Algorithms for Core Computation in Data Exchange [*]

Vadim Savenkov

Vienna University of Technology
Favoritenstraße 9, Vienna, Austria
savenkov@dbai.tuwien.ac.at

Abstract

We describe the state of the art in the area of core computation for data exchange. Two main approaches are considered: post-processing core computation, applied to a canonical universal solution constructed by chasing a given schema mapping, and direct core computation, where the mapping is first rewritten in order to create core universal solutions by chasing it.

1998 ACM Subject Classification H.2.5 [Heterogeneous Databases]: Data translation

Keywords and phrases data integration, data exchange, core computation

Digital Object Identifier 10.4230/DFU.Vol5.10452.39

1 Introduction

Data exchange is concerned with the transfer of data between databases with different schemas, according to declarative specifications known as schema mappings. Unlike virtual data integration, concerned with query translation among distributed databases [15, 9], data exchange aims at actually materializing a target database, for the later use offline.

In this chapter, we consider the most common schema mapping language, based on *tuple-generating dependencies* (tgds) and *equality-generating dependencies* (egds). Our setting assumes two parties: the source and the target data storages with respective relational schemas **S** and **T**, and the single direction of data flow. Such scenario is typically guided by the source-to-target tgds (*st-tgds* for short) and target constraints based on egds and tgds.

The *data exchange problem* for a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a set of st-tgds and target constraints, is defined by Fagin, Kolaitis, Miller and Popa in [6] as a task of constructing a target instance J for a given source instance I, s.t. the combined instance $\langle I, J\rangle$ satisfies the dependencies in Σ. Such J is called a *solution for* I to the data exchange problem associated with $\mathcal{M}$.

▶ **Example 1** ([20]). Let Tutorial(course, tutor) and BasicUnit(course) be relations in a source schema, and NeedsLab(id_tutor,lab), Tutor(id_tutor,tutor), Teaches(id_tutor, id_course) and Course(id_course,course) be relations in a target schema. The following source-to-target tgds relate the two schemas:

1. $\forall C\,(\mathsf{BasicUnit}(C) \rightarrow \exists Idc\,\mathsf{Course}(Idc, C))$,
2. $\forall C \forall T\,(\mathsf{Tutorial}(C, T) \rightarrow \exists Idc\,\exists Idt\,(\mathsf{Course}(Idc, C) \wedge \mathsf{Tutor}(Idt, T) \wedge \mathsf{Teaches}(Idt, Idc)))$.

Target dependencies are given by the two tgds:

3. $\forall Idc \forall C\,(\mathsf{Course}(Idc, C) \rightarrow \exists Idt\,\exists T\,(\mathsf{Tutor}(Idt, T) \wedge \mathsf{Teaches}(Idt, Idc)))$,
4. $\forall Idt \forall Idc\,(\mathsf{Teaches}(Idt, Idc) \rightarrow \exists L\,\mathsf{NeedsLab}(Idt, L))$.

* The author was partly supported by the Vienna Science and Technology Fund (WWTF) through projects ICT08-032 and ICT12-15, by the Austrian Science Fund (FWF): P25207-N23, and by the Erasmus Mundus External Co-operation Window Programme of the European Union.

For the source instance I consisting of two facts Tutorial('java', 'Yves') and BasicUnit('java'), the following instances are all valid solutions:

$J = \{$Course(C_1, 'java'), Tutor(T_2,N), Teaches(T_2,C_1), NeedsLab(T_2,L_2),
Course(C_2, 'java'), Tutor(T_1,'Yves'), Teaches(T_1,C_2), NeedsLab(T_1,L_1)$\}$,

$J_c = \{$Course(C_1,'java'), Tutor(T_1,'Yves'), Teaches(T_1,C_1), NeedsLab(T_1,L_1)$\}$,

$J' = \{$Course('java','java'),Tutor(T_1,'Yves'), Teaches(T_1,'java'), NeedsLab(T_1,L_1)$\}$

Existentially quantified variables in tgds can be interpreted as arbitrary values. To reflect this, solutions in data exchange may contain *labeled nulls*, serving as placeholders for unknown constants. Labeled nulls are denoted by capitalized identifiers without quotes in this chapter. ◀

As demonstrated by Example 1, data exchange problems may admit multiple solutions, due to the use of implicational dependencies with existentially quantified variables in schema mappings. Fagin et al. in [6] proposed clear criteria for evaluating the quality of solutions. The most prominent requirement is *universality*, disallowing materialization of facts not implied by $I \cup \Sigma$, where I is seen as conjunction of atoms of the source instance, and Σ is the set of dependencies in the mapping. This requirement can be captured as follows: a solution K for I is *universal*, if for arbitrary solution K' for I, there is a function h mapping labeled nulls of K to values occurring in K' and preserving non-null values of K, such that $h(K) \subseteq K'$ holds. Such h is called a *homomorphism.* Note that J' in Example 1 is not universal, since there exists no homomorphism transforming J' into the solution J. Indeed, a homomorphism preserves constants, and thus the fact Course('java','java') cannot be mapped onto any fact in J. At the same time, J is a universal solution (and hence, so is J_c, which is a subset of J, up to a renaming of labeled nulls): in particular, J can be transformed into J' by mapping C_1 and C_2 to 'java', T_2 to T_1, L_2 to L_1 and N to 'Yves'.

The number of universal solutions is usually infinite: unless very restrictive target egds are part of the mapping, arbitrary number of facts consisting of fresh distinct labeled nulls can be added to a universal solution J, without affecting its universality. Fagin, Kolaitis and Popa [7] thus recognized the *size* of universal solution as an important quality criterion, and proposed the notion of *core universal solution.* It is inspired by the graph theoretic concept of the *core of a graph* [13] defined as smallest subgraph which also is a homomorphic image of the graph. Since universal solutions have homomorphisms to any other solutions, core universal solution can be defined as *the smallest universal solution possible* (Thus, the smallest solution J_c in Example 1 is the core universal solution). The following holds [7]:

1. For each source instance I and a mapping $\mathcal{M}$, a smallest universal solution is unique up to isomorphism (that is, up to renaming of labeled nulls). Therefore, we can speak of *the core universal solution* (or, simply, *the core*).
2. Every universal solution contains the core universal solution as its subset.
3. All universal solutions for I under $\mathcal{M}$ have the same core.
4. For some classes of queries, *certain answers* (or the best approximations thereof) can be found by evaluating the queries on the core universal solution. Certain answers (cf. Chapter 5) are the answers that are found in every solution for a data exchange problem.

Being an attractive option for materialization, core universal solutions are not always easy to compute. For mappings with expressive target constraints, the question of feasibility of finding the core universal solution remained open for several years. In 2006 Gottlob and Nash answered it positively for mappings with target egds and tgds, appropriately restricted to ensure the termination of the data exchange process based on incrementally satisfying all dependencies in the mapping (known as *chasing* the dependencies, or just *the chase*) [12]. Their technique eliminates redundant facts from instances that result from the chase, and

Table 1 Development of Algorithms for Core Computation.

Algorithm	Year$^{1)}$	Type	Σ_t	Scale: 300s	Comments
GreedyCoreComp [7]	2003	PP	egds	n/a	
BlockCoreComp [7]	2003	PP	egds	n/a	
Hd-Core [10]	2005	PP	simple tgds + egds	n/a	hypertree-decomp. *not covered*
FastCore [10]	2005	PP	full tgds + egds	n/a	*not covered*
FindCore [12]	2006	PP	tgds + egds	2K	Chase with egds: [20] Skolemized mappings, oblivious chase: [16]
Core mappings [19]	2009	D	$\emptyset$	500K	
Laconic mappings [22]	2009	D	$\emptyset$	n/a	FO$^<$ st-tgds
Spicy-FD [18]	2010	D	FDs	1M	FDs (best-effort)

$^{1)}$ Conference versions of the articles:
[7]: In *Proceedings of PODS 2003*, pp. 90–101, ACM 2003
[12]: In *Proceedings of PODS 2006*, pp. 40–49, ACM 2006
[20]: In *Proceedings of LPAR 2008*, LNCS(5330), pp. 62–78, Springer
[19]: In *Proceedings of SIGMOD 2009*, pp. 655–668, ACM 2009

thus can be called a *post-processing core computation* method. Despite theoretical tractability, it has not yet been proven to scale in practice.

Much better performance is demonstrated by the method of *direct core computation* proposed by Mecca, Papotti and Raunich [19], and independently by ten Cate, Kolaitis, Chiticariu and Tan [22] in 2009. Its idea is to rewrite the dependencies in the mapping in such a way that chasing them immediately yields a core universal solution. The downside of this approach is that far less expressive mappings can be supported: both algorithms of [19, 22] deal with mappings without target constraints, whereas in [18], Marnette, Mecca and Papotti extend direct core computation to encompass target functional dependencies on the best-effort basis. Both [19, 18] report experimental results witnessing that direct core computation scales to instances with up to million records.

Table 1 contrasts the published algorithms for core computation in data exchange. The columns are (1) the name of the algorithm, (2) year of its first publication, (3) type: post-processing or direct, (4) the class of target constraints supported, (5) estimate of the source instance size (in tuples) for which the core universal solution can be found in 5 minutes, based on the latest published results, and (6) additional comments. The prototypical algorithms GreedyCoreComp and BlockCoreComp, proposed by Fagin et al. in their foundational paper [7], are discussed in Sections 3.1, 4.1 and 4.2. The algorithms HD-Core and FastCore by Gottlob were the first to encompass restricted classes of target tgds along with egds. The former supports the class of *simple tgds* having a single atom without repeated variables in the antecedent. This class leads to the target database instances with bounded hypertree-width (see Section 3.1 for brief discussion). The latter algorithm allows *full tgds* (introducing no new labeled nulls, see Section 2) and egds. Due to space restrictions, we will not discuss HD-Core and FastCore here, but istead focus on their successor, FindCore algorithm by Gottlob and Nash, supporting mappings with egds and *weakly-acyclic tgds*, a broad class of dependencies for which the chase always terminates. Marnette [16] has shown that FindCore can be lifted, in fact, to arbitrary terminating mappings based on tgds. In both [12, 16], egds are supported via encodings as tgds. Pichler and Savenkov [20] have shown how the need for such an encoding in FindCore can be eliminated, and provided a prototype implementation of post-processing core computation. FindCore algorithm and its enhancements is subject of Sections 4.3 and 4.4.

The last three lines in Table 1 are direct core computation methods. The Core schema mappings by Mecca et al. (Section 5.1.1) and Laconic schema mappings by ten Cate et al. (Section 5.1.2) were first such approaches, supporting source-to-target dependencies only. The algorithm Spicy-FD by Marnette et al. relies on these methods to provide a best-effort direct core computation facility in presence of target functional dependencies (Section 5.3).

The rest of this paper is organized as follows: after presenting the preliminaries in Section 2, we discuss general complexity of core computation Section 3, then present post-processing algorithms in Section 4, and direct core computation in Section 5. After outlining the performance of currently existing implementations in Section 6, we conclude with Section 7.

2 Preliminaries

Data exchange problem. A *schema* $\mathbf{R} = \{R_1, \dots, R_n\}$ is a set of relation symbols R_i each of a fixed arity. An *instance* over a schema $\mathbf{R}$ consists of a relation for each relation symbol in $\mathbf{R}$, s.t. both have the same arity. We only consider finite instances. We will usually identify a relation with its relation symbol (and vice versa).

Tuples of the relations may contain two types of elements: *constants* and *labeled nulls*. For every instance J, we write $\mathit{nulls}(J)$ to denote the set of labeled nulls of J and $\mathit{const}(J)$ to denote the set of constants of J. The two sets are disjoint: $\mathit{nulls}(J) \cap \mathit{const}(J) = \emptyset$. The *domain* of J $\mathit{dom}(J)$ is thus the union of $\mathit{nulls}(J)$ and $\mathit{const}(J)$. If a tuple $(x_1, x_2, \dots, x_n)$ belongs to the relation R, we say that J contains the *fact* $R(x_1, x_2, \dots, x_n)$. We also write $\vec{x}$ for a tuple $(x_1, x_2, \dots, x_n)$ and if $x_i \in X$, for every i, then we also write $\vec{x} \in X$ instead of $\vec{x} \in X^n$. Likewise, we write $r \in \vec{x}$ if $r = x_i$ for some i.

Let $\mathbf{S} = \{S_1, \dots, S_n\}$ and $\mathbf{T} = \{T_1, \dots, T_m\}$ 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 combined schema $\{S_1, \dots, S_n, T_1, \dots, T_m\}$. Instances over $\mathbf{S}$ (resp. $\mathbf{T}$) are called *source instances* (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 $\langle \mathbf{S}, \mathbf{T} \rangle$. A *subinstance* of an instance J is an instance over the same schema as J, containing a subset of facts of J.

Dependencies. A common class of database dependencies considered in the area of data exchange and data integration is the class of *embedded dependencies* [5]. These are first-order sentences $\forall \vec{x} \forall \vec{x}_0 \, (\phi(\vec{x}, \vec{x}_0) \to \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$, where *premise* ϕ and *conclusion* ψ are conjunctions of atomic formulas with relational symbols from some schema $\mathbf{R}$ or equalities. Throughout this paper, we shall omit the outermost universal quantifiers, and assume all variables occurring in the premise to be universally quantified (over the entire formula), and all variables occurring only in the conclusion to be existentially quantified over the entire conclusion. For instance, we shall write

$$S_1(x_1, x_2) \wedge S_2(x_1, x_3) \to \exists y_1 \exists y_2 \; Q(x_1, y_1) \wedge P(x_2, y_1, y_2)$$

instead of

$$\forall x_1 \forall x_2 \forall x_3 \; (\; S_1(x_1, x_2) \wedge S_2(x_1, x_3) \to \exists y_1 \exists y_2 \, (Q(x_1, y_1) \wedge P(x_2, y_1, y_2)) \;) \, .$$

The dependencies considered in this chapter fall in one of the two categories: *tuple-generating dependencies (tgds)* with conjunctions of atoms in the conclusions and *equality-generating dependencies (egds)* where conclusions are restricted to equality predicates. In Section 5, we will also extend the class of embedded dependencies by $\mathcal{L}$ tgds, whose antecedents $\phi(\vec{x}, \vec{x}_0)$ are formulas over the language $\mathcal{L}$. In particular, an important role will play FO tgds with antecedents being arbitrary first-order formulas, and FO$^<$ tgds enhancing the latter with the linear order relation $<$. Given a tgd $\tau\colon\ \phi(\vec{x}, \vec{x}_0) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})$

- the elements of $\vec{x}, \vec{x}_0$ are called the $\forall$-variables of τ, and the elements of $\vec{y}$ are called the $\exists$-variables of τ; it is assumed that all elements of $\vec{x}$ actually occur in $\psi(\vec{x}, \vec{y})$,
- $|\vec{x}|$ as $\forall$-*width* of τ and $|\vec{y}|$ $\exists$-*width* of τ. If $\exists$-*width* $= 0$, the tgd is called *full*.

