the sequel, we concentrate on the concept of hypertree width because, as shown in [18], determining whether $hw(Q) \leq k$ for a fixed constant k is tractable, while determining $qw(Q) = k$ is NP hard. Moreover, for each query Q, $hw(Q) \leq qw(Q) \leq tw(Q)$, but for some queries Q we have $hw(Q) < qw(Q) < tw(Q)$. Thus, hypertree decompositions are preferrable to the other decomposition methods.

LEMMA 3.7.
1. *Let Q be a Boolean conjunctive query with at most b variables and let db be a database. Denote by a the size of the largest relation of db. Evaluating Q over db is feasible in time $O(\|Q\|) + O(|Q| \times a^{\lfloor b/2 \rfloor + 1})$.*
2. *Let Q and Q' be conjunctive queries where Q has b variables. The check $Q \triangleright Q'$ can be performed in time $O(\|Q\|) + O(|Q| \times \|Q'\|^{\lfloor b/2 \rfloor + 1})$.*

PROOF. (Sketch) (1) If we just combined Proposition 3.4 with Theorem 3.5, we would get as upper bound somethig like $O(\|Q\|^2) + O(|Q^2| \times \|Q'\|^{\lfloor b/2 \rfloor + 1})$, where the first term accounts for computing an rhd T and the second for evaluating Q using T. This is because the size $\|T\|$ of the rhd T of Q from the proof of Theorem 3.5 can be quadratic in $\|Q\|$ in the worst case. Let us roughly sketch a smarter method which yields the desired bound. A detailed proof is given in [16]. Consider a rhd T as constructed for Q in the proof of Theorem 3.5. The "normal" way of evaluating Q based on this decomposition T is by associating a relation rel_v to each hypernode v of T (consisting of the join of all relations corresponding to the atoms of v) and then performing semijoins in a bottom up fashion and answer *yes* (*no*) if the root is nonempty (empty) after this process. We will shortcut the intermediate level. Let r be the root relation rel_{root}. First, for each v containing a singleton atom A, such that $var(A) \subseteq var(R)$, we reduce r via the semijoin $r \rtimes rel_v$. Next, let $x \in var(Q) - var(R)$. We expand r by joining it with the relation corresponding to an arbitrary atom $A_x \in S(x)$. We then reduce r via semijoins with the (relations corresponding to the) remaining atoms A from $S(x)$, and then we project out the x-column from r. We repeat this for each $x \in var(Q) - var(R)$. The final r is nonempty iff the query Q has a positive answer.

(2) Statement (2) follows from (1) and the well-known fact that query containment can be trivially reduced to Boolean query evaluation [9] (see [16] for more details). $\square$

By combining Lemma 3.2 and Lemma 3.7, we get:

Theorem 3.8. *Using algorithm* CORECOMP*, the core of a structure of size n and of block size b can be computed in time $O(n^{\lfloor b/2 \rfloor +2})$.*

From Proposition 3.4 and Lemma 3.2 we get similarily:

Theorem 3.9. *Let T be a given hd of width k for an instance I of size n, then $\underline{Core}(I)$ can be computed in time $O(t \times n^{k+1})$, where t is the number of hypernodes of t.*

While, by Theorem 3.5, the hypertree with of a query with b variables is *in the worst case* $\lfloor b/2 \rfloor + 1$, it is in practice often much smaller [38]. Many queries, and most likely also many relational instances with nulls that arise in practice are of hypertree width 2 or 3. Testing whether an instance has hypertree width k, where k is a constant, and computing – in the positive case – a hypertree decomposition of width k is feasible in polynomial time [18]. In practical contexts, it may thus make sense to recognize whether an instance with nulls has a very small hypertree width, or to try to use heuristics for obtaining hypertree decompositions of reasonably small width. Note that in a number of relevant cases, the hypertree width of a universal solution of a data exchange problem $\langle \sigma, I \rangle$ does not depend on the source instance I but just on the fixed data exchange setting σ. An example will be given at the end of Section 5.

4. FIXED-PARAMETER INTRACTABILITY

The improvement of the worst case upper bound for computing $\underline{Core}(J)$ from $O(n^{b+3})$ to $O(n^{\lfloor b/2 \rfloor +2})$ is certainly substantial, however, it would be interesting to know whether we could make a much more drastic improvement and eliminate the parameter b from the exponent of n. This amounts to ask whether the core computation problem is *fixed parameter tractable (fp-tractable)* or *fixed parameter intractable (fp-intractable)* wrt. parameter b [11]. The problem is fp-tractable if there exists a function f and a constant c such that the core of each instance Jf can be computed in time $O(f(b) \times n^c)$, where $n = \|J\|$ is the size of J, and where $f(b)$ depends only on b and c is independent of J and b. On the other hand, if the problem is fp-intractable, then it is most likely that the parameter b has to remain in the exponent of n, for otherwise unexpected collapses of parametrized complexity classes would happen. To prove that a problem is fp-intractable one usually reduces another problem, known to be fp-intractable, to it via a *parametric reduction* (see [12, 35] or also [16]for more details). Such a reduction involves a standard polynomial time reduction f between problem instances, and a mapping g between the parameters. A well-known fp-intractable problem is the k-CLIQUE problem: given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, the parameter, decide whether G has a clique of size k. From k-clique it was shown that Boolean conjunctive queries with k variables are fp-intractable [12, 35]. See [24] for other interesting fp-intractability results on query evaluation. Because of the close relationship between core computation and conjunctive query evaluation, the following is not very surprising.

Theorem 4.1. *For an instance J having blocksize$(J) \leq k$ (where k is the parameter), and a ground set $C \subseteq J$, it is fp-intractable to decide whether $\underline{Core}(J) = C$.*

Proof. From an instance (G, k) of k-CLIQUE, where $G = (V, E)$, compute a relational instance J by the following parametric reduction: $(G, k) \rightarrow (J, k)$ where $J = E \cup \{(x_i, x_j) \mid 1 \leq i, j \leq k\}$ such that $x_1, \ldots, x_k$ are mutually distinct variable symbols. Interpret the vertices of G as constants of J. It is easy to see that G has a clique of size k iff $\underline{Core}(J) = E$. This is effectively a parametric reduction (with parameter mapping $g(k) = k^2$). $\square$

Note that from Theorem 3.9 and from the fact that a hypertree decomposition of width c can be computed in polynomial time for each fixed constant c [18], it follows that the above problem (with the same block size parameter) becomes fp-tractable for instances where $hw(J) \leq c$. The same is true for weaker notions than hypertree width, such as *treewidth (tw)* [37, 27, 10]. In fact, as shown in [18, 21], for each structure I, $hw(I) \leq tw(I)$.