For a mapping $\mathcal{M}$, $\forall$-*width* and $\exists$-*width* of $\mathcal{M}$ are defined, respectively, as the maximal $\forall$-*width* and $\exists$-*width* of a tgd in $\mathcal{M}$.

Schema mappings, data exchange problem. A *schema mapping* $\mathcal{M}$ is given by a triple $(\mathbf{S}, \mathbf{T}, \Sigma)$ consisting of the source schema $\mathbf{S}$, the target schema $\mathbf{T}$, the set of dependencies. Typically, Σ contains a set of *source-to-target dependencies* Σ_{st} and the set of *target dependencies* Σ_t. Each source-to-target dependency of Σ_{st} is a tgd with its antecedent over $\mathbf{S}$ and conclusion over $\mathbf{T}$. The target dependencies Σ_t range over $\mathbf{T}$.

The *data exchange problem* associated with a mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$ is the following: Given a null-free source instance I, find a target instance J, s.t. $\langle I, J\rangle \models \Sigma_{st}$ and $J \models \Sigma_t$. Such a J is called a *solution for I under $\mathcal{M}$* or, simply, a *solution* if I and $\mathcal{M}$ are clear from the context.

Skolemization. We will also consider *skolemized mappings*. The *standard skolemization* replaces each $\exists$-variable $y \in \vec{y}$ in the tgd $\phi(\vec{x}, \vec{x}_0) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})$ with a Skolem term $f(\vec{x})$ where f is a fresh distinct function symbol.

Chase. The data exchange problem can be solved using the *chase* procedure [1], which iteratively introduces new facts or equates terms until all desired dependencies are fulfilled.

Tgd chase step. Let $\phi(\vec{x}, \vec{x}_0) \rightarrow \exists \vec{y}\ \psi(\vec{x}, \vec{y})$ be a tgd, s.t. $I \models \phi(\vec{a}, \vec{a}_0)$ for some assignments $\vec{a}, \vec{a}_0$ on $\vec{x}$ and $\vec{x}_0$ respectively. For each such assignment $\vec{a}$, I is extended with the facts instantiating the atoms of $\psi(\vec{a}, \vec{Z})$ where $\vec{Z}$ consists of distinct labeled nulls not present in $dom(I)$. If the tgd is skolemized, the functional terms it generates are considered labeled nulls. Chase based on this definition of tgd application is often called *oblivious*, or *naïve* in the literature. A more fine-grained classification, proposed in Chapter 1, calls the so defined chase *semi-oblivious* (since the target facts are created for each assignment $\vec{a}$ and not for every combination of $\vec{a}$ and $\vec{a}_0$).

Egd chase step. Consider an egd $\tau\colon\ \phi(\vec{x}\,) \rightarrow x_i = x_j$, s.t. $I \models \phi(\vec{a}\,)$ for some assignment $\vec{a}$ on $\vec{x}$. This egd enforces the equality $a_i = a_j$. If $a_i, a_j \in const(I)$ and $a_i \neq a_j$, the chase *aborts with failure*. Otherwise, if one of a_i, a_j is a labeled null, all its occurrences in the instance are replaced by the other term in the pair (a_i, a_j).

The result of chasing an instance I with dependencies Σ *restricted to the target schema* is denoted as $chase(I, \Sigma)$. An important property of mappings with target tgds is *termination of the chase*. Unless specifically noted, here we assume that sets of dependencies are *terminating*, that is, never cause infinite sequence of chase steps on any source instance. We refer the reader to Chapter 1 for detailed discussion of chase variants and chase termination.

Homomorphisms and the core. Let I,I' be instances. A *homomorphism* $h\colon I \rightarrow I'$ is a mapping $dom(I) \rightarrow dom(I')$, s.t. (1) whenever $R(\vec{x}\,) \in I$, then $R(h(\vec{x}\,)) \in I'$, and (2) for every constant c, $h(c) = c$. An *endomorphism* is a homomorphism $I \rightarrow I$, and a *retraction* is an idempotent endomorphism, i.e. $r \circ r = r$. The image $r(I)$ under a retraction r is called a *retract* of I. An endomorphism or a retraction is *proper* if it is not surjective (for finite instances, this is equivalent to not being injective), i.e., if it sends two distinct nulls onto the same term.

▶ **Definition 2.** An instance is called a *core* if it has no proper endomorphisms. A core C of an instance I is an endomorphic image of I, s.t. C is a core.

It can be easily shown that all cores of an instance I are unique up to isomorphism [13]. We can therefore speak about *the core* of I.

Universal solutions and canonical instances. Consider a terminating schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t)$. Given a null-free source instance I, the universal solution for I under $\mathcal{M}$ can be computed as follows: We start with the instance $(I, \emptyset)$ over the combined schema $\langle \mathbf{S}, \mathbf{T} \rangle$, i.e., the source instance is I and the target instance is initially empty. Chasing $(I, \emptyset)$ with Σ_{st} yields the instance (I, J^{st}), where J^{st} is called a *preuniversal instance*: we write $\mathit{chase}(I, \Sigma_{st}) = J^{st}$, using the convention that $\mathit{chase}(I, \Sigma_{st})$ is restricted to $\mathbf{T}$. This chase always succeeds since Σ_{st} contains no egds. Then J^{st} is chased with Σ_t. If Σ_t contains egds, this chase may fail. If the chase succeeds, we end up with the instance $J = \mathit{chase}(J^{st}, \Sigma_t) = \mathit{chase}(I, \Sigma)$, which is referred to as a *canonical universal solution* for I. A *universal solution* has a homomorphism into any other solution for I. If universal solution J' is a core, it is called *the core universal solution for* I. Finally, we call an instance J over $\mathbf{T}$ *canonical*, if for some source instance I, $J = \mathit{chase}(I, \Sigma)$.

3 Complexity of core computation

We start with discussing the complexity of core computation for arbitrary instances, first studied by Hell and Nešetřil [13] and then by Fagin, Kolaitis and Popa in [7], where the following decision problems are formulated:

CoreRecognition: *Given an instance A over some schema $\mathbf{R}$ is it a core?*

According to Definition 2, the instance is the core if it has no homomorphism into its proper subinstance. Since testing for homomorphism is a well known **NP**-complete problem, it is immediate that CoreRecognition is in **coNP**. Hell and Nešetřil [13] have shown that it is actually **coNP**-complete, even if A is an undirected graph. The proof uses a reduction from Non-3-Colorability on graphs with girth (shortest cycle contained in the graph) of length at least 7. The next problem brings us yet one step further to the complexity of core computation. It was first formalized and studied by Fagin et al. in [7].

CoreIdentification: *Given an instance A and its subinstance B over some schema $\mathbf{R}$, is $\mathit{core}(A) = B$?*

The intuition suggests, that deciding CoreIdentification amounts to testing a homomorphism between A and B, and then solving CoreRecognition for B. Hence, the problem can be split into a **NP**-complete and **coNP**-complete parts, and therefore might be complete for both classes. This guess appears to be correct: Fagin et al. showed that core identification problem is **DP**-complete (where the class **DP** is the class of decision problems that can be expressed as a conjunction of an **NP** problem and a **coNP** problem). Building upon results of [13], the authors provide a reduction from 3-Colorability/Non-3-Colorability.

Taking the possible instance size into account, the above results render core computation on arbitrary instances as a prohibitively expensive task. Importantly though, in data exchange one is typically confronted with target instances with certain regularities: they must fulfill

Procedure BlockCoreComp ("The Blocks algorithm" [7])

Input: An instance J
Output: The core of J

```
(1) Identify the fact blocks $\{B_1, \dots, B_n\}$ of $J$
(2) Set $C := J$
(3) for each $X \in nulls(J)$ do
(4)     Set $C^{-X} := \{R(\vec{a}) \mid R(\vec{a}) \in J \wedge X \in \vec{a}\}$
(5)     Let $B_X \in \{B_1, \dots, B_n\}$ be the block containing $X$: $X \in nulls(B_X)$
(6)     if $X \in nulls(C)$ and exists a homomorphism $h : B_X \to C \setminus C^{-X}$ then
(7)         Set $C := (C \setminus B_X) \cup h(B_X)$
(8) return $C$
```

data dependencies and, moreover, are often created from scratch with this requirement in sight. These regularities allow to dramatically improve the efficiency of core computation.

In contrast, less assumptions can be typically made about the structure of source instances. At the same time, many data exchange frameworks disallow labeled nulls at the source side[1]. Most algorithms considered in this survey crucially depend on this simplifying assumption. Thus, speaking of core computation for data exchange, we assume that source instances do not contain nulls.

The next section is devoted to the complexity of core computation relative to certain structural parameters of the instance. In Section 4, these parameters will be related to syntactic properties of schema mappings.

3.1 Parameterized complexity

Core computation comes down to a search for homomorphisms. The decision version of this problem can be formulated as follows:

Homomorphism: *Given instances A, B over schema $\mathbf{R}$, does $A \to B$ hold?*

This problem can be reformulated as the problem of evaluation of boolean conjunctive queries (BCQ), one of the most thoroughly studied topics in database theory [2]. Hence, numerous results for BCQ evaluation carry over to Homomorphism, and vice versa. One of the most immediate parameters for Homomorphism is the maximal size of independent subinstance, called *fact block*.

► **Definition 3.** *Fact blocks* are connected components of the *fact graph* of an instance J, where the fact graph has the facts of J as vertices and edges drawn between two vertices whenever the facts at these vertices share a labeled null. We define $blocksize(J)$ as $\max\{|nulls(B)| \mid B$ is a fact block of $J\}$. Finally, for a null $X \in nulls(J)$, $B(X)$ denotes the fact block of J which contains X.

It follows immediately from the definition, that for two distinct blocks B_1, B_2 in J, $nulls(B_1) \cap nulls(B_2) = \emptyset$. Hence, every homomorphism h for J can be decomposed into a union of homomorphisms $h_i : B_i \to A$ where i ranges over all blocks of J, and the homomorphisms h_i and h_j can be defined independently of each other if $i \neq j$. This

[1] Data exchange semantics for source instances with nulls has been proposed in [8]

motivates the *Blocks algorithm* BlockCoreComp, first considered in [7]. It searches for local block-wise homomorphisms that eliminate at least a single null from the domain of J. For any fact block $B \subseteq J$ the homomorphism $h : B \to J$ can be immediately turned into an endomorphism on J by taking it in a union with the identity mapping on all other fact blocks of J. Hence, BlockCoreComp computes a sequence of nested instances $J \supseteq J_1 \ldots \supseteq J_n$, such that the endomorphisms $J \to J_1 \ldots \to J_n$ hold, and $J_n = core(J)$.

The BlockCoreComp algorithm can be shown to run in time $O(|nulls(J)| \cdot (c + m)$, where $m = |I|$ and c is the cost of the homomorphism test at step 5. This cost depends crucially on $blocksize(J)$, which we will denote by b.

A naïve way of testing a single homomorphism $B \to J$ takes time $O(|dom(J)|^b)$. This result can be improved considerably by employing such parameters of J as treewidth $tw(J)$ [14], query-width $qw(J)$ [3], or hypertree-width [11]. Gottlob and Nash take the latter parameter, and describe a procedure for deciding Homomorphism, that on the input $(B(X), C \setminus C^{-X})$ on line (5) of BlockCoreComp takes time $O(m^{\lfloor b/2 \rfloor + 2})$, where $hw(B(X))$ is approximated by b.

A number of favorable properties of hypertree decomposition is given in [12], motivating its usage for core computation. In particular, the hypertree decomposition is

- Robust: for every instance J, $hw(J) \leq qw(J) \leq tw(J)$ holds. Moreover, there exist instances J', J'', for which inequalities $hw(J') < qw(J')$ and $qw(J'') < tw(J'')$ are proper.
- Useful: Homomorphism(J, A) can be decided in time $O(t \cdot a^k)$, where a is the size of the largest relation in A, t a number of hypernodes in the hypertree decomposition of J and k is a bound for $hw(J)$. Moreover, $k \leq \lfloor b/2 \rfloor + 1$, where $b = blocksize(J)$.
- Efficiently decidable: For each fixed constant k, the problems of determining whether $hw(J) \leq k$ and of computing (in the positive case) a hypertree decomposition of width $\leq k$ are feasible in polynomial time.

The precise complexity of BlockCoreComp relative to treewidth and query-width has not been considered in the literature so far, but can be derived easily from the results on CQ evaluation.

In all the expressions above, we have $b = blocksize(J)$ in the exponent of the running time estimation. The authors of [12] show that this cannot be avoided (unless **P**=**NP**). They use a parameterized reduction from the k-CLIQUE problem, to show the following:

▶ **Theorem 4.** *[12] If J has $blocksize(J) \leq k$ and $C \subseteq J$ is null-free, then the problem* CoreIdentification *(J, C) is fixed parameter-intractable in the parameter k.*

4 Core computation as a post-processing

In this section we show how the knowledge that the given instance results from the chase with certain type of dependencies can be leveraged to facilitate the core computation.

Most post-processing algorithms restrict the homomorphism search to subinstances whose *blocksize* only depends on the given mapping, thus achieving polynomial data complexity of the core computation. In Section 4.1, we also present a particularly simple Greedy algorithm [7] which requires no endomorphism search at all, provided that the target dependencies of the mapping contain no tgds.

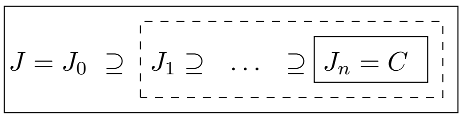


Figure 1 Recursive core approximation.