We can extend the notion of fixed parameter intractability to cover parameterized search problems by calling a search problem A with parameter k fp-intractable if its solution in time $O(f(k) \times n^c)$, where f depends only on k, and c is a fixed constant independent of k, would imply that some fp-hard decision problem is fp-tractable.

Theorem 4.2. *The following search problems are fixed parameter intractable wrt. parameters blocksize(J) and ℓ, respectively:*
P1: *Given an instance J, compute $\underline{Core}(J)$.*
P2: *Given a data exchange problem $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ with no target constraints, where the maximum number of variables occurring in a TGD of Σ_{st} is bounded by parameter ℓ, and a source instance I for σ, compute the core of the universal target instances of σ.*

Proof. The fp-intractability of P1 is obvious by Theorem 4.1. Now assume P2 is fp-tractable and can be solved in time $O(f(k) \times n^c)$. Let (G, k) be an instance of k-CLIQUE. Transform (G, k) into the data exchange problem $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \emptyset)$, where $\mathbf{S} = \{g(.,.)\}$ and $\mathbf{T} = \{g'(.,.)\}$, and where Σ_{st} consists of the following two TGDs:

ST1: $\forall x \forall y \; (g(x, y) \rightarrow g'(x, y))$.

ST2: $\exists x_1 \cdots \exists x_k (\bigwedge_{1 \leq i,j \leq k} g'(x_i, x_j))$.

Let the graph $G = (V, E)$ be the source instance of σ (where there is an atom $g(a, b)$ for each edge (a, b) in E). It is clear that the core of the universal solutions of σ is G iff G has a clique of size k. Note that the parameter ℓ of the resulting problem-instance of $P2$ is precisely equal to k. The reduction from k-CLIQUE to P2 requires time $O(k^2 + \|G\|)$. Thus the k-CLIQUE problem would be solvable in time $O(f(k) \times (k^2 + \|G\|)^c)$ which is $O(f(k) \times \|G\|^{\max\{2,c\}})$, given that $k \leq |V|$. Thus the k-CLIQUE problem would be fp-tractable. It follows that problem P2 is fp-intractable. $\square$

5. A TRACTABLE CLASS INVOLVING EGDS AND SIMPLE TGDS

In this section we define the class of *simple TGDs* as TGDs whose left sides consist of a single atom with no shared variables. We will show that weakly acyclic simple TGDs plus arbitrary EGDs form a class of constraints for which the core computation problem is tractable. This class is "practical" because it encompasses functional dependencies and acyclic inclusion dependencies, the arguably most widely used in database design, cf. [8, 6, 28, 29].

Definition 5.1. *A TGD is* simple *if it is of the form*

$$\begin{aligned}&\forall x_1 \cdots \forall x_k\,(R(x_1,\ldots,x_k) \rightarrow\\ &\quad \exists y_1,\cdots \exists y_m\,(R_1[x_1,\ldots,x_k,y_1,\ldots,y_m]\\ &\quad \wedge \cdots \wedge R_r[x_1,\ldots,x_k,y_1,\ldots,y_m])),\end{aligned}$$

where the $x_1 \ldots x_k, y_1, \ldots y_m$ are mutually distinct variables, and R and the $R_1 \ldots, R_r$ are database relation names, and for $1 \leq i \leq r$, $R_i[x_1,\ldots,x_k,y_1,\ldots,y_m]$ is a database atom whose arguments are among $x_1,\ldots,x_k,y_1,\ldots,y_m$.

Note that, in particular, inclusion dependencies (short: INDs) are simple TGDs. For example the inclusion dependency $R[AB] \subseteq S[EF]$ over the schemas $R(A,B,C)$ and $S(D,E,F)$ is equivalently expressed as the simple TGD

$$\forall x \forall y \forall z\,(R(x,y,z) \rightarrow \exists u\, S(u,x,y)).$$

Simple TGDs can also express some constraints on database schemas that are not directly representable by INDs.

Before presenting new material, we summarize some important results from [13] and [14].

Proposition 5.2 ([13]). *Let Σ be the union of a weakly acyclic set of TGDs with a set of arbitrary EGDs. Then there exists a polynomial q depending only on Σ, such that for each pair of instances I, J, whenever J is obtained by chasing I with Σ, then $|dom(J)| \leq q(|dom(I)|)$. Moreover, any such chase terminates in polynomial time.*

Definition 5.3 ([14]). *An endomorphism h of J is* useful *if $h(J) \subset J$, i.e., if $h(J) \neq J$.*

Definition 5.4 ([14]). *Let K and M be two instances such that the nulls of K form a subset of the nulls of M, that is, $var(K) \subseteq var(M)$. Let h be an endomorphism of M and let B be a block of nulls for K. We say that h is K*-local for *B if $h(x) = x$ whenever $x \notin B$. We say that h is K*-local *if it is K-local for B for some block B of K.*

Proposition 5.5 ([14]). *Let L be an instance and let M be the result of chasing a set Σ_E of EGDs on L. Then there exists a useful endomorphism of M iff there exists a useful L-local endomorphism of M. Moreover, the core of M can be computed by letting initially $K := M$ and by then successively identifying a useful L-local endomorphism h of K and letting $K := h(K)$ until no further useful L-local endomorphism of K can be found; the final value of K is the core of M.*

Proposition 5.6 ([14]). *Let N be an instance, let M be the result of chasing a set Σ_E of EGDs on N, and let $b = blocksize(N)$. Computing the core of M by the method stated in Proposition 5.5 takes at most time $O(n^{b+3})$, where $n = \|N\|$ is the size of N.*

A proof of this proposition[1] is given in [16].

[1]Note that in [14] this result is stated in a slightly different (but equivalent) form. There, n designates the maximum of the total number $\#e$ of elements and the total number $\#t$ of tuples of N. If α is the total number of columns in J (i.e., the sum of all arities of relations in J), we have: $\|N\|/\alpha \leq \max\{\#e, \#t\} \leq \|N\|$. Since α is a constant (because we assume fixed schemas), the quantity n used here and the n used in [14] are linearly related and do not substantially differ.

The following algorithm computes the core of the universal solutions for a source instance I to a data exchange setting $\sigma = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$, where $\Sigma_t = \Sigma_T \cup \Sigma_E$ consists of the union of a weakly acyclic set Σ_T of simple TGDs, and an arbitrary set Σ_E of EGDs.