All currently known post-processing algorithms follow the *recursive approximation scheme*, in which the core is found as a sequence of ever shrinking endomorphic images of the canonical universal solution J, such that each image can be seen as an approximation to the core. It is desirable that every such core approximation J_i be itself a *universal solution* to the data exchange problem. Then, each subsequent approximation can be immediately used for query answering instead of the previous one (and instead of the original canonical instance J); the core is found when no further improvement of the current approximation is possible. The BlockCoreComp algorithm, however, never checks that the output instance C is a solution. It is easy to show, that for mappings with target constraints restricted to egds, each iteration of BlockCoreComp delivers a universal solution, without further ado. Indeed, the algorithm computes a sequence of nested endomorphic images. Egds are closed under the subset relation, while for st-tgds the following lemma holds:

▶ **Lemma 5.** *Let Σ be a set of st-tgds, I, J be respectively a source and a target instance, and let h be a homomorphism on J. Then, $(I, J) \models \Sigma$ implies $(I, h(J)) \models \Sigma$.*

Proof idea. The proof is based on the three observations: (1) I contains no nulls, (2) h preserves constants, and (3) conjunctive queries are closed under homomorphisms. Assume that for a st-tgd τ and some assignment $\vec{a}$ satisfying the antecedent of τ there is an assignment $\vec{b}$ for $\exists$-variables of τ. Then, $h(\vec{b})$ is also a satisfying assignment for $\exists$-variables of τ. ◀

In presence of target tgds, however, the situation is a little more complex. Consider the following example:

▶ **Example 6.** [12] Let J be an instance with a binary relation R containing the following tuples: $\{(X, Z), (X, a), (a, a), (Z, Y), (a, Z)\}$, were a is a constant and the other values are labeled nulls. Then, $h = \{X \to Z, Y \to Z, Z \to a, a \to a\}$ is an endomorphism on A, $h(J) = \{(X, a), (a, a), (a, Z)\}$. Now, the following full tgd is satisfied by A but not by $h(A)$:

$$R(x_1, x_2) \wedge R(x_2, x_2) \wedge R(x_2, x_3) \to R(x_1, x_3)$$

Indeed, applied to $h(J)$, the tgd yields (X, Z), which is not part of $h(J)$. ◀

The above endomorphism is somewhat particular: namely, it is non-idempotent, mapping Z onto a and re-introducing it as a image of Y. As shown by Gottlob and Nash, no such example would be possible for an idempotent endomorphism (retraction):

▶ **Lemma 7.** *[12] Let J be an instance, and $r : J \to J$ its retraction. Then, for arbitrary set of tgds and egds Σ, $J \models \Sigma$ implies $r(J) \models \Sigma$.*

Hence, the core approximation via proper retractions is a viable alternative for mappings with target tgds. Their use becomes even better justified, taking into account the cost of transformation of an arbitrary endomorphism into an idempotent one, as demonstrated by Gottlob and Nash:

▶ **Lemma 8.** *[12] Let h be a proper endomorphism on some instance J: that is, $\exists x, y \in dom(J)$, such that $h(x) = h(y)$. Then, there exists a retraction $r \colon J \to h(J)$, such that $r(x) = r(y)$. Moreover, such r can be found in time $O(|dom(J)|^2)$.*

Proof hint. The retraction r can be obtained by computing a sequence $h = h_0, h_1, \ldots, h_k = r$ of endomorphisms, where h_{i+1} is obtained by composing h_i with itself n_i times. It can be shown that $\Sigma_{0 \leq i < k}\, n_i \leq |dom(J)|^2$. We will refer to this iteration-based transformation as to Procedure ToRetraction. ◀

Procedure GreedyCoreComp ("The Greedy algorithm" [7])

Input: Source instance I, schema mapping $\Sigma = \Sigma_{st} \cup \Sigma_t$ where $\Sigma_t = \emptyset$ or consists of egds
Output: The core universal solution for I under Σ

```
(1) Set $J := chase(I, \Sigma)$,    $C := J$
(2) for each $R(\vec{a}) \in J$ do
(3)     if $\langle I, C \setminus \{R(\vec{a})\}\rangle \models \Sigma_{st}$ then
(4)         Set $C := C \setminus \{R(\vec{a})\}$
(5) return $C$
```

In the remainder of this section, post-processing core computation is considered under different classes of target dependencies. Most algorithms that we present deliver core approximations which are universal solutions. If target constraints are restricted to egds, this property comes for free. In presence of target tgds, transforming proper endomorphisms into proper retractions is needed. The only complicated case is when target dependencies comprise both tgds and egds, since some algorithms *simulate egds by tgds* and thus egds may not be satisfied until the core is found. This issue is addressed in Section 4.4.

4.1 No target constraints

The Blocks algorithm. In the absence of target constraints, each canonical instance has *blocksize* bounded by the $\exists$-*width* of the mapping, as can be readily seen from the definition of a chase step with a tgd. Indeed, each such step instantiates the $\exists$-variables of a tgd with fresh distinct nulls, and hence two facts created at different chase steps never share a null.

Hence, the BlockCoreComp algorithm from Section 3.1 can be applied to $chase(I, \Sigma)$ without any modifications, and the inequality $blocksize(J) \leq \exists\text{-}width(\Sigma)$ holds.

The Greedy algorithm. The Greedy algorithm GreedyCoreComp is defined for mappings whose set of target constraints is empty or consists of target egds. This procedure does not explicitly check the existence of an endomorphism from the canonical universal solution J to its subinstance C. This is not an omission, since C is a solution for I: the test on line 3 verifies that Σ_{st} is satisfied after the atom $R(\vec{a})$ is removed. Being a universal solution, J has a homomorphism into any other solution, so $J \to C$ holds after each iteration. Obviously, $C \to J$ holds too, by $C \subseteq J$. Hence, this algorithm does not depend on the block size of J, but rather on the complexity of evaluating the st-tgds in Σ_{st} over $\langle I, C\rangle$. This approach is not quite in the spirit of data exchange, however, since the test on line 3 requires the source instance to be accessible all the time until the core is not found.

4.2 Target egds

The Blocks algorithm. If mapping includes target egds, the *blocksize* of the instance is not fixed anymore, as can be seen on a following simple example:

▶ **Example 9.** Consider the mapping Σ with one st-tgd and one egd:

- $S(x, z) \to \exists y_1, y_2\ P(x, y_1) \wedge R(z, y_1, y_2)$
- $R(x, x_1, y_1) \wedge R(x, x_2, y_2) \to y_1 = y_2$

For each source instance I, the canonical preuniversal instance $chase(I, \Sigma_{st})$ has *blocksize* 2. The *blocksize* of canonical universal solution, however, is $k + 1$, where k is the maximal number of distinct facts in I that agree on the second attribute. ◀

Recall that the complexity of core computation depends exponentially on *blocksize*. Therefore, the unbounded *blocksize* would effectively make core computation intractable. The idea due to Fagin et al. [7] is to *redefine the notion of fact block* in order to keep its size bounded despite the effects of egds. Recall the process of constructing a universal solution via chase: first the source instance I is chased into a preuniversal canonical instance J^{st}, to which, in turn, the target egds are applied. The following property, discovered by Fagin et al., will play an important role in several core computation algorithms:

▶ **Lemma 10** (Rigidity). *[7] Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ be a mapping in which Σ_t consists of target egds, and two labeled nulls $X, Y \in nulls(J^{st})$ belong to different fact blocks in J^{st}. If the chase of target egds enforces the unification of X and Y so that they are both substituted by a term t in the canonical universal solution J, then t is* rigid *in J: That is, for any endomorphism e on J, $e(t) = t$ holds.*

The intuition behind this property is that conjunctive queries in the antecedents of egds cannot distinguish between the fact block and its endomorphic image, and thus make egds perform the same labeled null unifications in both. The consequence for core computation is very favorable: nulls affected by egds during the target chase can be treated as constants. Hence, we consider *non-rigid fact blocks* (*nr-blocks* for short) constructed as in Definition 3, but disregarding the sharing of rigid nulls between facts. A corresponding parameter of the target instance measuring the maximal number of nulls in an nr-block of an instance is called *nr-blocksize*, and the BLOCKCORECOMP algorithm can be adapted accordingly.

Since non-rigid fact blocks are contained in the fact blocks of the preuniversal instance (modulo unification of nulls), target egds actually facilitate core computation. The only downside is the necessity to track egd applications in order to identify rigid nulls.

The Greedy algorithm. The procedure GREEDYCORECOMP defined in Section 4.1 handles mappings with target egds without any modification. Note that line 3 only checks that the source-to-target constraints in Σ are satisfied after the fact $R(\vec{a})$ is eliminated from C. It is easy to show that whenever an instance J satisfies an egd, then any its subinstance does so, too. Since C is a subinstance of J and $J \models \Sigma_t$, $C \models \Sigma_t$ holds as well.

4.3 Target tgds

Neither Greedy nor Blocks algorithm can be easily extended to support target tgds. The problem with the Greedy algorithm is that unlike egds, tgds are not closed under the subset relation, so the test if the combined instance $\langle I, C \setminus \{R(\vec{a})\}\rangle$ satisfies Σ_t as well as Σ_{st} needs to be performed at each iteration in addition to the test $\langle I, C \setminus \{R(\vec{a})\}\rangle \models \Sigma_{st}$. Moreover, eliminating a single fact at a time is no longer sufficient in presence of target tgds.

The negative effect of target tgds on the Blocks algorithm is twofold. Besides merging the blocks of a preuniversal instance (by putting nulls from different blocks into the same fact), chase with target tgds can introduce a polynomial number of new nulls, and thus the blocks of $chase(I, \Sigma_{st} \cup \Sigma_t)$ can be substantially larger than those of $chase(I, \Sigma_{st})$. No direct analog of the Rigidity property is available for the case of target tgds. Thus, new ideas are needed to tackle mappings with target tgds. In this section we describe one such idea by Gottlob and Nash, implemented in their algorithm FINDCORE [12].

Let J be a canonical instance for some mapping $\mathcal{M}$, and $C \subseteq J$ be its current core approximation, satisfying $\mathcal{M}$. C is the core if there is no endomorphism r from J into some *proper subinstance* of C. It is easy to show, that any such r must unify at least two terms from the domain of C: that is, $\exists X, Y \in dom(C)$ such that $r(X) = r(Y)$. Gottlob and Nash proposed a construction of a bounded-size instance $K \in J$, $X, Y \in \mathit{dom}(K)$, such that a desired r exists iff a homomorphism $h : K \to C$ exists, with $h(X) = h(Y)$. We will call such K a *kernel* of J w.r.t. the terms X, Y, written K_{XY}. (Note that it is *not a problem kernel*, as used in the parameterized complexity theory, but a database instance). To define K_{XY}, some new definitions are needed.

▶ **Definition 11** (Parents, Ancestors, Siblings)**.** Let $\vec{Y}$ be a vector of nulls created by a tgd $\tau\colon \phi(\vec{x}) \to \exists \vec{y}\ \psi(\vec{x}, \vec{y})$, that fired with a satisfying assignment $\vec{a}$ for the variables of ϕ. Then, the elements in $\vec{a}$ are called *parents* of each null in $\vec{Y}$. Moreover, all elements of $\vec{Y}$ are called *siblings* w.r.t. each other, and *τ-children of $\vec{a}$*, written $\vec{a} \overset{\tau}{\Rightarrow} \vec{Y}$. The *ancestor* relation is then defined as a transitive closure over parents.

We say that a subinstance K of a canonical instance J is closed under parents and siblings, if (1) whenever $X \in \mathit{nulls}(K)$, then parents and siblings of X are in $\mathit{dom}(K)$, and (2) all facts of J which are over $\mathit{dom}(K)$ are in K.

▶ **Definition 12** (Depth)**.** The depth of constants (copied from the source instance or contained in the right-hand side of the tgd) is taken to be 0. Then, the depth of each labeled null is defined to exceed by one the maximal depth of its parents.

For a broad class of terminating mappings, every null in the canonical target instance has depth bounded by a constant depending only on the mapping: we say that such a mapping has the *bounded depth property.* Gottlob and Nash used this insight to limit the kernel size $|K_{XY}|$. The results of Marnette [16] imply that a similar property holds for arbitrary terminating mappings defined by tgds, and thus FindCore can be lifted to handle all such mappings. This lifting will be the subject of Section 4.3.1.

The following example shows how the depth of a labeled null can remain small even though its derivation takes a long sequence of chase steps.

▶ **Example 13.** Let mapping Σ consist of a single st-tgd τ_{st} and three target tgds $\tau_{1,2,3}$:

- $\tau_{st}\colon E(x_1, x_2) \to \exists y\ D(x_1, y) \wedge D(x_2, y)$
- $\tau_1\colon D(x_1, x_2) \wedge D(x_1, x_3) \to C(x_2, x_3)$
- $\tau_2\colon C(x_1, x_2) \wedge C(x_2, x_3) \to C(x_1, x_3)$
- $\tau_3\colon D(2, x_1) \wedge C(x_1, x_2) \to \exists z\ C_2(x_2, z)$

τ_{st} copies the vertices of a graph given by the source relation E (a list of egdes) along with the unique edge identifier, generated as a labeled null. The binary relation C is then initialized by τ_1 with the pairs of identifiers of adjacent edges. The transitive closure of C is computed by τ_2. Finally, τ_3 puts in C_2 the identifiers of edges which belong to the connected component with a specific vertex "2". Although C_2 depends on the transitive closure of C and thus takes unlimited number of chase steps to compute, any null in it has bounded depth. Indeed, the nulls in D are created by the source-to-target chase, and thus have depth 1. C contains only the nulls from D. The tgd τ_3 which populates C_2 depends on C and on D and therefore creates nulls of depth at most 2. ◀

We are now ready to define a kernel subinstance K_{XY}:

▶ **Definition 14** (Kernel)**.** Let J be a canonical universal solution under a terminating mapping $\mathcal{M}$ defined by st-tgds and target tgds, and let J^{st} be its preuniversal subinstance. Given a pair $X, Y \in \mathit{dom}(J)$, let A_{XY} be a minimal set of terms containing X, Y and closed under the parent and sibling relations. Then, the kernel K_{XY} of J is defined as a set of all facts of J over A_{XY}, together with the fact blocks of J^{st} having nulls in common with A_{XY}: $\bigcup_{N \in \mathit{nulls}(A_{XY})} B^{st}(N)$, where B^{st} denotes the fact blocks of J^{st}.

```
Procedure EXTEND

Input:   Canonical universal solution $J$,
         retraction $r$ for $J$,
         homomorphism $h : K \to r(J)$ where $K \subseteq J$ closed under parents and siblings.
Output: Endomorphism $g : J \to r(J)$ such that $\forall x \in dom(h)\, g(x) = h(x)$

(1)  Initialize   $g(x) = \begin{cases} h(x) & \text{if } x \in dom(h) \\ r(x) & \text{if } x \in dom(J^{st} \setminus dom(h)) \end{cases}$
(2)  while $dom(g) \subset dom(J)$ do
(3)      Find a tgd $\tau \in \Sigma_t$,  $\vec{a} \in dom(g)$ and $\vec{Y} \in nulls(J) \setminus dom(g)$, such that $\vec{a} \overset{\tau}{\Rightarrow} \vec{Y}$;
(4)      Set $g := g \cup \{\vec{Y} \to r(\vec{Y}')\}$,  where $g(\vec{a}) \overset{\tau}{\Rightarrow} \vec{Y}'$;
(5)  return $g$;
```

▶ **Theorem 15** (Properties of K_{XY})**.** *Let J be a canonical universal solution under a mapping $\mathcal{M}$ defined by st-tgds and target tgds and having the bounded depth property. Let r be a retraction on J and let $C = r(J)$. Then, for any two terms $X, Y \in dom(C)$, the kernel K_{XY} constructed according to Definition 14 has the following properties:*

1. *$|K_{XY}|$ is bounded by a constant depending solely on $\mathcal{M}$.*
2. *An endomorphism $g : J \to C$, $g(X) = g(Y)$ exists if and only if the homomorphism $h : K_{XY} \to C$ exists, such that $h(X) = h(Y)$.*

Proof Sketch. Claim (1) follows from the Definition 14. Indeed, an estimation $|A_{XY}| \leq 2edw^d$ can be shown by induction, where where w and e are respectively $\forall$-*width* and $\exists$-*width* of the mapping, and d its depth (see [12], Lemma 1). A combination with the blocks of J^{st} raises the bound to $2e^2dw^d$ ($blocksize(J^{st})$ is at most e). This is a constant, as we only consider data complexity and take the mapping fixed. Thus, also the number of facts in K_{XY} is bounded (the target schema is fixed, and there is only a constant number of distinct facts one can build over a fixed domain).