Algorithm 5.7.
Input: *source instance I.*
Output: *the core of the universal solutions for I, if solutions exist, and "fail" otherwise.*

1. *Compute an instance J, by chasing I with Σ_{st}.*
2. *Compute an instance J' from J by chasing J with Σ_T.*
3. *Compute the blocks of J'.*
4. *Compute an instance J'' by chasing J' with Σ_E. If the chase fails, then stop with "fail". Otherwise, initialize K with J'', i.e., $K := J''$.*
5. *Check whether there exists a useful J'-local endomorphism h of K. If not, then stop with output K.*
6. *Update K to be $h(K)$, and goto step 5.*

Theorem 5.8. *Algorithm 5.7 is correct and runs in polynomial time.*

Proof. By Proposition 5.2, chasing J with weakly acyclic TGDs and arbitrary EGDs terminates in polynomial time. We claim that after having enforced all simple TGDs from Σ_T (step 2) and then all EGDs from Σ_E (step 4), the obtained set J'' is already a result of chasing J with $\Sigma_t = \Sigma_T \cup \Sigma_E$, i.e., all dependencies from $\Sigma_T \cup \Sigma_E$ are satisfied by J''. For dependencies from Σ_E, this is obvious, given that J'' is the result of chasing the dependencies of Σ_E on J'. Now consider a TGD τ from Σ_T. Note that chasing the EGDs from Σ_E on J' in Step 4 amounts to perform successive substitutions, replacing variables by other variables or by constants. Let $\theta : var(J') \longrightarrow var(J'') \cup const(J'') \subseteq var(J') \cup const(J')$ denote the global substitution performed by the chase in step 4, such that $J'\theta = J''$; this substitution[2] θ is obtained by suitably composing and combining the single substitutions performed by this chase. Given that τ is simple, it is of the form

$$\begin{aligned}&\forall x_1 \cdots \forall x_k\,(R(x_1,\ldots,x_k) \rightarrow\\ &\quad \exists y_1,\cdots \exists y_m\,(R_1[x_1,\ldots,x_k,y_1,\ldots,y_m]\\ &\quad \wedge \cdots \wedge R_r[x_1,\ldots,x_k,y_1,\ldots,y_m])),\end{aligned}$$

as in Definition 5.1. To show that τ is satisfied in J'', it suffices to prove that whenever there are terms $t_1,\ldots,t_k$ such that $R(t_1,\ldots,t_k) \in J''$, then there exist $y_1,\ldots,y_m$ such that for $1 \leq i \leq r$, $R_i[t_1,\ldots,t_k,y_1,\ldots,y_m] \in J''$. Thus assume that $R(t_1,\ldots,t_k) \in J''$. There must exist a database atom $R(t_1^*,\ldots,t_k^*) \in J'$ such that $t_1 = t_1^*\theta,\ldots,t_k = t_k^*\theta$. Since τ is satisfied in J', there must exist terms $y_1',\ldots,y_m'$ such that for $1 \leq i \leq r$, $R_i[t_1^*,\ldots,t_k^*,y_1',\ldots,y_m'] \in J'$. Thus, $R_i[t_1^*,\ldots,t_k^*,y_1',\ldots,y_m']\theta = R_i[t_1^*\theta,\ldots,t_k^*\theta,y_1'\theta,\ldots,y_m'\theta] = R_i[t_1,\ldots,t_k,y_1'\theta,\ldots,y_m'\theta] \in J''$ and therefore τ is satisfied in J''. Hence J'' satisfies Σ_t and J'' is the final result of a chase sequence, and is thus a universal instance[3].

[2]For any object ξ and substitution λ, $\xi\lambda$ stands for $\lambda(\xi)$.
[3]A similar distinction of two separate phases in the chase procedure for the more restricted context of functional dependencies and acyclic inclusion dependencies was already observed by [25] and used in [31, 28, 29].

By Proposition 5.5, steps 5 and 6 of Algorithm 5.7 correctly compute the core of J''. Thus, Algorithm 5.7 is correct.

It remains to show that Algorithm 5.7 runs in polynomial time. Steps 2 and 4 run in polynomial time by Proposition 5.2. Identifying the blocks of J' (step 3) amounts to compute the connected components of the connection graph $G(J')$ of J' and can be done in polynomial time (see the proof of Proposition 5.6).

By Proposition 5.6, for showing that the core computation of steps 5 and 6 requires polynomial time only, it is sufficient to prove that the blocks of J' are of bounded size. Towards this goal we state the following key observation whose proof is available in [16].

Fact A. There is a constant c such that for each block B of J', either $|J'[B]| \leq c$ or there is a block B_0 of J and an instance K obtained by chasing Σ_t on $J[B_0]$ such that $J'[B] \subseteq K$.

From Fact A and Proposition 5.2 it follows that each $|dom(J'[B])|$ is bounded by $q(|dom(J[B_0])|)$ for some block B_0 of J, where q is a polynomial, and thus by a constant, since I is ground and thus $|dom(J[B_0])|$ is obviously bounded by a constant depending on Σ_{st} only. Since B consists of the variables in $dom(J'[B])$, the size of the blocks of J' is bounded by a constant. $\square$

Corollary 5.9. *Computing the core of the universal solutions of data exchange problems whose target constraints consist of weakly acyclic inclusion dependencies and arbitrary equality generating dependencies can be done in polynomial time by Algorithm 5.7.*

Let us now sketch how the concept of bounded hypertree width can be exploited to significantly improve Algorithm 5.7 and to obtain a very low polynomial upper bound for the time complexity of core computation if Σ_t contains weakly acyclic simple TGDs only (more general settings will be explored in the future). Denote by $mr(\Sigma)$ the maximum number of atoms that occur in a rhs of a TGD of a set Σ of TGDs. The next theorem states that the hypertree width is (basically) preserved when chasing a weakly acyclic set of simple TGDs over an instance.

Theorem 5.10. *Let J be an instance and let Σ be a set of weakly acyclic simple TGDs. Let K be obtained by chasing Σ over J. Then $hw(K) \leq \max\{hw(J), mr(\Sigma)\}$.*

Proof. (Rough Sketch.) Assume we have a hypertree decomposition T of width k of Σ. We construct on the fly a hypertree decomposition T' for K as follows. Whenever a simple TGD of the form $A_0 \rightarrow A_1 \wedge A_2 \wedge \ldots \wedge A_r$ fires with a left side atom A such that $A = A_0\theta$ for a suitable substitution θ, then the generated new atoms among $A_1\theta, A_2\theta, \ldots, A_r\theta$ are put together into a new hypernode which is attached as a child to an already existing hypernode which contains A (or at least covers all variables of A). It is not hard to see that the resulting decomposition is effectively a hypertree decomposition of the desired width. $\square$

As shown by the next theorem, the above result is extremely valuable for efficiently computing cores.

Theorem 5.11. *Computing the core of a universal solution J' of data exchange problems whose target constraints σ_t consist of weakly acyclic simple TGDs can be done in time $O(|J'| \times \|J'\|^{k+1})$, where $k = \max\{mr(\Sigma_{st}), mr(\Sigma_t)\}$.*

Proof. (Sketch.) If J is obtained (as in Algorithm 5.7) by chasing a set Σ_{st} over a ground input instance I, then it is trivial to see that $hw(J) \leq mr(\Sigma_{st})$. When $\Sigma_t = \Sigma_T$ is chased over J, yielding J', then, by Theorem 5.10, $hw(J') \leq k = \max\{mr(\Sigma_{st}), mr(\Sigma_t)\}$. As explained in the proof of Theorem 5.10, the hypertree decomposition of J' can be generated on the fly at no significant cost. The number of nodes in this decomposition is bounded by $|J'|$. By Theorem 3.9 we thus get the desired time bound of time $O(|J'| \times \|J'\|^{k+1})$. $\square$

6. ANOTHER TRACTABLE CLASS: FULL TGDS AND EGDS AS TARGET CONSTRAINTS

In this section we show that computing the core of data exchange problems whose target constraints consist of full TGDs and arbitrary EGDs is tractable. This answers an open question of [14]. We start by deriving some useful algebraic lemmas, then analyze full TGDs alone, and finally add EGDs.

6.1 Some algebraic lemmas

Definition 6.1. *Let K and M be two instances with $K \subseteq M$. We say that K is* a nucleus *of M if each legal mapping $f : dom(K) \longrightarrow dom(M)$, for which $f(K) \subseteq M$, extends to an endomorphism $f^* \in end(M)$ such that for each $x \in dom(K)$, $f^*(x) = f(x)$ and for each $x \in var(M) - var(K)$, $f^*(x) = x$.*

Definition 6.2. *Let K and M be two instances with $K \subseteq M$. K* controls *M if K is a nucleus of M and $var(K) = var(M)$.*

Note that *control* as just defined is a very strong relationship between instances. In fact, if K controls M, then there is a one-to-one correspondence between the endomorphisms of M and the elements of $legal(K, M)$. Thus, in particular, each $h \in end(M)$ is fully identified by its restriction to K.

Recall that Chasing a set Σ of full TGDs on an instance K always produces a unique result and that we denote this result by $CHASE_\Sigma(K)$.

Lemma 6.3. *Let K, M be instances such that $K \subseteq M$. If $h : dom(K) \longrightarrow dom(M)$ is a legal mapping, then $h(CHASE_\Sigma(K)) \subseteq CHASE_\Sigma(h(K))$.*

Proof. Consider any particular sequence of chasing the TGDs of Σ on K. Since the chase procedure is finite, there is a number m such that the procedure stops after m firings of TGDs. (Here, by a "firing", we understand a single application of a TGDs which generates one or more new facts). Let $K_0 = K$ and let, for $1 \leq i \leq m$, K_i be the cumulative set of all facts obtained via the chase after the i-th firing of some TGD (we thus have $K_0 \subset K_1 \subset K_2 \cdots \subset K_m$). It suffices to show by induction on i that for $1 \leq i \leq m$ if $A \in K_i$, then $h(A) \in CHASE_\Sigma(h(K))$. For the base case this holds, since obviously $h(K_0) = h(K) \subseteq CHASE_\Sigma(h(K))$. Now assume the statement hols for $j < m$. We show that it also holds for $j + 1$. Let $A \in K_{j+1}$. Then there is a TGD $d : L \rightarrow R$ and a substitution θ such that $\theta(L) \subseteq K_j$ and $A \in \theta(R)$. By the induction hypothesis, it follows that $h(\theta(L)) \in CHASE_\Sigma(h(K))$, and thus d must also fire with the substitution $h \circ \theta$ and produce from facts in $CHASE_\Sigma(h(K))$ a fact $h \circ \theta(R) = h(\theta(R)) = h(A)$. Thus $h(A) \in CHASE_\Sigma(h(K))$. $\square$

Lemma 6.4. *Let K be an instance and let M be the chase of K w.r.t. a set Σ of full TGDs. Then K controls M.*

Proof. Since $var(K) = var(M)$, it suffices to show that K is a nucleus of M, which, given that $var(K) = var(M)$ means that each legal mapping $h : dom(K) \longrightarrow dom(M)$ where $h(K) \subseteq M$ is contained in $end(M)$, i.e., $h(M) \subseteq M$. Since $M = CHASE_\Sigma(K)$ we have $h(M) = h(CHASE_\Sigma(K))$. From Lemma 6.3, we thus have $h(M) \subseteq CHASE_\Sigma(h(K))$. But by the monotonicity of $CHASE_\Sigma$, since $h(K) \subseteq M$, we have $CHASE_\Sigma(h(K)) \subseteq CHASE_\Sigma(M) = M$. Thus, by putting the last two inclusions together we obtain $h(M) \subseteq M$. □

Definition 6.5. *An endomorphism $h \in end(M)$ is* idempotent *if $h^2 = h$, i.e., if for each $x \in h(M)$, $h(x) = x$.*

Idempotent endomorphisms enjoy a nice interchange property:

Lemma 6.6 (Interchange Lemma). *Let K, M be instances such that $K \subseteq M$ and $M = CHASE_\Sigma(K)$, where Σ is a set of full TGDs. If $h \in end(M)$ is idempotent, then $CHASE_\Sigma(h(K)) = h(CHASE_\Sigma(K)) = h(M)$.*

Proof. $h(M) = h(CHASE_\Sigma(K)) \subseteq CHASE_\Sigma(h(K))$ already follows from Lemma 6.3. Let us thus show that $CHASE_\Sigma(h(K)) \subseteq h(CHASE_\Sigma(K))$. Since full TGDs do not introduce new variables, each atom A in $CHASE_\Sigma(h(K))$ contains variables from $var(h(K))$ only. Since h is idempotent, these variables are all invariant under h and thus $h(A) = A$. Hence $A \in h(M) = h(CHASE_\Sigma(K))$. □

We will use the following simple corollary.