For Claim (2), suppose no homomorphism $K_{XY} \to h(J)$ can unify X and Y. Since K_{XY} is a subinstance of J, there is also no endomorphism of J with this property. Now, to the contrary, let h be an arbitrary homomorphism $h : K_{XY} \to r(J)$. Such a homomorphism can be extended to an endomorphism g on J, consistent with h:

(a) We initialize g to be a homomorphism $W \to C$, where W is an instance, similarly to K_{XY} closed under ancestors and siblings, but also containing the preuniversal subinstance J^{st} of J. To do so, we define

$$g(x) = \begin{cases} h(x) & \text{if } x \in dom(h)\,, \\ r(x) & \text{if } x \in dom(J^{st} \setminus dom(h)) \end{cases}$$

To see that g is indeed a homomorphism it suffices to note that the facts in K_{XY} and in $J^{st} \setminus K_{XY}$ do not have nulls in common, as readily follows from Definition 14.

(b) We now extend g to an endomorphism on the whole instance J. The extension proceeds by "replaying" the chase steps:

- Let $\phi(\vec{x}) \to \psi(\vec{x})$ be a full target tgd in $\mathcal{M}$, and let $\vec{a}$ be an assignment for $\vec{x}$ such that $W \models \phi(\vec{a})$ but $W \not\models \psi(\vec{a})$. Then, W is extended with $\psi(\vec{a})$. After this step, g is still a homomorphism for W, since $W \to C$ implies $C \models \phi(g(\vec{a}))$, $C \models \mathcal{M}$ and hence, also $C \models \psi(g(\vec{a}))$ holds.

```
Procedure FindCore
Input:  Source instance $I$, terminating mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$
Output: Core universal solution for $I$ under $\mathcal{M}$

(1) Set $J := chase(I, \Sigma)$, and let $J^{st}$ be a preuniversal subinstance of $J$, $J^{st} = chase(I, \Sigma_{st})$
(2) Initialize retraction $r$ to be identity on $dom(J)$;
(3) for each $X \in nulls(r(J))$, $Y \in range(r)$, $X \neq Y$ do
(4)     Compute $K_{XY}$;
(5)     if exists $h : K_{XY} \to r(J)$ s.t. $h(X) = h(Y)$ then
(6)         Set $g :=$ Extend$(J, r, h)$;
(7)         Set $r :=$ ToRetraction$(g)$;        //Lemma 8
(8) return $r(J)$
```

- For a non-full target tgd $\tau\colon \phi(\vec{x}) \to \exists\vec{y}\ \psi(\vec{x}, \vec{y})$ in $\mathcal{M}$ such that $W \models \phi(\vec{a})$ but the τ-children $\vec{Y}$ of $\vec{a}$ are not in $dom(W)$, both W and g have to be extended: W is augmented with $\psi(\vec{a}, \vec{Y})$, while g is extended to map $\vec{Y}$ into $dom(C)$. Since J has been created by the oblivious chase, we know that there are τ-children of $g(\vec{a})$ among the nulls of J: $\vec{Y}' \in nulls(J)$, such that $g(\vec{a}) \overset{\tau}{\Rightarrow} \vec{Y}'$ holds. Moreover, as C is a retract of J, $C \models \psi(g(\vec{a}), r(\vec{Y}'))$ holds as well. It suffices to extend g in order to map elements of $\vec{Y}$ onto the respective elements of $r(\vec{Y}')$.

By replaying all chase steps with non-full tgds, g can be extended to an endomorphism on J. It takes the time linear in the size of J, provided that the parent relation is maintained during the chase of J.[2] Procedure Extend captures this idea. ◀

Finally, we can define the core computation algorithm FindCore, which, given a terminating mapping $\mathcal{M}$, performs the following steps. It starts with a trivial automorphism r, and tries to improve it: The main cycle of FindCore performs an exhaustive search for all pairs X, Y where $X \in nulls(r(J))$, $Y \in range(r)$ and a homomorphism $h : K_{X,Y} \to r(J)$ exists, such that $h(X) = h(Y)$. When such homomorphism h is found, it is lifted to an endomorphism g on J by applying the procedure Extend. Since g unifies at least two terms in $r(J)$, it is an *improvement* for r and can be shown to send J onto some proper subset of $r(J)$. Thus, r is updated to be g and transformed into a retraction by an iterative procedure from the proof of Lemma 8, after which the computation starts over from line 3.

4.3.1 Core computation for skolemized mappings

In [16] Marnette made a number of important contributions to the problem of tractability of core computation. He proved that the idea of FindCore is applicable to any terminating mapping based on tgds, and that this result can be extended to support egds by encoding them as tgds. Moreover, the reformulation of FindCore for skolemized mappings resulted in a simpler and more efficient version of the algorithm.

▶ **Example 16** (Skolemized mapping)**.** Consider the skolemization of the mapping from Example 13. The full tgds $\tau_{1,2}$ are not affected by the skolemization. The dependencies τ_{st} and τ_3 after skolemization have the form

- $\tau_{st}^{sk}\colon E(x_1, x_2) \to D(x_1, f_{st}(x_1, x_2)) \wedge D(x_2, f_{st}(x_1, x_2))$
- $\tau_3^{sk}\colon D(2, x_1) \wedge C(x_1, x_2) \to C_2(x_2, f_3(x_2))$ ◀

[2] For skolemized mappings, no special tracking of parents is required, see Section 4.3.1

Note that the chase with skolemized mappings produces nulls labeled with Skolem terms. These terms can be nested: in particular, the dependency τ_3^{sk} in Example 16 generates terms of the form $f_3\big\langle f_{st}\langle\cdot,\cdot\rangle\big\rangle$. Each such Skolem term denoting a labeled null in the target instance contains its *ancestors* (see Definition 11) as *subterms*. The notion of ancestor here is refined in comparison to Definition 11. Consider the dependency τ_3^{sk}. According to Definition 11, each null introduced by τ_3^{sk} in the C_2 relation has two parents, instantiating the variables x_1 and x_2. However, the Skolem term in τ_3^{sk} only has x_2 as argument, since x_1 does not occur in the conclusion of the dependency. This skolemization strategy ensures that the size of each Skolem term in an instance created by chasing a mapping has bounded size, provided that the mapping is based on tgds and is terminating.

The above observation is crucial, since the kernel K_{XY} from Definition 14 now can be redefined using closure under subterms. Also the procedure EXTEND, lifting a homomorphism on a kernel to an endomorphism on the target instance can be defined much more concisely than for the non-skolemized tgds:

Given a retraction r selecting the current approximation of the core of the canonical universal solution J (with the preuniversal instance J^{st}), and the homomorphism $h : K_{xy} \to r(J)$, the endomorphism g for J is defined recursively as

$$g(x) = \begin{cases} x & \text{if} \quad x \in \mathit{const}(J)\,, \\ h(x) & \text{if} \quad x \in \mathit{dom}(h)\,, \\ r\left(f\big\langle e(t_1),\ldots,e(t_n)\big\rangle\right) & \text{if} \quad x \notin \mathit{dom}(h) \cup \mathit{const}(J) \text{ and } x = f\langle t_1,\ldots,t_n\rangle\,. \end{cases}$$

This formulation is similar to that given in the proof of Claim (2) of Theorem 15 and replaces the procedure EXTEND for skolemized mappings. A clear advantage of this new extension procedure is that no special tracing of the chase process is needed, in contrast to the original procedure. Another improvement of [16] to FINDCORE is concerned with target egds and therefore will be considered in the next section.

4.4 Target tgds and egds

The FINDCORE algorithm can be extended to the case when target dependencies contain egds in addition to tgds. The strategy, considered by Gottlob and Nash [12] and by Marnette [16], is to encode egds by target tgds, generating special "equality" facts in the target instance. This solution fits quite naturally to the setting of [16], which assumes databases with equality constraints.

The set Σ_t of egds and tgds over the target schema $\mathbf{T}$ is transformed into the set $\bar{\Sigma}_t$ of tgds over the schema $\mathbf{T} \cup \{\mathcal{E}\}$, where equality relation $\mathcal{E}$ is not part of $\mathbf{T}$. The transformation in [12] consists of the following steps:

1. Replace all equations $x = y$ with $\mathcal{E}(x,y)$, turning every egd into a tgd.
2. Add constraints for *symmetry* $\mathcal{E}(x,y) \to \mathcal{E}(y,x)$, *transitivity* $\mathcal{E}(x,y) \wedge \mathcal{E}(y,z) \to \mathcal{E}(x,z)$, and *reflexivity* of $\mathcal{E}$: $R(x_1,\ldots,x_k) \to \mathcal{E}(x_i,x_i)$ for every $R \in \mathbf{T}$ and $i \in \{1,2,\ldots,k\}$.
3. Add *consistency* constraints: $R(x_1,\ldots,x_k), \mathcal{E}(x_i,y) \to R(x_1,\ldots,y,\ldots,x_k)$ for every $R \in \mathbf{T}$ and $i \in \{1,2,\ldots,k\}$.

The consistency constraints are problematic, as they can cause non-termination of the mapping:

▶ **Example 17** ([16]). Consider two target dependencies τ, ϵ, and a tgd encoding τ_ϵ of the latter, together with the $\mathcal{E}$-symmetry constraint τ_s and the consistency constraint τ_R^c for R:

- $\tau\colon R(x) \to \exists y\ P(x,y)$
- $\epsilon\colon P(x,x') \to x = x'$
- $\tau_\epsilon\colon P(x,x') \to \mathcal{E}(x,x')$
- $\tau_s\colon \mathcal{E}(x,x') \to \mathcal{E}(x',x)$
- $\tau_R^c\colon$ $R(x) \wedge \mathcal{E}(x,x') \to R(x')$

While the original set of dependencies $\{\tau, \epsilon\}$ is terminating, the rewriting $\{\tau, \tau_\epsilon, \tau_s, \tau_R^c\}$ is not: Oblivious chase does not terminate on *any* instance with a non-empty relation R, while the non-oblivious chase terminates only if τ_s is satisfied before τ_R^c. ◀

Gottlob and Nash address this problem by defining a special *nice* order of tgd applications in the non-oblivious chase, determined at execution time. In [16] Marnette gives an improved encoding scheme coinciding with the approach used by Duschka et al. for query answering using views [4]. It is based on so-called *rectification* of antecedents: for instance, a rectification of $P(x,x) \wedge Q(x,z)$ is $P(x,x') \wedge Q(x'',z) \wedge \mathcal{E}(x,x') \wedge \mathcal{E}(x,x'')$, while the tgd τ from Example 17 after rectification rewrites as $R(x) \wedge \mathcal{E}(x,x') \to \exists y\ P(x',y)$. Consistency constraints can now be avoided.[3]

Whatever encoding scheme is chosen, it should be noted that $chase(I, \Sigma_{st} \cup \bar{\Sigma}_t)$ is not a universal solution and may violate the egds of Σ_t. In [16] this is circumvented by assuming that the target instance includes equality constraints, whereas in the encoding approach of [12] the satisfaction of egds is a by-product of core computation. The subinstances found by the iterations of post-processing algorithms suffer from the same problem. A further disadvantage is a necessity to instantiate the $\mathcal{E}$-facts in the target instance, instead of unifying the nulls and thus reducing its domain size.

These shortcomings motivated the introduction of FINDCORE^E by Pichler and Savenkov [20], an adaptation of FINDCORE for the immediate application of egds in the chase. The main idea is to redefine the kernel K_{XY}, using the *parent* relation over *facts* rather than nulls (the latter is not robust w.r.t. egds). To identify facts, each relation in the target schema is equipped with a new Id attribute, to be instantiated with fresh unique nulls (fact identifiers) and neither copied to other facts nor affected by egds (cf. Example 18).

The *sibling facts* are those created at the same chase step. The assumption is, that sibling facts always form a single *fact block*. Such assumption is harmless, since a tgd $\phi(\vec{x}) \to \exists \vec{y}_1, \vec{y}_2\ \psi_1(\vec{x}, \vec{y}_1) \wedge \psi(\vec{x}, \vec{y}_2)$ where $\vec{y}_1 \cap \vec{y}_2 = \emptyset$ can be rewritten as $\phi(\vec{x}) \to \exists \vec{y}_1\ \phi_1(\vec{x}, \vec{y}_1)$ and $\phi(\vec{x}) \to \exists \vec{y}_2\ \psi_2(\vec{x}, \vec{y}_2)$. If no such rewriting is possible, the tgds are called *normalized.*

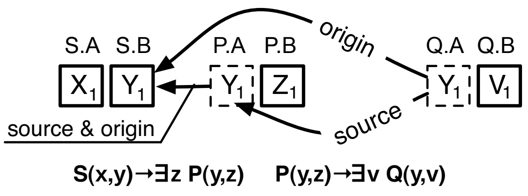


Figure 2 Parent relation over facts.