Corollary 6.7. *Let M be an instance satisfying a set Σ of full TGDs, and let $h \in end(M)$ be idempotent. Then also $h(M)$ satisfies Σ, that is, $CHASE_\Sigma(h(M)) = h(M)$.*

Proof. Use Lemma 6.6 letting $K = M$. □

Note that Lemma 2.6 and Corollary 2.7 do not work if we drop the condition that h is idempotent. To see this, it is sufficient to give an example showing that Corollary 2.7 does not work in asence of idempotency.

Example 6.8. *Let $M = \{p(x,z), p(x,a), q(y,z), q(z,a), q(a,a), s(a)\}$, where x, y, z are variables and a is a constant. Let $\Sigma = \{\forall u \forall v \forall w(p(u,w) \wedge q(v,w) \wedge s(w) \rightarrow p(u,v))\}$, and let $h : var(M) \rightarrow dom(M)$ be the mapping defined by: $h(x) = x, h(y) = z$, and $h(z) = a$. Clearly, $h(A) \in M$ for each atom A of M, thus, $h \in end(M)$. Moreover, M is closed under Σ. Note that h is not idempotent, as, for example, $h(h(y)) = a$ whereas $h(y) = z$. We have: $h(M) = \{p(x,a), q(z,a), q(a,a), s(a)\}$. Now, $CHASE_\Sigma(h(M))$ contains $p(x,z)$ in addition to the facts from $h(M)$. Thus, $CHASE_\Sigma(h(M))$ is not contained in $h(M)$.*

Lemma 6.9. *There is an algorithm SMOOTHEN(M,h), which in time $O(|var(M)|^2)$ for each instance M and endomorphism $h \in end(M)$ computes an idempotent endomorphism $h' \in end(M)$ such that $h'(M) \subseteq h(M)$.*

Proof. We outline an algorithm SMOOTHEN(M,h) that computes the desired h' from h. If h is already idempotent, then SMOOTHEN(M,h) outputs h. Otherwise, observe that the directed deterministic function graph defined by h on $h(M)$ is such that after no more than $|var(h(M))|$ edges each element of $h(M)$ reaches a cycle or self-loop. The algorithm proceeds as follows: First, the SMOOTHEN algorithm computes $h' := h^v$ where $v = |var(h(M))|$. Observe that h' restricted to $var(h'(M))$, is an element of the symmetric group $S(dom(h'(M)))$, i.e., h' restricted to $dom(h'(M))$ is one-to-one on $dom(h'(M))$. It is well-known that each such permutation can be written as a set of permutation cycles. For example, the permutation $\{x_1 \rightarrow x_2, x_2 \rightarrow x_3, x_3 \rightarrow x_1, x_4 \rightarrow x_5, x_5 \rightarrow x_4, x_6 \rightarrow x_6, a \rightarrow a, b \rightarrow b\}$ has the five cycles $(x_1,x_2,x_3)(x_4,x_5)(x_6)(a)(b)$. Note that since h' is legal, only variables take part in cycles of length > 1 of h'. While there are still cycles of length > 1, the SMOOTHEN algorithm picks an arbitrary cycle C of length $c > 1$ and computes $h' := h'^c$. Obviously, at the end of this while loop, for each $x \in var(C), h'(x) = x$, i.e. h' is idempotent. The final h' thus leaves each element of $var(M)$ invariant and it clearly holds that $h'(M) \subseteq h(M)$. Thus the final h' is output as the desired endomorphism. In total, SMOOTHEN has performed no more than $2 \times var(M)$ homomorphism compositions. Each composition is feasible in time linear in $var(M)$, hence the total runtime is $O(|var(M)|^2)$. □

6.2 Full TGDs alone

Definition 6.10. *Let M be an instance and $h \in end(M)$. An* improvement *of h is an endomorphism $g \in end(M)$ such that $g(M) \subset h(M)$.*

The notion of improvement introduced here is related to the one of *useful endomorphism* introduced in [14] (see also Def. 5.3). In particular, h has an improvement iff there is a useful endomorphism for $h(M)$. As usual, if h is a mapping (in particular, an endomorphism) we denote by h^{-1} its set-valued inverse, i.e., the mapping that associates to each value x the set $h^{-1}(x) = \{z | h(z) = x\}$.

Definition 6.11. *Let K, M be instances such that K controls M and let $h \in end(M)$. An improvement g of h is called (K,h)-*quasilocal *if there exists a variable x_0 such that:*

- *for each block $B \in blocks(h^{-1}(x_0), K)$, $g(B) \subseteq h(M)$ and $x_0 \notin var(g(B))$; and*
- *for each block $B \notin blocks(h^{-1}(x_0), K)$, for each $x \in B$ $g(x) = h(x)$.*

For an instance M, let us denote by $CORE(M)$ the set of all (isomorphic) cores of M.

Lemma 6.12. *Let Σ is a set of full TGDs, let K, M be instances such that $M = CHASE_\Sigma(K)$, and let h be an idempotent endomorphism of M. If $h(M) \notin CORE(M)$, then there exists a (K,h)-quasilocal improvement of h.*

Proof. Clearly, $K \subseteq M$. Moreover, by Lemma 6.4, K controls M. Since $h(M) \notin CORE(M)$, there exists an improvement f of h and an element $x_0 \in var(M)$ such that $x_0 \in var(h(M)) - var(f(M))$. Clearly for each block $B \in blocks(h^{-1}(x_0), K)$, $f(B) \subseteq h(M)$ and $x_0 \notin var(f(B))$. Let g be the endomorphism of M defined as follows:

- for each $x \in var(blocks(h^{-1}(x_0), K))$, $g(x) = f(x)$; and
- for each $x \in var(M) - var(blocks(h^{-1}(x_0), K))$, $g(x) = h(x)$.

Since K controls M, g is effectively an endomorphism of M. By Lemma 6.3, we have:

(1) $g(M) = g(CHASE_\Sigma(K)) \subseteq CHASE_\Sigma(g(K))$.