As in the tgds-only case, the parent relation ensures a bound on the kernel size. To define it, the notion of *term position* is introduced as a pair (T, A) of a tuple id T and attribute name A; such a position is called *native* if at the chase step with the tgd τ introducing T, a fresh null was created for the attribute A in T; otherwise, the position is called *foreign.* The *origin* of a native position (T, A) is defined as the fact T and its sibling facts; if (T, A) is foreign, we first find its *source* as a position (T', A'), from which τ has copied the value to instantiate (T, A): T' is among the facts that satisfied the antecedent of τ, and at the moment of instantiation, (T, A) has the same value as its sources. The origin of a foreign position is than defined as the origin of any its source position (chosen non-deterministically). Finally, the *parent facts* of T and its sibling facts S_T is the union of the origin facts for foreign positions in $\{T\} \cup S_T$.

[3] In [16, 17] Marnette proves that core computation remains tractable for mappings whose encodings according to the rectification scheme are terminating. However, it is never explicitly discussed if this result holds for *any terminating mapping* with tgds and egds as target dependencies.

▶ **Example 18.** Consider two target tgds from Figure 2 of which an *id-aware* version is

- $\sigma_1\colon S(t_s, x, y) \to \exists t_p, z\ P(t_p, y, z)$
- $\sigma_2\colon P(t_p, y, z) \to \exists t_q, v\ Q(t_p, z, v)$

and assume that the preuniversal instance contains the fact $S(T_s, X_1, Y_1)$. With its antecedent satisfied by the fact T_s, σ_1 yields a fact $R(T_r, Y_1, Z_1)$, and then σ_2 introduces $Q(T_q, Z_1, V_1)$, where T_{rq}, Y_1, Z_1 are fresh nulls. The three facts are shown in Figure 2, without the ids. Although the Q-fact was introduced by a tgd firing on the fact T_p, T_p is not a parent of T_q, since it has not contributed unique nulls to it: V_1 is native to T_q, whereas the origin of Y_1 at the foreign position of T_q is the fact T_s. Hence, T_s is the only parent of T_q (and of T_p). ◀

Similarly to the Definition 14, the kernel K'_{XY} is defined as a set containing the origin facts of X, Y, and closed over the siblings and parents relation (on facts). No other facts of J^{st} resp. J have to be taken in the kernel, unlike the original definition from Section 4.3. If target constraints consist of tgds, the inclusion $K'_{XY} \subseteq K_{XY}$ holds, where K_{XY} is constructed according to Definition 14. Moreover, the rigidity of nulls has to be taken into account for proving an analog of Theorem 15 for mappings with tgds and egds.

5 Direct core computation

The algorithms presented so far followed the same general strategy: they first created a solution with redundant facts, and then optimized it. An immediate question is, if it would be possible to create only the necessary facts in the first place. This is the goal of direct core computation. This question has been conceived already by Fagin et al. in [7]. They pointed out, that simple rewriting of individual rules is not enough, by giving the following example:

▶ **Example 19.** Consider an instance $I = \{S(1, 1, 2, 3)\}$ chased with the two st-tgds $\tau_{1,2}$:

- $\tau_1\colon S(a, b, c, d) \to \exists y_1 \exists y_2 \exists y_3 \exists y_4 \exists y_5$
  $R(y_5, b, y_1, y_2, a)$
  $\wedge R(y_5, c, y_3, y_4, a)$
  $\wedge R(d, c, y_3, y_4, b)\,)$
- $\tau_2\colon S(a, b, c, d) \to \exists y_1 \exists y_2 \exists y_3 \exists y_4 \exists y_5$
  $R(d, a, a, y_1, b)$
  $\wedge R(y_5, a, a, y_1, a)$
  $\wedge R(y_5, c, y_2, y_3, y_4)\,)$

The chase of I yields the following six facts (left column is due to τ_1, the right one to τ_2):

$$\begin{array}{ll}
R(N_5, 1, N_1, N_2, 1), & \underline{R(3, 1, 1, N'_1, 1)},\\
R(N_5, 2, N_3, N_4, 1), & R(N'_5, 1, 1, N'_1, 1),\\
\underline{R(3, 2, N_3, N_4, 1)}, & R(N'_5, 2, N'_2, N'_3, N'_4).
\end{array}$$

The core universal solution contains the two underlined facts. However, in isolation each tgd yields an instance which cannot be reduced. ◀

This example sheds some light on the intricacy of direct core computation. In particular, it is clearly not possible to consider individual st-dependencies, or update the definition of the chase step, without taking the interference between different st-tgds into account. Since the above example was published in 2005, it was not until 2009 that a full-fledged solution for direct core computation has been proposed, at least for the case of mappings without target constraints: Core schema mappings by Mecca et al. [19] and Laconic schema mappings by ten Cate et al. [22]. We will give an overview of these approaches in the next subsection.

5.1 No target dependencies

The essence of direct core computation is predicting which dependencies can eventually introduce redundant facts (that is, facts which are not part of the core), and under which conditions. In absence of target constraints, there is a finite number of ways in which st-tgds can interfere with each other. This gave rise to two approaches which we consider in this section.

5.1.1 Core schema mappings

An illustrative example of the interference between st-tgds resulting in target redundancy we take the *coverage of conclusion atoms*, in terminology of [19]:

▶ **Example 20.** Consider the mapping Σ consisting of the following four st-tgds:

- $\tau_1 : S_1(x_1, x_2) \rightarrow \exists y_1 \exists y_2\ R(x_1, y_1) \wedge P(x_2, y_2, y_1)$
- $\tau_2 : S_2(x_1, x_2) \rightarrow \exists y\ R(x_1, y) \wedge P(x_2, x_1, y)$
- $\tau_3 : S_3(x_1, x_2) \rightarrow R(x_1, x_2)$
- $\tau_4 : S_4(x_1, x_2, x_3) \rightarrow P(x_2, x_1, x_3)$

Those tuples $(a, b) \in S_1$ which also occur in S_2, trigger creation of the facts we denote as $J_{ab} = \{R(a, Y_1), P(b, Y_2, Y_1)\}$ which do not belong to the core: indeed, τ_2 yields the instance $J'_{ab} = \{R(a, Y'), P(b, a, Y')\}$, onto which J_{ab} is mapped by a homomorphism $\{Y_1 \rightarrow Y'_1, Y_2 \rightarrow a\}$. We say, that τ_1 is *covered* by τ_2. Similarly, both τ_1 and τ_2 are covered by a pair of dependencies $\{\tau_3, \tau_4\}$. To see this, consider a chase of an instance $I'' = \{S_1(a, b), S_2(a, b), S_3(a, c), S_4(a, b, c)\}$. In addition to the facts of J_{ab} and J'_{ab}, $chase(I'', \Sigma)$ contains the facts $R(a, c)$ and $P(b, a, c)$, onto which J'_{ab} can be mapped with the homomorphism $\{Y' \rightarrow c\}$ and J_{ab} with $\{Y_1 \rightarrow c, Y_2 \rightarrow a\}$. ◀

The goal of dependency rewriting is to discover potential coverages by means of *static analysis* of mappings: that is, analysis performed at design time and valid for arbitrary inputs. To this end, coverages are formalized as relationships between dependencies rather than facts in possible target instances.

▶ **Definition 21.** Let $\psi(\vec{x}, \vec{y})$ be a conclusion of a tgd τ with the $\forall$-variables $\vec{x}$ and $\exists$-variables $\vec{y}$. We say that τ is *covered* by the tgds with conclusions $\psi_1(\vec{x}_1, \vec{y}_1), \ldots, \psi_k(\vec{x}_k, \vec{y}_k)$, if there exists a unification θ for $\forall$-variables $\vec{x}$, $\vec{x}_1$, …, $\vec{x}_k$, and a substitution λ for $\vec{y}$, such that $\psi(\vec{x}\theta, \vec{x}\lambda)$ is a subformula of $\phi_0(\vec{x}, \vec{y}_0) \wedge \bigwedge_{1 \leq i \leq k} \phi(\vec{x}_i\theta, \vec{y}_i)$, where $\psi_0(\vec{x}, \vec{y}_0)$ is a subformula of ψ with $\vec{y}_0 \subset \vec{y}$. Moreover, $\forall i\ 1 \leq i \leq k\ (\vec{x}_i \cup \vec{y}_i) \cap range(\lambda) \neq \emptyset$ must hold. If also $(\vec{x}_0 \cup \vec{y}_0) \cap range(\lambda) \neq \emptyset$ holds, the coverage is called *partial* (some atoms of ψ are mapped onto other atoms of ψ), otherwise, the coverage is *total*.

Example 20 illustrates the total coverage. As Mecca et al. point out, for tgds without self-joins in the conclusions, only this type of coverages is possible. To address such cases, the antecedent of each tgd τ must be taken in conjunction with the negated antecedents of tgds that cover τ.

▶ **Example 22.** Generation of redundant facts by the tgd τ_1 from Example 20 can be prevented by the following rewriting:

1. $S_1(x_1, x_2) \wedge \neg S_2(x_1, x_2) \wedge \neg S_3(x_1, x_2) \wedge \neg(\exists x_3 S_4(x_1, x_2, x_3)) \rightarrow$
$\exists y_1 \exists y_2\ R(x_1, y_1) \wedge P(x_2, y_2, y_1)$
2. $S_1(x_1, x_2) \wedge S_3(x_1, x_2) \wedge \neg(\exists x_3 S_4(x_1, x_2, x_3)) \rightarrow \exists y_1 \exists y_2\ P(x_2, y_2, y_1)$
3. $S_1(x_1, x_2) \wedge S_4(x_1, x_2, x_3) \wedge \neg S_3(x_1, x_2) \rightarrow \exists y_1\ R(x_1, y_1)$ ◀

For tgds with self-joins in the conclusions also the partial coverages, as in the Example 19, have to be taken into account. A solution of Mecca et al. [19] uses *atom labeling* as a starting point for enumeration of partial coverages:

▶ **Example 23.** The st-tgds from Example 19 can be labeled as follows:

- $\tau_1^*\colon S(a,b,c,d) \to \exists y_1\exists y_2\exists y_3\exists y_4\exists y_5$ $R^1(y_5,b,y_1,y_2,a)$ $\wedge R^2(y_5,c,y_3,y_4,a)$ $\wedge R^3(d,c,y_3,y_4,b)\,)$
- $\tau_2^*\colon S(e,f,g,h) \to \exists z_1\exists z_2\exists z_3\exists z_4\exists z_5$ $R^4(d,e,z_1,f)$ $\wedge R^5(z_5,e,e,z_1,e)$ $\wedge R^6(z_5,g,z_2,z_3,z_4)\,)$

A possible partial coverage of τ_1^*, enabled by the unification $\theta = \{b \to c\}$ of $\forall$-variables in τ^*, is given by a substitution $\{y_1 \to y_3, y_2 \to y_4\}$ on $\exists$-variables, sending R^1 onto R^2. ◀

Coverages of the tgd $\tau\colon \phi(\vec{x}) \to \psi(\vec{x},\vec{y})$ are represented by conjunctive formulas called *expansions*, of the form $\chi_i \wedge \psi_i$. Here, χ_i contains atoms that cover ψ, and ψ_i consists of ψ together with equalities $\mathcal{E}_i$ such that there exists a substitution λ for $\vec{y}$ that turns it into a subformula of χ_i, provided that the universal variables are unified according to $\mathcal{E}_i$.

▶ **Example 24.** The dependency τ_1^* from Example 19 gives rise to the following expansions (among others):

- $e_{23}\colon R^2(y_5,c,y_3,y_4,a) \wedge R^3(d,c,y_3,y_4,b) \wedge (R^1(y_5,b,y_1,y_2,a) \wedge b = c)$
- $e_{44}\colon R^4(h,e,e,z_1,f) \wedge R^4(h',e',e',z_1,f') \wedge h = h' \wedge$
  $(R^1(y_5,b,y_1,y_2,a) \wedge R^2(y_5,c,y_3,y_4,a) \wedge R^3(d,c,y_3,y_4,b) \wedge$
  $e = b \wedge f = a \wedge e' = c \wedge f' = a \wedge h' = d \wedge e' = c \wedge f' = b)$

The expansion formulas start with a covering part χ followed by the covered part in parenthesis, consisting of the covered atoms ψ and a set $\mathcal{E}$ of equalities. The expansion e_{23} is taken from Example 23 while e_{44} shows that two copies of τ_2^* provide a total coverage for τ_1^*. ◀

For a tgd τ with a conclusion ψ the coverages can be found by exhaustively enumerating all mappings of ψ onto the multisets of tgd conclusions. Yet this alone does not bring us to the goal of preventing redundant facts: While an expansion $\chi_i \wedge \psi_i$ indicates that some atoms of ψ should not be instantiated because of the atoms in χ_i, should the atoms of χ_i be instantiated? If an atom R^k in χ_i is covered by the atom R^l in some tgd conclusion, there will be also an expansion of τ using R^l instead of R^k. Hence, avoiding redundancy comes down to selecting the "safest" coverage at execution time. Mecca et al. distinguishes two orders on expansions, one according to the size of a covering conjunction χ and another favoring coverages with fewer existential variables: for example, a coverage of τ_1 with $\{\tau_3,\tau_4\}$ in Example 20 is safer than the coverage with τ_2, since the former two tgds have fewer existential variables. We use an informal order "safer" for both cases, leaving the exact details to [19].

The core computation in [19] is then implemented as a two-stage data exchange. The first stage uses a target schema $\mathbf{T}'$, obtained from $\mathbf{T}$ by taking the labeled atoms in Σ as new distinct relation names (For instance, the $\mathbf{S} \to \mathbf{T}'$ exchange with two tgds of Example 19 can use the labeled tgds of Example 23). The second phase transfers the data from $\mathbf{T}'$ to $\mathbf{T}$, ruled by the set Σ' of dependencies obtained from expansions as follows:

- If expansion $e\colon \chi \wedge \psi$ is most safe, a full tgd $\tau_e\colon \chi \wedge \psi \to \chi^{\neg *}$ is added to Σ', where $^{\neg *}$ denotes elimination of labels.
- Otherwise, τ_e has the form $\chi \wedge \psi \wedge \neg(\bigwedge_j e_j) \to \chi^{\neg *}$ where j ranges over expansions which are safer than e.

The combination of expansions in the latter case is quite similar to the way tgd antecedents in the Example 22 were obtained. It ensures that the safest possible coverage is taken into account when the tgd is applied.