Since $g(K) \subseteq h(M)$, by the monotocicity of the Chase procedure for full TGDs, it follows that

$$(2) \quad CHASE_\Sigma(g(K)) \subseteq CHASE_\Sigma(h(M)).$$

Because h is idempotent, by Corollary 6.7

$$(3) \quad CHASE_\Sigma(h(M)) = h(M).$$

Putting (1), (2), and (3) together, we obtain $g(M) \subseteq h(M)$. Since x_0 does not occur in the image of g, this inclusion is proper, i.e., $g(M) \subset h(M)$. Thus g is a (K, h)-quasilocal improvement of h. □

Assume $M = CHASE_\Sigma(K)$ and $h \in end(M)$ is idempotent. The following algorithm IMPROVE(M, K, h) outputs a (K, h)-quasilocal improvement of h if one exists, i.e., if $h(M) \notin CORE(M)$ and outputs *fail* if $h(M) \in core(M)$.

```
ALGORITHM IMPROVE($M, K, h$);
INPUT: Instances $M$, $K \subseteq M$, s.t. $M = CHASE_\Sigma(K)$
          for some set $\Sigma$ of full TGDs; and
          an idempotent endomorphism $h \in end(M)$.
OUTPUT: A $(K, h)$-quasilocal improvement $g$ of $h$ if
          one exists, otherwise fail.

BEGIN
$found := false$;
$vars := var(h(M))$;
REPEAT
pick a variable $x_0 \in vars$;
$goodx = x_0$;
$vars := vars - \{x_0\}$;
   BEGIN
   compute $blocks := blocks(h^{-1}(x_0), K)$;
   $matches := true$;
   Let $\theta = \{\}$ be the empty substitution;
   REPEAT
      pick a block $B$ from $blocks$;
      $blocks := blocks - \{B\}$;
      search for a substitution
       $\lambda : var(B) \longrightarrow (dom(M) - \{x_0\})$
         such that $\lambda(B) \subseteq h(M)$;
      IF such a $\lambda$ can be found
               THEN $\theta := \theta \cup \lambda$
               ELSE $matches := false$;
   UNTIL not $matches$ OR $blocks = \emptyset$.
   IF $matches$ THEN $found := true$;
UNTIL $found$ OR $vars = \emptyset$;
IF not $found$ THEN OUTPUT($fail$)
ELSE BEGIN
      Let $g$ be defined by:
         $g(x) = \theta(x)$ if $x \in var(blocks(h^{-1}(goodx), K))$,
         and $g(x) = h(x)$ otherwise;
      OUTPUT($g$)
      END
END.
```

The algorithm systematically explores whether some $x_0 \in var(h(M))$ gives rise to a quasi-local improvement and if so, outputs the first such improvement. The following lemma, whose proof is available in [16], states its correctness and a bound on its complexity.

LEMMA 6.13. *Algorithm IMPROVE on input (M, K, h) outputs an improvement of h if $h(M) \notin CORE(M)$ and "fail" otherwise. The algorithm runs in time $O(|var(M)|^2 \times |M|^{blockwidth(K)})$.*

We are now ready for stating our new algorithm FASTCORE that computes in polynomial time the core of a data exchange problem $< \sigma, I >$, for some fixed data exchange setting $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where Σ_t consiste of full TGDs only, and a source instance I.

```
ALGORITHM FASTCORE
INPUT: Source instance $I$ to $\sigma$.
OUTPUT: The core of $\sigma$, i.e., of a canonical universal
   solution $J'$ of $< \sigma, I >$.
BEGIN
   Compute a canonical solution $J$ for $< \sigma, I >$;
   $J := CHASE_{\Sigma_{st}}(I)$;
   $J' := CHASE_{\Sigma_t}(J)$;
   $g :=$ identity on $var(J') = var(J)$;
   REPEAT
      h:=g;
      g:= IMPROVE($J', J, g$)
      IF $g \neq fail$ THEN $g :=$ SMOOTHEN($J', g$);
   UNTIL $g = fail$;
   OUTPUT($h(J')$);
END.
```

The FASTCORE algorithm successively performs quasilocal improvements until no further such improvement is possible and thus the core has been attained. In the extended version [16] of this paper we prove:

THEOREM 6.14. *The algorithm FASTCORE is correct and runs in polynomial time.*

6.3 Adding EGDs

In order to extend the above results to full TGDs plus EGDs, we use a new trick, which consists in simulating the effect of EGDs by special full TGDs and by exploiting implicit endomorphisms for the core computation. Let us illustrate this by an example.

Assume the target signature $\mathbf{T}$ contains two unary relations P and Q and a binary relation R and no other relation. Assume Σ_t contains an EGD $e : \forall x, y, z\, (R(x, x) \wedge R(y, x) \rightarrow x = y)$. Then we can simulate this EGD through the following eight full TGDs:

$$\begin{array}{l}
\forall x, y(R(x, x) \wedge R(y, x) \wedge P(x) \rightarrow P(y));\\
\forall x, y(R(x, x) \wedge R(y, x) \wedge P(y) \rightarrow P(x));\\
\forall x, y(R(x, x) \wedge R(y, x) \wedge Q(x) \rightarrow Q(y));\\
\forall x, y(R(x, x) \wedge R(y, x) \wedge Q(y) \rightarrow Q(x));\\
\forall x, y, u(R(x, x) \wedge R(y, x) \wedge R(x, u) \rightarrow R(y, u));\\
\forall x, y, u(R(x, x) \wedge R(y, x) \wedge R(y, u) \rightarrow R(x, u));\\
\forall x, y, u(R(x, x) \wedge R(y, x) \wedge R(u, x) \rightarrow R(u, y));\\
\forall x, y, u(R(x, x) \wedge R(y, x) \wedge R(u, y) \rightarrow R(u, x)).
\end{array}$$

In summary, for the EGD e, and for each target predicate Π, and argument position i, we create a new TGD which specifies under the premise that the EGD e applies, that, whenever x occurs at position i in a Π-atom, then another Π atom should be generated by substituting y for x at this position. Moreover, we create another TGD that performs the inverse substitution. Note that, since there is a constant number of EGDs in Σ_t and a constant number of predicate symbols in the target schema $\mathbf{T}$, each of which has a constant number of argument positions, the overall number of new TGDs created is thus constant.

Obviously, whenever, through the enforcement of an equality, a new fact is generated via the EGD e, then the same fact is also generated via the above TGDs. In this sense, the above full TGDs *simulate* the EGD e. But note that some additional facts will be generated by the TGDs. For example, assume we want to apply e on the set of facts $\{R(z,z), R(a,z), P(z), P(z')\}$. Thus, we infer $z = a$ by e and have to enforce the substitution $z \leftarrow a$. As a result we obtain the set $S = \{R(a,a), P(a), P(z')\}$. If, instead, we chase the above TGDs, on the same set of facts $\{R(z,z), R(a,z), P(z), P(z')\}$, we obtain the larger instance $S' = \{R(a,a), R(z,a), R(a,z), R(z,z), P(a), P(z), P(z')\}$. However, the smaller set S is an endomorphic image of larger set S' under the endomorphism h defined by $h(z) = a$ and $h(z') = z'$. Note that this endomorphism is defined precisely by the substitution obtained by enforcing the EGD e. So, the core $\{R(a,a), P(a)\}$ of the smaller instance S is equal to the core of the larger instance S'. Thus, in order to compute (an isomorphic image of) the core of the smaller set, it suffices to compute the core of the larger (full-TGD-generated) set. This can be done using the polynomial algorithm FAST-CORE presented in the last sesction.

We have shown for a simple example that for computing the core it is sufficient to replace an EGD by a set of full TGDs. That this works generally is proven in the extended version of the paper [16]. We thus have:

THEOREM 6.15. *Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, be a data exchange setting, where Σ_t is allowed to contain both EGDs and full TGDs. Given an input instance I for σ, computing the core of a universal solution for I is feasible in polynomial time.*

7. INTRACTABILITY IN PRESENCE OF THE NULL PREDICATE

In many query languages and/or database management systems, a builtin predicate of the syntactic form $NULL(x)$ or similar is available, which distinguishes null values from definit values (i.e., non-nulls). For example, in SQL, the logical expression "*x IS NULL*" evaluates to true iff x is a null value. Such null-checks can also be used within integrity constraints. Note also that such a null-check is usually conceived as a builtin predicate which is evaluated *ad hoc* rather than being part of a relational signature. In fact, if a null value X is equated with a constant a, say, by an FD enforcement, of course, $NULL(a)$ does not become true. Thus, the $NULL$ predicate is not subject to variable-replacements or homomorphisms. In the present paper, we deal with database instances with *labeled nulls* and not with standard nulls. Nevertheless it is interesting to see what happens in case we allow a $NULL$ predicate to occur in the target itegrity constraints.

Let us thus assume that a monadic $NULL$ predicate as described is available. That is, during the evaluation of a rule body, $NULL(x)$ evaluates to true if x is matches a null value, and false if x matches a constant. We show that the core computation problem becomes intractable, even if the set of target constraints consists of a single full TGD. This contrasts with our result that the core computation problem is tractable for arbitrary TGDs (plus EGDs). Thus, the non-ability of distinguishing between nulls and constants in the integrity constraints of data exchange settings as defined by Fagin et al. in [13, 14] turns out to be an essential requirement for the tractability of core computation (at least in the case of full TGDs).

THEOREM 7.1. *If the NULL predicate is available, computing the core of data exchange problems where Σ_{st} consists of TGDs and Σ_t consists of full TGDs only is NP-hard. The problem remains NP-hard even if Σ_t is acyclic and consists of a single full TGD.*

PROOF. Let $\sigma = \langle \mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t \rangle$, where

- $\mathbf{S} = \langle\ v(.)\ , e(.,.),\ red(.),\ green(.)\ , blue(.)\ \rangle$
- $\mathbf{T} = \langle\ w(.,.),\ e'(.,.),\ g(.,.) \rangle$
- Σ_{st} contains the dependencies

ST1: $\forall x\, (v(x) \rightarrow \exists y\, w(x,y))$.
ST2: $\forall x \forall y\ (e(x,y) \rightarrow e'(x,y))$.
ST3: $\forall x \forall y\ (red(x) \wedge green(y) \rightarrow g(x,y))$.
ST4: $\forall x \forall y\ (red(x) \wedge green(y) \rightarrow g(y,x))$.
ST5: $\forall x \forall y\ (red(x) \wedge blue(y) \rightarrow g(x,y))$.
ST6: $\forall x \forall y\ (red(x) \wedge blue(y) \rightarrow g(y,x))$.
ST7: $\forall x \forall y\ (green(x) \wedge blue(y) \rightarrow g(x,y))$.
ST8: $\forall x \forall y\ (green(x) \wedge blue(y) \rightarrow g(y,x))$.
ST9: $\forall x \forall y\ (\ v(x) \wedge red(y) \rightarrow w(x,y))$.
ST10: $\forall x \forall y\ (\ v(x) \wedge green(y) \rightarrow w(x,y))$.
ST11: $\forall x \forall y\ (\ v(x) \wedge blue(y) \rightarrow w(x,y))$.

- Σ_t contains the single dependency

T1: $\forall x \forall y \forall x' \forall y'\ (\ (w(x,x') \wedge w(y,y') \wedge e'(x,y)) \wedge NULL(x') \wedge NULL(y') \rightarrow g(x',y'))$.

We now transform the graph three-colorability problem 3COL in polynomial time into a source instance I for the above data exchange setting σ.

Let $G = (V, E)$ be an instance of 3COL. Then the vocabulary of I consists of the set $\underline{\text{Const}} := V \cup \{c_r, c_g, c_b\}$, where c_r, c_g, c_b are constant symbols not occurring in V, which intuitively denote the colors red, green, and blue. The tuples in the relation instances of I according to the source schema $\mathbf{S}$ are obtained as follows:

- For each vertex $a \in V$, add a tuple $v(a)$.
- For each edge $\{a,b\} \in E$, add the tuples $e(a,b)$ and $e(b,a)$[4].
- Add tuples $red(c_r)$, $green(c_g)$, and $blue(c_b)$.

The source instance I is thus well-defined. Note that Σ_t is acyclic and therefore also weakly acyclic. It follows that a universal solution can be computed by a chase procedure in polynomial time [13]. Let J denote the universal solution produced by this chase. Let $CV = \{w(a,x_a) | a \in V\}$ where for each a, x_a is the null created by ST1. Let $Rep(G) = \{g(x_a,x_b), g(x_b,x_a)|\ \{a,b\} \in E\}$. It can be seen (and is formally shown in [16]) that $\underline{\text{Core}}(J) = J - (CV \cup Rep(G))$ iff G is 3-colorable. $\square$

The data exchange problem constructed in the above proof shows that even a single full TGD can have a detrimental effect on the block size. This remains true even if without use of the NULL predicate (e.g. if we drop the NULL atoms from rule T1). This means that in presence of TGDs with more than one atom in the left side, a universal solution to a data exchange problem can have unbounded blockwith, even

[4]It would be sufficient to add only one of these two tuples to I, but we found it conceptually simpler to add both tuples, thus explicitly representing an undirected graph.

if no EGDs are applied. Thus, neither the metods for polynomial core computation described in [14] nor our methods for simple TGDs will work, because both methods require a bounded blockwidth after the application of the TGDs. This may be seen as a justification of the more complex proof of the tractability of core computationin the case of TGDs.

Acknowledgments This work was supported in equal parts by the Austrian Science Fund (FWF) project: *Nr. P17222-N04 Complementary Approaches to Constraint Satisfaction* and the EU IST project *2001-33570 INFOMIX: Knowledge-Based Information Integration* The author thanks Fang Wei, Ron Fagin, Gianluigi Greco, Chris Koch, and Steffi Scherzinger for their comments on earlier versions of different parts of this paper.