▶ **Example 25.** The expansion e_{44} is safer than e_{23}. Hence, the antecedent of the tgd, obtained from e_{23} will contain the following conjuncts:

$$\begin{aligned} e_{23}^{rew}: & R^2(y_5,c,y_3,y_4,a) \wedge R^3(d,c,y_3,y_4,b) \wedge (R^1(y_5,b,y_1,y_2,a) \wedge b = c) \\ & \neg\big(R^4(h,e,e,z_1,f) \wedge R^4(h',e',e',z_1',f') \wedge h = h' \wedge \\ & \quad \big(R^1(y_5',b',y_1',y_2',a') \wedge R^2(y_5',c',y_3',y_4',a') \wedge R^3(d',c',y_3',y_4',b') \wedge \\ & \qquad e = b' \wedge f = a' \wedge e' = c' \wedge f' = a' \wedge h' = d' \wedge f' = b'\big) \\ & \quad \wedge\, c = e \wedge a = f \wedge d = h' \wedge c = e' \wedge b = f' \ \big) \end{aligned}$$

The corresponding conclusion of the tgd is $\exists y_3 \exists y_4 \exists y_5\ R(y_5,c,y_3,y_4,a) \wedge R(d,c,y_3,y_4,b)$. ◀

However, two further problems with isomorphic fact blocks are yet to be addressed: based on expansions which are equally safe, distinct $\mathbf{T}' \to \mathbf{T}$ tgds with isomorphic conclusions can be produced in Σ'. The second problem is concerned with a particular type of tgds:

▶ **Example 26.** Consider a tgd $S(x_1,x_2) \to \exists y\ R(x_1,y) \wedge R(x_2,y)$. Given a "reflexive" source $\{S(1,2), S(2,1)\}$, it yields a target instance $\{R(1,Y_1), R(2,Y_1), R(2,Y_2), R(1,Y_2)\}$ with two cores. Such tgd is said to have a conclusion with a *non-trivial automorphism*: indeed, under the unification $x_1 \to x_2$, there is a renaming of $\exists$-variables that map the first conclusion atom on the second one and vice versa. ◀

Core schema mappings address both issues using a special skolemization strategy, in the case of tgds with non-trivial automorphisms in the conclusion also involving interpreted functions *sort*. The Skolem terms that replace $\exists$-variables are strings encoding the structure of the fact block, instantiated by applications of the tgd (This technique assumes that the dependencies are *normalized* as described in Section 4.4):

1. All facts in the block ($\exists$-variables omitted). In case of fact blocks with non-trivial automorphisms, the list of facts is sorted before producing the Skolem string.
2. Joins between nulls.
3. A self-reference to the null represented by the Skolem string, in the fact block.

In this way, two variables will be instantiated with the same Skolem terms if and only if they correspond to the respective positions in isomorphic fact blocks.

▶ **Example 27.** The $\exists$-variable y in the tgd with non-trivial automorphism from Example 26 is skolemized with a string of the following pattern:

$$sort(R[A:x_0], R[A:x_1]);\ j:[R.B = R.B];\ v:j$$

The first component lists two facts in the block together with their $\forall$-variables. The prefix *sort* indicates that actual values of x_0, x_1 must be sorted before composing the string. The second component, prefixed with j, denotes the join between the two facts, while the last component $v:j$ associates the Skolem string to the positions participating in the join. Taking the source instance of Example 26, both facts $S(1,2)$ and $S(2,1)$ generate the Skolem string `'R[A:1] R[A:2]; j:[R.B=R.B]; v:j'`. ◀

To summarize, core computation is performed by chasing the skolemized mappings, with FO antecedents and interpreted Skolem functions. The two-phase data exchange via the intermediate schema $\mathbf{T}'$ is avoided in practice by *rewriting expansions over the source schema* [19]. In the next section, we will consider another algorithm for direct core computation in the absence of target constraints.

5.1.2 Laconic schema mappings

A different approach for direct core computation, named Laconic schema mappings has been developed by ten Cate, Chiticariu, Kolaitis and Tan [22].

▶ **Definition 28** (Laconicity)**.** A mapping $\mathcal{M}$ is *Laconic* if for each source instance I, the canonical universal solution for I under $\mathcal{M}$ is a core.

A favorable property of Laconic mappings is that they allow core computation by means of standard SQL queries, without any procedural extensions, e.g., for sorting the arguments of Skolem terms. Besides the algorithm itself, the authors provide a number of optimality results for their SQL encoding. These results take advantage of an abstract representation of skolemized mappings, in which every k-ary target relation $R \in \mathbf{T}$ has a form:

$$R \;:=\; \{(t_1(\vec{x}), \dots, t_k(\vec{x})) \mid \phi(\vec{x})\} \;\cup\; \cdots \;\cup\; \{(t'_1(\vec{x}'), \dots, t'_k(\vec{x})) \mid \phi'(\vec{x}')\} \tag{1}$$

Here, $t_1, \dots, t_k, \dots, t'_1, \dots, t'_k$ are terms and $\phi, \dots, \phi'$ are first-order queries over the source schema. Since FO queries correspond to SQL queries, one can easily use a relational DBMS in order to compute the tuples in the relation R.

▶ **Definition 29** (L-term interpretation)**.** Let $\mathcal{L}$ be any query language. An *$\mathcal{L}$-term interpretation* Π is a map assigning to each k-ary relation symbol $R \in \mathbf{T}$ a union of expressions of the form (1) where $t_1, \dots, t_k \in \mathit{Terms}[\vec{x}]$ and $\phi(\vec{x})$ is an $\mathcal{L}$-query over $\mathbf{S}$.

Here, $\mathit{Terms}[\vec{x}]$ is a set of terms built using the set of constants $\vec{x}$ and functional symbols from some countably infinite vocabulary. As usual, the proper terms $\mathit{Terms}[\vec{x}] \setminus \vec{x}$ are considered as nulls while the members of $\vec{x}$ are constants. The goal of direct core computation is then to find an $\mathcal{L}$-term interpretation of a core universal solution for a given schema mapping $\mathcal{M}$. Moreover, to reduce the complexity of data exchange, it is desirable to use the least expressive language $\mathcal{L}$.

Given a Laconic mapping with $\mathcal{L}$ st-tgds, it is straightforward to obtain a $\mathcal{L}$-term interpretation, whose target relations are core universal solutions: it suffices to apply the standard Skolemization, and use the antecedent of a st-tgd as a precondition of conclusion atom. The main result of ten Cate et al. in [22] is that every mapping based on FO$^<$ st-tgds can be converted into a logically equivalent Laconic mapping, also consisting of FO$^<$ st-tgds. They also show optimality of this language, even for input mappings consisting of *CQ* st tgds: see Section 5.2. Hence, from now on we will focus on obtaining the Laconic mappings, rather than term interpretations.

Unlike Core schema mappings, which adapts each individual st-tgd to the case when it fires along with other dependencies, ten Cate et al. follow a top-down approach: taking a global perspective on a given mapping, they create its Laconic version from scratch.

The algorithm builds upon the observation exploited in Section 4.1: Namely, that in the absence on target constraints, the size of fact blocks in the target instance is bounded by the maximal number of conclusion atoms in the tgds. Hence, one can enumerate all possible fact block patterns (up to renaming of nulls and unifications of constants) in the core universal solution. This is captured by the notion of *fact block type* (*f-block type* for short):

▶ **Definition 30.** An f-block type $t(\vec{x}; \vec{y})$ is a set of atomic formulas in two disjoint sets of variables $\vec{x}$ and $\vec{y}$, respectively called *c-variables* and *n-variables*. A fact block B is said to have the type $t(\vec{x}; \vec{y})$, if it can be obtained by instantiating c-variables of t with constants, and replacing each n-variable with a distinct null. Let a fact block $B = t(\vec{a}, \vec{Y})$ be such an instantiation. We say that t is *realized* at $\vec{a}$.

The algorithm proceeds in four steps which are outlined in the subsequent paragraphs.

1. Identify all f-block types that can be realized in a core universal solution under $\mathcal{M}$, for any source instance I.
2. For each f-block type $t(\vec{x},\vec{y})$, construct a query $\mathit{precon}_t(\vec{x})$ over the source schema, retrieving all assignments $\vec{a}$ for $\vec{x}$, such that t is realized in $\mathit{core}(I,\Sigma)$ at $\vec{a}$. Such query is called a *precondition* of t.
3. For each f-block type t' with non-trivial automorphisms, strengthen $\mathit{precon}_{t'}(\vec{x})$ to ensure that if two assignments $\vec{a}_1,\vec{a}_2$ for $\vec{x}$ are distinct, the corresponding fact blocks $t'(\vec{a}_1,\vec{N}_1)$ and $t'(\vec{a}_2,\vec{N}_2)$ are not isomorphic. Such additional constraints for the precondition of t' are called *side-conditions* denoted as $\mathit{sidecon}_t(\vec{x})$.
4. For each f-block type $t(\vec{x},\vec{y})$, produce a tgd $\mathit{precon}_t(\vec{x}) \wedge \mathit{sidecon}_t(\vec{x}) \rightarrow \exists\vec{y}\ t(\vec{x},\vec{y})$.

Generating f-block types for $\mathcal{M}$ amounts to examination of tgd conclusions in $\mathcal{M}$, and taking certain subsets of them. Importantly, f-block types are (1) connected w.r.t. to n-variables, and (2) cores — if considered as instances where c-variables are constants and n-variables are nulls, — and (3) cannot be obtained from any other f-block type by renaming c- or n-variables. The result of this step is the set $\textsc{Types}_{\mathcal{M}}$ of f-block types *generated by* $\mathcal{M}$.

Finding the preconditions. This is the crux of the algorithm, for which one can give an intuition as follows. Consider an f-block type t as a query $q_t(\vec{x}) \leftarrow \exists\vec{y}\ t(\vec{x},\vec{y})$. For all homomorphic images of t in a canonical target instance J, q_t selects satisfying assignments for the c-variables $\vec{x}$. Suppose that we manage to restrict $q_t(\vec{x})$ in order to select only the assignments for $\vec{x}$ at *which t is realized*, *in the core* of J. By Definition 30, we have to filter out every assignment $\vec{a}$ for $\vec{x}$, such that

1. $\vec{a}$ contains nulls, or
2. $\exists\vec{b} \in \mathit{dom}(J)\colon J \models t(\vec{a},\vec{b})$ and for some $i \leq |\vec{b}|$, $b_i \in \mathit{const}(J)$, or
3. $\exists\vec{N} \in \mathit{nulls}(J)\colon J \models t(\vec{a},\vec{N})$ and for some $i,j \leq |\vec{N}|$ $N_i = N_j$ holds, or
4. $\exists\vec{N}_1 \in \mathit{nulls}(J)\colon J \models t(\vec{a},\vec{N}_1)$ and for some fact block $B \subseteq J$, $\vec{N}_1 \subset \mathit{nulls}(B)$: this case prohibits mapping of $t(\vec{x},\vec{y})$ into a bigger fact block of J.

The first item is addressed by considering only the certain answers of q_t: we can be sure that all certain answers belong to the core of J. Moreover, we can immediately rewrite $\mathit{certain}(q_t(\vec{x}))$ over the source schema. This rewriting, denoted as $\mathit{certain}_{\mathcal{M}}(\exists\vec{y}\ t)(\vec{x})$, uses well-known techniques and will be discussed shortly. It remains to address the items (3) and (4): so far, the assignments of $\vec{y}$ of are not restricted in any way.

Concerning (3), suppose that we want to query for all images of t in which some $y_i \in \vec{y}$ is mapped onto a constant in the core of J. It suffices to bring y_i into the set of c-variables of t, and ask for certain answers for $\exists y_{-i}\ t(\vec{x}y_i;\vec{y}_{-i})$, where $\vec{y}_{-i}$ is $\vec{y}$ without elements equal to y_i. Then the assignments for $\vec{x}$ can be projected and excluded from the answers to $\mathit{certain}_{\mathcal{M}}(\exists\vec{y}\ t)(\vec{x})$. The same is done for each n-variable of t, and a similar approach allows to handle the case (4). A corresponding query is defined as an *approximated precondition* $\mathit{precon}'_t(\vec{x})$ of the form

$$\mathit{certain}_{\mathcal{M}}(\exists\vec{y}\bigwedge t)(\vec{x}) \quad\wedge\quad \bigwedge_i \neg\exists x'\ \mathit{certain}_{\mathcal{M}}(\exists\vec{y}_{-i}\bigwedge t[y_i/x'])(\vec{x},x')$$
$$\wedge \quad \bigwedge_{i\neq j} \neg\mathit{certain}_{\mathcal{M}}(\exists\vec{y}_{-i}\bigwedge t[y_i/y_j])(\vec{x})$$

Procedure ConvertToLaconic

Input: Mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ consisting of FO$^<$ st-tgds
Output: Laconic mapping $\mathcal{M}' \equiv \mathcal{M}$ with the set of FO$^<$ st-tgds Σ'

```
(1) Set $\Sigma' := \emptyset$
(2) Generate $\textsc{Types}_{\mathcal{M}}$
(3) for each $t(\vec{x};\vec{y}) \in \textsc{Types}_{\mathcal{M}}$ do
(4)     Compute preconditions $precon_t(\vec{x})$
(5)     Compute side-condition $sidecon_t(\vec{x})$
(6)     Add the following FO$^<$ st-tgd to $\Sigma'$:
(7)         $\forall \vec{x}\,(precon_t(\vec{x}) \wedge sidecon_t(\vec{x}) \rightarrow \exists \vec{y} \bigwedge t(\vec{x};\vec{y}))$
(8) return $(\mathbf{S}, \mathbf{T}, \Sigma')$
```

To handle (5), $precon'_t$ is combined with negated approximated preconditions $precon'_{t'}$ for each f-block type t', on which t can be mapped by a non-surjective homomorphism:

$$precon_t(\vec{x}) \quad = \quad precon'_t(\vec{x}) \quad \wedge \bigwedge_{\substack{t'(\vec{x}';\vec{y}') \,\in\, \textsc{Types}_{\mathcal{M}} \\ h\,:\,t(\vec{x};\vec{y}) \,\rightarrow\, t'(\vec{x}';\vec{y}')\text{ non-surjective}}} \neg\exists \vec{x}' \Big(\bigwedge_i (x_i = h(x_i)) \quad \wedge \quad precon'_{p'}(\vec{x}') \Big)$$

As pointed out in [22], one of possibilities for rewriting $t(\vec{x};\vec{y})$ over the source schema is splitting up $\mathcal{M}$ into a composition $\mathcal{M}_1 \circ \mathcal{M}_2$, where $\mathcal{M}_1$ consists of full st-tgds and tgds in $\mathcal{M}_2$ have single atoms over some intermediary schema in the antecedents; such tgds can be rewritten using an algorithm like MiniCon [21] (cf. Section 3.3 in Chapter 5), after which the unfolding of atoms according to $\mathcal{M}_1$ would give a desired rewriting.