8. REFERENCES

[1] S. Abiteboul, R. Hull, V. Vianu. *Foundations of Databases*, Addison-Wesley Publishing Company, 1995.

[2] A. Aho, J. Hopcroft, and J. Ullman. Data Structures and Algorithms. Addison-Wesley, Reading MA, 1983.

[3] A. Aho, J. Sagiv, and J. Ullman. Equivalence of Relational Expressions. *SIAM J. Comput.* 8:2, 1979.

[4] A. Aho, J. Sagiv, and J. Ullman. Efficient Optimization of a Class of Relational Expressions. *TODS* 4:4,1979.

[5] M. Arenas, P. Barceló, R. Fagin, and L. Libkin, Locally Consistent Transformations and Query Answering in Data Exchange. *Proc. ACM PODS 2004*, pp.229-240.

[6] P. Atzeni and E.P.F. Chan. Independent Database Schemes under Functional and Inclusion Dependencies. *Acta Informatica* 28:8, pp. 777-779, 1991.

[7] C. Beeri and M. Vardi. A Proof Procedure for Data Dependencies. *J.ACM.* 31:4, PP. 718-741,1984.

[8] M.A. Casanova, R. Fagin, and C.H. Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *JCSS 28:1, 1984.*

[9] A.K. Chandra and P.M. Merlin. Optimal Implementation of Conjunctive Queries in Relational Databases. *STOC 1977.*

[10] C. Chekuri and A. Rajaraman. Conjunctive Query Containment Revisited. *TCS 239:2, 2000.*

[11] R.G. Downey and M.R. Fellows. *Parameterized Complexity.* Springer, 1999.

[12] R.G. Downey and M.R. Fellows and U.Tailor. The parameterized Complexity of Relational Database Queries and an Improved Characterization of $W[1]$. In D.S. Bridges et al., editors, *Combinatorics, Complexity, and Logic – Proc. DMTCS'96.*

[13] R. Fagin, Ph. Kolaitis, R. Miller, and L. Popa, Data Exchange: Semantics and Query Answering, ICDT'03, pp. 207–224. Full version to appear in *TCS.*

[14] R. Fagin, Ph. Kolaitis, and L. Popa, Data Exchange: Getting to the Core. In *Proc. PODS'03*, pp. 90–101, 2003. Full version to appear in *ACM TODS.*.

[15] J. Flum, M. Frick, and M. Grohe. Query Evaluation via Tree-Decompositions. *J.ACM* 49:6, 2002.

[16] G. Gottlob. Computing Cores for Data Exchange: New Algorithms and Practical Solutions. Extended version of the present paper. Currently available at: www.dbai.tuwien.ac.at/staff/gottlob/extcore.pdf

[17] G. Gottlob and A. Leitsch. On the efficiency of Subsumption Algorithms. *J.ACM* 32:2, 1985.

[18] G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decompositions and Tractable queries. *Journal of Computer and System Sciences* 64:3, pp. 579-627, 2002.

[19] G. Gottlob, N. Leone, and F. Scarcello. The Complexity of Acyclic Conjunctive Queries. *J.ACM* 48:3, pp. 431-498, 2001.

[20] G. Gottlob, N. Leone, and F. Scarcello. Robbers, Marshals, and Guards: Game Theoretic and Logical Characterizations of Hypertree Width. *Journal of Computer and System Sciences* 66(4): 775-808, 2003.

[21] G. Gottlob, N. Leone, and F. Scarcello. A Comparison of Structural CSP Decomposition Methods. *Artificial Intelligence* 124:2, pp. 243–282, 2000.

[22] G. Gottlob, R. Pichler. Hypergraphs in Model Checking: Acyclicity in Hypertree-Width versus Clique-Width. *SIAM J.Comput.*, 33:2,2004.

[23] C. Gutiérrez, C. Hurtado, and A. Mendelzon. Foundations of Semantic Web Databases. *ACM PODS 2004*, pp. 95-106, 2004.

[24] M. Grohe. Parameterized Complexity for the Database Theorist. *ACM SIGMOD Record*, 31:4, December 2002, pp.86-96,2002.

[25] T. Imielinski. Abstraction in Query Processing. *Journal ofthe ACM* 38:3, pp. 534–558, 1991.

[26] Ph. Kolaitis. Data Exchange: Schema Mappings, Data Exchange, and Metadata Management.*PODS'05.*

[27] Ph.G. Kolaitis, M.Y. Vardi. Conjunctive-Query Containment and Constraint Satisfaction. *Journal of Computer and System Sciences* 61:2, pp. 302-332, 2000.

[28] M. Levene, G. Loizou. How to Prevent Interaction of Functional and Inclusion Dependencies. *Information Processing Letters* 71:3-4, pp.115-125, 1999.

[29] M. Levene, G. Loizou. Guaranteeing no interaction between functional dependencies and tree-like inclusion dependencies. *TCS* 254:1-2, 2001.

[30] A.Y. Levy, A. O. Mendelzon, Y. Sagiv, D. Srivastava. Answering Queries Using Views. *PODS 1995.* . pp.95-104, 1995.

[31] H. Mannila and K.-J. Räihä. *The Design of Relational Datases.* Addison Wesley 1992.

[32] D. Maier, A. Mendelzon, and Y. Sagiv. Testing Implication of Data Dependencies, *ACM TODS* 4:4, pp. 455-469, 1979.

[33] R. Miller, L. Haas, and M. Hernández. Schema Mapping as Query Discovery.*VLDB 2000.*

[34] L. Popa, Y. Velegrakis, R. Miller, M. Hernández, and R. Fagin. Translating Web Data. *VLDB 2002.*

[35] C.H. Papadimitriou and M. Yannakakis. On the Complexity of Database Queries. *Journal of Computers and System Sciences*, 58:407–427, 1999.

[36] X. Qian. Query Folding. *ICDE 1996.*

[37] N. Robertson and P.D. Seymour. Graph Minors II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7:309-322, 1986.

[38] F. Scarcello, G. Greco, and N. Leone. Weighted Hypertree Decompositions and Optimal Query Plans. *PODS 2004.*

[39] J.D. Ullman. *Principles of Database and Knowledge Base Systems*, Computer Science Press, Rockville, MD, vol. I, 1988, and vol. II, 1989.

[40] M. Yannakakis. Algorithms for Acyclic Database Schemes. *VLDB 1981, pp. 82-94.*