Adding side-conditions. A special tgd from Example 26 considered in the Section 5.1.1 has to be taken care of in the context of Laconic mappings as well. Unlike Core schema mappings, a non-standard skolemization is not necessary now: the preconditions are enhanced with side-conditions which rely on inequalities and are defined over the source schema. This can be seen on example:

▶ **Example 31.** The st-tgd $S(x_1, x_2) \rightarrow \exists y\ R(x_1, y) \wedge R(x_2, y)$ from Example 26 is rewritten as $(S(x_1, x_2) \vee S(x_2, x_1)) \wedge x_1 \leq x_2 \rightarrow \exists y\ R(x_1, y) \wedge R(x_2, y)$. It is easy to see that on a problematic source instance $\{S(1, 2), S(2, 1)\}$ the rewritten tgd is triggered only once. ◀

Side-conditions are only introduced for f-block types with non-trivial automorphisms. In particular, they are not used for mappings in which tgds have no self-joins in the conclusion.

Generating the st-tgds. Given the set $\textsc{Types}_{\mathcal{M}}$ of f-block types of $\mathcal{M}$, together with their preconditions and side-conditions, generation of the new st-tgds for the Laconic version of $\mathcal{M}$ comes down to combining the f-block type and its preconditions resp. side-conditions in a single tgd, as specified in the procedure ConvertToLaconic.

5.1.3 Discussion

We have described two approaches to direct core computation, which use the same language elements: st-tgds with antecedents FO and linear order on the source constants. Despite of these similarities, these mappings are obtained in quite different ways: Core schema mappings are built bottom-up, adapting existing st-tgds to take care of other dependencies and, whereas Laconic schema mappings are constructed top-town, by using a given schema mapping as a black box and applying query rewriting algorithms like MiniCon [21].

The algorithm of Mecca et al. is currently the only known implementation of direct core computation. This can be hardly overestimated, especially taking into account the promising performance reports, with millions of tuples in the source instance processed in a few minutes (More detailed discussion of experimental results is postponed until Section 6). At the same time, ten Cate et al. provide important optimality results, justifying the language constructs found both in Laconic mappings and in Core mappings. These results will be the subject of the next section.

5.2 Complexity and expressiveness

The first theoretical result of ten Cate et al. addresses complexity of a test for laconicity:

▶ **Theorem 32.** *[22] Testing laconicity of schema mappings specified by FO st-tgds is undecidable. It is **coNP**-hard already for schema mappings specified by LAV st-tgds.*

Producing a Laconic or Core schema mapping based on a set of st-tgds can result in an exponential increase in the number of dependencies. This is not a coincidence: ten Cate et al. show, that this cannot be avoided:

▶ **Theorem 33.** *[22] There is a sequence of schema mappings $\mathcal{M}_1, \mathcal{M}_2, \ldots$ specified by LAV st-tgds such that the specification of each $\mathcal{M}_k$ is of length $O(k)$, and such that every Laconic schema mapping logically equivalent to $\mathcal{M}_k$ specified by* $\mathrm{FO}^<$ *st-tgds contains at least 2^k many* $\mathrm{FO}^<$ *st-tgds.*

Concering the optimality of $\mathrm{FO}^<$ as a language used in the antecedents of the Laconic st-tgds, the following results show, that such neither linear order on constants nor negation can be avoided.

▶ **Theorem 34.** *[22] Consider the schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ where $\mathbf{S} = \{S\}$, $\mathbf{T} = \{R\}$ and Σ consists of a single LAV st-tgd $S(x_1, x_2) \to \exists y\ S(x_1, y) \wedge S(y, x_2)$. No* FO*-term interpretation yields, for each source instance I, the core universal solution of I w.r.t. $\mathcal{M}$.*

▶ **Theorem 35.** *[22] There exists schema mapping $\mathcal{M}$ with dependencies given by st-tgds, such that no $UCQ^<$-term interpretation can compute the core universal solution for each source instance.*

Proof hint. Any mapping with *coverage* between st-tgds, like that in Example 20, can be shown to require negation for achieving laconicity. ◀

The language ingredients used by Core schema mappings in Section 5.1.1 are fully consistent with the results cited above: The interpreted *sort* function used to produce Skolem strings (see Example 27) assume the linear order on the source constants, and negation in the antecedents is used to combine expansions (Example 25).

5.3 Target constraints

Two direct core computation algorithms presented in the previous chapter only dealt with the mappings without target constraints. This is a major restriction in comparison to the post-processing approach. However, as ten Cate et al. show [22], there is a good reason for that: for a mapping with full target tgds, there is no Laconic version based on $\mathrm{FO}^<$ st-tgds *and target tgds and egds.*

▶ **Theorem 36.** *[22] There is a schema mapping $\mathcal{M}$ specified by finitely many LAV st-tgds and full target tgds, for which there is no schema mapping $\mathcal{M}'$ specified by* $\mathrm{FO}^<$ *tgds, target tgds and target egds, such that for every source instance I, the canonical universal solution of I under M' is the core universal solution of I under $\mathcal{M}$.*

Proof idea. Let $\mathcal{M}$ be the schema mapping with $\mathbf{S} = \{S', S_1, S_2, S_3\}$, $\mathbf{T} = \{R', P_1, P_2, P_3, Q_1, Q_2, Q_3\}$ specified by four LAV s-t tgds and three full target tgds:

- $S(x_1, x_2) \to R(x_1, x_2)$
- $S_i(x) \to \exists y\ Q_i(y)$ for $i \in \{1, 2, 3\}$
- $R(x, y) \land R(y, z) \to R(x, z)$
- $R(x_1, x_1) \land Q_1(x_2) \to Q_3(x_2)$
- $R(x_1, x_1) \land Q_2(x_2) \to Q_3(x_2)$

For source instances I in which all source relations are non-empty, the core universal solution J will have the following shape: R is the transitive closure of S, and $Q_{1,2,3}$ are non-empty. Moreover, if S contains a cycle, then the core universal solution contains the facts $Q_1(N_1), Q_2(N_2)$ and $Q_3(N_1), Q_3(N_2)$ for distinct null values N_1, N_2. If S in I is acyclic, $Q_{1,2,3}$ each contain a single null, occurring exactly once in the core universal solution.

Suppose that a fact $Q_3(N')$ is present in the target instance. It is a part of the core universal solution if and only if the source relation S is acyclic. One can show, that the Laconic mapping must contain a dependency that fires on cyclic instances and not fires on acyclic ones, and that such behavior can be achieved neither by st-tgds (we cannot detect cycles with a $\mathrm{FO}^<$ antecedent) nor by monotone target dependencies. ◀

In [22], it is conjectured that the same inexpressibility result should hold for the mappings with target egds. Hence, the problem of direct core computation becomes highly non-trivial even in presence of restricted target constraints. However, Marnette, Mecca and Papotti give an experimental evidence based on the system +SPICY [18], that a *best-effort approach* via FO-term interpretations can tackle practically relevant mappings with *target functional dependencies* (FDs). We will outline their algorithm which we refer to as SPICY-FD in the rest of this section.

Recall the Rigidity Lemma from Section 4.2: let an egd equate the nulls X, Y from the domains of different blocks in the preuniversal instance J^{st} (the canonical universal solution with respect to the st-tgds Σ_{st} of the mapping), then the null resulting from this unification is rigid: e.g., assume that both X and Y have been replaced by the same term a in the canonical universal instance J, obtained by enforcing the target egds on J^{st}. Then, for any endomorphism e on J, $e(a) = a$ holds. One of the key ideas behind the SPICY-FD approach is reminiscent of this property:

1. Suppose that the mapping $\mathcal{M}$ whose set of dependencies Σ consists of st-tgds and target FDs is such that a FO-term interpretation for universal solutions under $\mathcal{M}$ exists. In [18], this interpretation is constructed in the form of skolemized FO st-tgds, called a *FO implementation* $\mathcal{R}_\mathcal{M}$ of $\mathcal{M}$. $\mathcal{R}_\mathcal{M}$ is *sound and complete*, if for each source instance I, *chase*$(I, \mathcal{R}_\mathcal{M}) \models \Sigma$ iff *chase*(I, Σ) does not fail.
2. A sound and complete FO implementation $\mathcal{R}_\mathcal{M}$ correctly instantiates the $\exists$-variables that would be affected by FDs in the target chase. Some of them are (rigid) nulls and some are constants: we refer to them as to *rigid terms*. $\mathcal{R}_\mathcal{M}$ can be rewritten in a way to store rigid terms in an auxiliary schema $\mathbf{F}$ with a relation F_i per each target FD ϵ_i in $\mathcal{M}$.
3. For core computation, rigid nulls are indistinguishable from constants. Marnette et al. notice that independently of the source instance I, each $\exists$-variable in the st-tgds Σ_{st} of $\mathcal{M}$ is instantiated either by rigid terms or by non-rigid nulls (see Example 39 below). The

former are *converted into* $\forall$*-variables*, stemming from the relations of the auxiliary schema $\mathbf{F}$, which are added to the antecedent of the st-tgd. The result of such transformation is then made Laconic (or converted to a Core mapping), giving a set of FO$^<$ st-tgds Σ_c.

4. The core universal solution is obtained by a sequence of chases with $\mathcal{R}_\mathcal{M}$ followed by Σ_c.

Constructing FO implementation $\mathcal{R}_\mathcal{M}$ of $\mathcal{M}$. Let Σ be a set of st-tgds and target FDs of $\mathcal{M}$. For each source instance I, $chase(I, \mathcal{R}_\mathcal{M}) \models \Sigma$ must hold, in which case $\mathcal{R}_\mathcal{M}$ is called *sound and complete* implementation of $\mathcal{M}$ with skolemized FO st-tgds.

The idea of this step is similar to that behind Core schema mappings: each st-tgd is rewritten to anticipate the effect of target FDs. We illustrate it by rewriting the following mapping over the source schema $\mathbf{S} = \{S_{1,2}\}$ and the target schema $\mathbf{T} = \{P, R, Q\}$ where R has attributes ABC, with a functional dependency ϵ: $R\langle A \to C\rangle$ defined. Besides ϵ, the mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, Σ contains two st-tgds:

- $\sigma_1\colon S_1(x_1, x_2) \to \exists y\ P(y) \wedge R(x_1, x_2, y)$
- $\sigma_2\colon S_2(x_1, x_2) \to \exists y\ R(x_1, x_2, y) \wedge Q(y)$

It is easy to see that on each pair of source facts $I = \{S_1(a, b), S_2(a, b')\}$ the single block $J = \{P(N), R(a, b, N), R(a, b', N), Q(N)\}$ is introduced in the target instance, due to the effect of the FD on R. This behavior can be captured by the st-tgd:

- $\sigma_{12}\colon\ S_1(x_1, x_2) \wedge S_2(x_1, x_3) \to \exists y\ P(y) \wedge R(x_1, x_2, y) \wedge R(x_1, x_3, y) \wedge Q(y)$

Such combined dependencies are called *overlap st-tgds* Σ_{st}^{ovl}. Two issues arise: firstly, the process of constructing Σ_{st}^{ovl} can fail to terminate. A solution is to abort with failure after certain limit of number of steps has been reached.

Secondly, the set $\Sigma_{st} \cup \Sigma_{st}^{ovl} = \{\sigma_{1,2,12}\}$ cannot yet be seen as an implementation of $\mathcal{M}$: oblivious chase of I yields the instance $J \cup J_1 \cup J_2 \not\models \epsilon$ where $J_1 = \{P(N_1),\ R(a, b, N_1)\}$ instantiates the conclusion of σ_1 and $J_2 = \{R(a, b', N_2),\ Q(N_2)\}$ instantiates that of σ_2: these dependencies fire whenever the overlap st-tgd σ_{12} does. To suppress redundant facts, the antecedents of σ_1 and σ_2 are rewritten respectively as $S_1(x_1, x_2) \wedge \neg(\exists x_3\ S_2(x_1, x_3))$ and $S_2(x_1, x_2) \wedge \neg(\exists x_3\ S_1(x_1, x_3))$. Procedure responsible for such rewriting is called AddNeg.

However, these measures still do not result in a desired implementation of $\mathcal{M}$ with source-to-target dependencies: an instance $I' = \{S_1(a, b), S_1(a, b')\}$ is a simple counter-example. The oblivious chase of I' with AddNeg$(\Sigma_{st} \cup \Sigma_{st}^{ovl})$ creates a target instance $J' = \{P(N_1), R(a, b, N_1),\ P(N_2), R(a, b', N_2)\} \not\models \epsilon$. This issue is solved by choosing a *non-standard skolemization strategy*: in our example, the $\exists$-variable y in all three st-tgds is substituted by a Skolem term with a single attribute x_1 (Standard skolemization would yield terms with attributes x_1, x_2 for $\sigma_{1,2}$, and x_1, x_2, x_3 in case of σ_{12}). A key here is finding a minimal set of attributes determining the rigid null: In general, there might be several FDs affecting it. The minimal set (called *determination* in [18]) must be unique, otherwise the procedure Skolemize aborts with failure.

In overall, the mapping Skolemize $\big($AddNeg$(\Sigma_{st} \cup \Sigma_{st}^{ovl})\big)$ is proven to be a sound and complete FO implementation of $\mathcal{M}$, provided that no failure occurs while creating Σ_{st}^{ovl} or performing the skolemization.

Eliminating rigid $\exists$-variables. We start by adorning each conclusion atom in st-tgds with a unique label, as it was done in Section 5.1.1.

▶ **Definition 37.** *Position* in a conclusion atom $R^l(z_1, ...z_k)$ of a st-tgd τ is a pair (l, i), for $i \leq k$. Let τ be applied in the chase, generating a fact $R(a_1, \ldots a_k)$ in the target instance.

Procedure SPICY-FD

```
Input:  Schema mapping $\mathcal{M} = \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t$ where $\Sigma_t$ as set of FDs
Output: Mappings $(\mathcal{R}_F, \mathcal{R}_C)$.
        /* The core universal solution for $I$ can be found as $chase(I \cup chase(I, \mathcal{R}_F), \mathcal{R}_C)$ */
(1)  Generate $\Sigma_{st}^{ovl}$ and Set $\mathcal{R} :=$ SKOLEMIZE $(\text{ADDNEG}(\Sigma_{st} \cup \Sigma_{st}^{ovl}))$ or fail
(2)  Let $\mathbf{F}$ be the schema $\{F_\epsilon \mid \epsilon\colon R\langle i_1 \dots i_m, j\rangle \in \Sigma_t\}$, $F_\epsilon$ fresh symbol of arity $m+1$
(3)  Let $\mathcal{R}_F = \emptyset$
(4)  for each $\phi(\vec{x}) \to \psi(\vec{x})$ in $\mathcal{R}$, $\epsilon$: $\langle i_1 \dots i_m, j\rangle \in \Sigma_t$ and $R(t_1, \dots t_n)$ in $\psi$
(5)      Set $\mathcal{R}_F := \mathcal{R}_F \cup \{\phi(\vec{x}) \to F_\epsilon(t_{i_1}, \dots, t_{i_m}, t_m)\}$
     /* Eliminate rigid ∃-variables */
(6)  Set $\Sigma_{st}^F := \Sigma_{st} \cup \Sigma_{st}^{ovl}$
(7)  while fixpoint is reached do
(8)      for each $\tau\colon \phi(\vec{x}) \to \exists y, \vec{z}\ \psi(\vec{x}, y, \vec{z})$ in $\Sigma'_{st}$ and $\epsilon : R\langle i_1, \dots i_m \to j\rangle \in \Sigma_t$
(9)              and each atom $R(t_1, \dots t_n)$ in $\psi$ such that $t_j = y$ and $\{t_{i_1}, \dots, t_{i_m}\} \in \vec{x}$
(10)                 Replace $\tau$ in $\Sigma_{st}^F$ by $\forall\vec{x}\forall y\,(F(t_{i_1}, \dots t_{i_m}, y) \wedge \phi(\vec{x}) \to \exists z\ \psi(\vec{x}, y, \vec{z}))$
     /* Apply algorithms from Section 5.1.1 or Section 5.1.2 */
(11) Convert $\Sigma_{st}^F$ into a Laconic or Core mapping $\mathcal{R}_C$
(12) return $(\mathcal{R}_F, \mathcal{R}_C)$
```

Positions $(l, 1), \dots (l, k)$ are said to be *instantiated* with the terms $a_1, \dots a_k$, respectively. A position (l, i) is called *rigid*, if for any source instance I, it is instantiated either with a constant or with a rigid null in $chase(I, \Sigma)$, and non-rigid otherwise.

It turns out, that each position can be uniquely classified as rigid or non-rigid, for arbitrary source instances:

▶ **Lemma 38.** *Let $\mathcal{M}$ be a mapping with an st-tgd τ. The position (l, j) of the conclusion atom $R^l(z_1, \dots z_j \dots z_k)$ in τ is rigid iff one of the following condition holds: (1) z_j is a ∀-variable, or (2) the positions $(l, i_1), \dots (l, i_m)$ are rigid and an FD $R\langle i_1 \dots i_m \to j\rangle$ is in $\mathcal{M}$, or (3) z_i is a ∃-variable occurring in a rigid position in the conclusion of τ.*

The first case of rigidity is trivial: the positions occupied by ∀-variable are instantiated by constants and thus are rigid. Concerning the inductive case, consider the following example (for brevity, we do not consider overlap st-tgds):

▶ **Example 39.** Consider a mapping with four st-tgds and a target FD:

- $\tau_1\colon S_1(x_1, x_2) \to \exists z\ R^1(x_1, x_2, z)$
- $\tau_3\colon S_3(x_1, x_2) \to Q^4(x_1, x_2, x_2)$
- $\tau_4\colon S_4(x) \to \exists y\ R^5(y, x, y)$
- $\tau_2\colon S_2(x_1, x_2, x_3) \to \exists y_1 \exists y_2\ R^2(x_1, y_1, y_2) \wedge Q^3(x_2, x_3, y_1)$
- $\epsilon_1\colon R\langle A, B \to C\rangle$

We assume that all target relations have attributes A, B, C. We will write $R^2.A$ to denote the position $(2, 1)$ in the conclusion of τ_2, occupied by a ∀-variable x_1. ∀-variables occur also at positions $R^1.AB$, $R^5.B$, $Q^3.AB$ and $Q^4.ABC$, rendering them all rigid. Also the position $R^1.C$ is rigid, since the attribute $R.C$ depends functionally on $R.AB$, and positions $R^1.AB$ are rigid. Indeed, let R^1 be instantiated as a fact $R(a_1, a_2, N)$ in the canonical universal solution U. If there exists an endomorphism for U that maps N onto some $c \neq N$, the fact $R(a_1, a_2, c)$ must be present in U. Thus $U \not\models \epsilon_1$, which is a contradiction.

The remaining positions $Q^3.C$, $R^5.AC$ and $R^2.BC$ are not rigid. As an example, consider a source instance $I = \{S_1(a,c),\ S_2(a,b,c),\ S_3(b,c)\}$. The canonical universal solution $J = \mathit{chase}(I,\Sigma) = \{R(a,c,Z),\ R(a,Y_1,Y_2),\ Q(b,c,Y_1),\ Q(b,c,c)\}$, with $\mathit{core}(J) = \{R(a,c,Z), Q(b,c,c)\}$ obtained by the endomorphism $\{Y_1 \to c, Y_2 \to Z\}$. Note that the facts $R(a,Y_1,Y_2)$ and $Q(b,c,Y_1)$ were generated by chasing τ_2: Y_2 instantiating the non-rigid position $R^2.C$ and Y_1 instantiating the non-rigid positions $R^2.B$ and $Q^3.C$. At the lines 7–10 of the procedure SPICY-FD, the rigid $\exists$-variables are transformed into $\forall$-variables in τ_1:

- τ_1': $S_1(x_1,x_2) \wedge F_1(x_1,x_2,z) \to R(x_1,x_2,z)$

Now, let Σ' be Σ extended with an FD ϵ_2: $Q\langle A \to C\rangle$. This makes position $Q^3.C$ rigid, by the same reason as $R^1.C$. In turn, also $R^2.B$ becomes rigid as sharing a $\exists$-variable with $Q^3.C$, and so is $R^2.C$. The procedure SPICY-FD now would also be able to rewrite τ_2:

- τ_2': $S_2(x_1,x_2,x_3) \wedge F_2(x_2,y_1) \wedge F_1(x_1,y_1,y_2) \to R(x_1,y_1,y_2) \wedge Q(x_2,x_3,y_1)$ ◀

Computing the core. The actual values for rigid nulls in relations F_i are provided by the FO implementation $\mathcal{R}_\mathcal{M}$ of $\mathcal{M}$. To this end, $\mathcal{R}_\mathcal{M}$ is rewritten to as the mapping $\mathcal{R}_F$ at the lines 3–5 of SPICY-FD, populating the relations F_i with the values instantiating the rigid nulls, and with the values, by which the nulls are determined. Lines 6–10 perform the elimination of rigid $\exists$-variables from $\Sigma_{st} \cup \Sigma_{st}^{ovl}$, resulting in the set of st-tgds Σ_{st}^F. Its Laconic version $\mathcal{R}_C$ is computed at line 11. Finally, a composition of $\mathcal{R}_F$ with $\mathcal{R}_C$ allows to produce a core universal solution for each source instance I.

6 Performance

Of the several presented core computation algorithms, only two have been actually implemented: a post-processing approach FINDCORE^E [12, 20] and the Core schema mappings, including the extension for target functional dependencies (the +SPICY system, [19, 18]). Both systems employ the database engines for performing the chase. In the latter case, this suffices to compute the core. In the post-processing case, searching for homomorphisms and extensions thereof is delegated to the DBMS, while the main cycle is driven by a Java program. The experimental evaluation allows to draw the following conclusions regarding practical feasibility of the core computation algorithms.

Post-processing approach: Despite polynomial data complexity, the most flexible algorithm based on FINDCORE only scales to with several thousands of nulls in the source database.

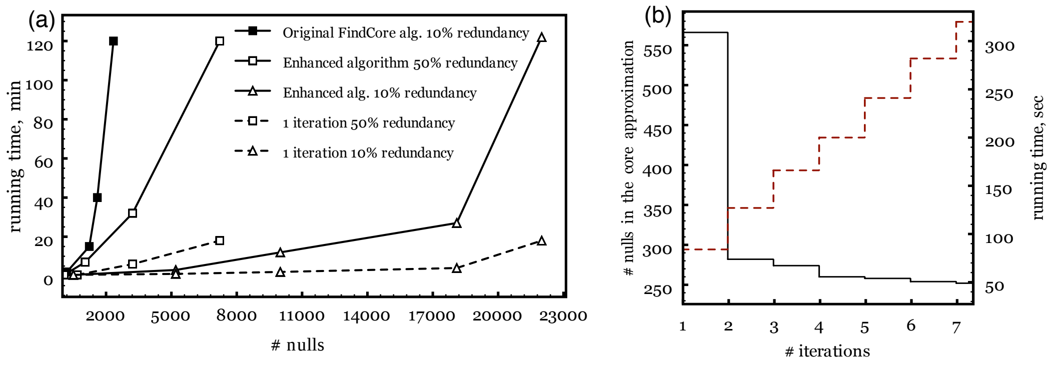
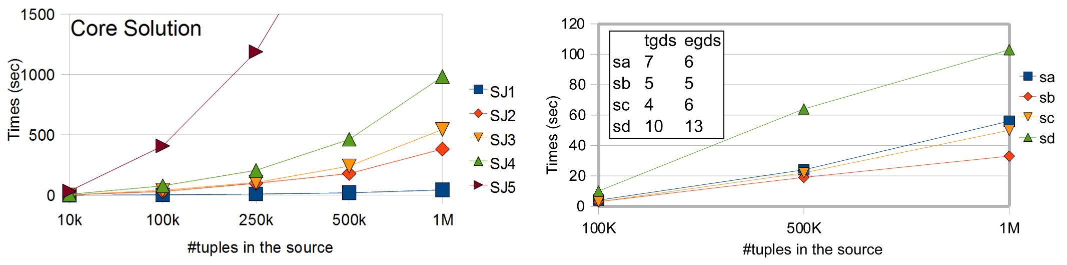


Figure 3 Performance (a) and the progress (b) of core computation [20].



	tgds	egds
sa	7	6
sb	5	5
sc	4	6
sd	10	13

Figure 4 Performance of direct core computation with +Spicy: no target constraints [19] (a) and target FDs [18] (b).

This is not completely unexpected, taking into account a quadratic number of iterations in the main cycle. As seen in Fig. 3(b), the core can be quite well approximated already by a single iteration, but much time has to be spent to eliminate few remaining nulls and to validate the minimality of the core.

Direct core computation approach of +Spicy, on the contrary, has proven to scale to source databases with millions of facts, even in the presence of target FDs. Figure 4 presents two charts adopted from [19] and [18] respectively, illustrating the performance of the two implementations. Fig. 4(a) shows performance charts for mappings with self-joins in the conclusions of st-tgds. The mapping 'SJ5' has been specially crafted to generate a rewriting with exponential number of dependencies. The chart in Fig. 4(b), produced with mappings with simpler st-tgds, which is compensated by adding target functional dependencies. It clearly demonstrates the robustness of the Spicy-FD procedure: the authors point out that the scenario 'sd' was specially designed to generate an exponential number of overlap st-tgds. A better performance of the core computation in presence of target egds is not surprising, taking into account the effect of rigid nulls, discussed in Sections 4.2 and 5.3.

7 Conclusion

We gave an overview of the algorithms for core computation in data exchange. They can be roughly divided into two groups: the post-processing algorithms, optimizing the canonical universal solution obtained by chasing a given set of dependencies, and direct computation, constructing the core as a result of the chase with the preprocessed dependencies. Both approaches provide polynomial data complexity of core computation. The advantage of post-processing is the support for expressive mappings, however no scalable implementation of this approach exists yet. In contrast, experiments with direct core computation have shown very encouraging performance results, but on rather restricted mappings. As shown in [22, 18], in presence of target dependencies it is often the case that no Laconic variant of the given mapping can be found. On the other hand, the best-effort approach of [18] can be used in many practical scenarios.

There is a need for further implementations of core computation in data exchange: so far, only a single scalable implementation (the +Spicy system by Mecca et al.[19, 18]) has been reported. For the post-processing approach, the optimization potential can be found in applying the decomposition-based homomorphism computation, adding the natural support of egds for skolemized mappings by combining the ideas of [16] and [20], and finding heuristics for approximation of the core. In the area of direct core computation, the algorithms supporting more expressive mappings can be considered as one of the primary goals. Furthermore, a combination of the two paradigms is conceivable, especially in the case of mappings with target egds for which no FO-term implementations can be found.

References

1 Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4):718–741, 1984.

2 Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

3 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

4 O.M. Duschka, M.R. Genesereth, and A.Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–74, 2000.

5 Ronald Fagin. Horn clauses and database dependencies. *J. ACM*, 29(4):952–985, 1982.

6 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89 – 124, 2005.

7 Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data exchange: getting to the core. *ACM Trans. Database Syst.*, 30(1):174–210, 2005.

8 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse data exchange: coping with nulls. In *Proc. of the 28th ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems (PODS'09)*, pages 23–32, 2009.

9 Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *Proc. PODS'07*, pages 133–142. ACM, 2007.

10 Georg Gottlob. Computing cores for data exchange: new algorithms and practical solutions. In *PODS*, pages 148–159, 2005.

11 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

12 Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2):1–49, 2008.

13 Pavol Hell and Jaroslav Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1-3):117 – 126, 1992.

14 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

15 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246. ACM, 2002.

16 Bruno Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.

17 Bruno Marnette. *Tractable Schema Mappings Under Oblivious Termination.* PhD thesis, University of Oxford, 2010.

18 Bruno Marnette, Giansalvatore Mecca, and Paolo Papotti. Scalable data exchange with functional dependencies. *PVLDB*, 3(1):105–116, 2010.

19 Giansalvatore Mecca, Paolo Papotti, and Salvatore Raunich. Core schema mappings: Scalable core computations in data exchange. *Inf. Syst.*, 37(7):677–711, 2012.

20 Reinhard Pichler and Vadim Savenkov. Towards practical feasibility of core computation in data exchange. *Theoretical Computer Science*, 411(7-9):935 – 957, 2010.

21 Rachel Pottinger and Alon Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J.*, 10(2-3):182–198, 2001.

22 Balder ten Cate, Laura Chiticariu, Phokion G. Kolaitis, and Wang Chiew Tan. Laconic schema mappings: Computing the core with sql queries. *PVLDB*, 2(1):1006–1017, 2009.