UNIVERSITY OF CALIFORNIA, SAN DIEGO

Foundations of Information Integration

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Mathematics and Computer Science

by

Alan Nash

Committee in charge:

Professor Jeff Remmel, Chair
Professor Russell Impagliazzo, Co-Chair
Professor Victor Vianu, Co-Chair
Professor Sam Buss
Professor Yannis Papakonstantinou
Professor Gila Sher

2006

The dissertation of Alan Nash is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

_____
Chair

University of California, San Diego

2006

To Rosemary Rooks.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

First of all, I want to thank my advisors: Jeff Remmel, Russell Impagliazzo, and Victor Vianu. I am very grateful to all three of them. To Jeff for his time, his advice, his patience, and his overall guidance. I started to work with him while he was still chair of the department of mathematics, yet it seemed he always had time to meet with me. Among other things, he encouraged me to explore many possibilities and foresaw that I may end up working in database theory at a time when I was thinking mostly of set theory. To Russell, I am very grateful for taking me on as an advisee when I already had an existing advisor in a different department, for letting me explore what computer science had to offer, and for his overall support even as I drifted away from computational complexity. To Victor, for introducing me to the world at the boundary of logic and databases and for his willingness to take me on as an advisee when I already had two other advisors.

I also want to thank Alin Deutsch, Bertram Ludäscher, and Yannis Papakonstantinou. All three of them were in the database group at UCSD when I first met them, but recently Bertram, who was at the San Diego Supercomputer Center, went over to UC Davis. Bertram took me on as a research assistant when I was just getting started in databases and my first database paper was with him. Alin has been essentially a fourth advisor to me. I have learned a lot from him, in particular about applications and properties of the chase. Chapter 3 of this thesis is joint work with him and Jeff Remmel. Yannis has been supportive throughout; in particular, he encouraged me and greatly assisted me in applying towards a Microsoft Research Fellowship.

On the Microsoft side, I think I owe my fellowship largely to Phil Bernstein. As a result of this fellowship and thanks to Phil I spent two summers at Microsoft Research where I learned a lot about information integration. Much of this I learned from Sergey Melnik, with whom I worked there in addition to Phil. Phil and Sergey have been very supportive throughout my job search.

Chapter 4 in this thesis is with Georg Gottlob, who was at the Technical University at Vienna during the time I worked with him on this topic and is now chair of the department of computer science at Oxford. He kindly invited me to Vienna to work with him, where I learned a lot about computing cores.

I did not expect to be in a doctoral program at all. It was Sam Buss who first suggested to me the possibility of taking graduate classes in mathematics through UCSD extension. The classes I took during my first quarter at UCSD where taught by

James Bunch and Nolan Wallach (Algebra), Bruce Driver (Analysis), Salah Baouendi (Differential Geometry), and Justin Roberts (Topology). It is thanks to their support and that of Thomas Enright (who was the vice chair of graduate studies at the time) that I was admitted into the doctoral program in mathematics at UCSD.

It is thanks to the support of Yannis Papakonstantinou, Jeff Remmel, and Russell Impagliazzo that I was allowed to pursue a dual Ph. D. in mathematics and computer science.

Portions of this dissertation are based on papers that I have co-authored with others. Listed below are my contributions to each of these papers.

1. "Data Exchange, Data Integration, and the Chase" by Alan Nash, Alin Deutsch, and Jeff Remmel [34]. I was responsible for developing all the concepts in this paper, except as follows. Jeff Remmel and Alin Deutsch contributed towards the proof of completeness, towards the overall presentation, introduced the unordered minimizing chase, and participated in many discussions in which the concepts discussed here were clarified.

2. "Data Exchange: Computing Cores in Polynomial Time" by Georg Gottlob and Alan Nash [17] (journal version [18]) I was responsible for developing the techniques to handle equality-generating dependencies and for the generalization beyond weakly-acyclic dependencies. I also contributed towards the presentation and the development of some of the minor technical points in the section on handling tuple-generating dependencies.

3. "Composition of Mappings Given by Embedded Dependencies" by Alan Nash and Phil Berstein and Sergey Melnik [32] (journal version [33]). I was responsible for developing all the concepts in this paper, except as follows. The section on de-Skolemization is joint work with Sergey Melnik. The main ideas and the presentation for the section on inexpressibility are by Sergey Melnik. Sergey Melnik and Phil Bernstein participated in many discussion where the concepts discussed were clarified.

VITA

2006	Dual Ph.D in Mathematics and Computer Science University of California, San Diego
2001	M.A. in Mathematics University of California, San Diego

PUBLICATIONS

"Data Exchange: Computing Cores in Polynomial Time" Georg Gottlob and Alan Nash, *25th ACM Symposium on Principles of Database Systems* (PODS 2006) Chicago, Illinois, 2006.

"Composition of Mappings Given by Embedded Dependencies" Alan Nash, Philip A. Bernstein, and Sergey Melnik, *24th ACM Symposium on Principles of Database Systems* (PODS 2005) Baltimore, Maryland, 2005.

"PTIME Queries Revisited" Alan Nash, Jeff Remmel, and Victor Vianu, *10th International Conference on Database Theory* (ICDT 2005) Edinburgh, Scotland, 2005.

"Rewriting Queries Using Views with Access Patterns Under Integrity Constraints" Alin Deutsch, Bertram Ludäscher, and Alan Nash, *10th International Conference on Database Theory* (ICDT 2005) Edinburgh, Scotland, 2005.

"Processing First-Order Queries Under Limited Access Patterns" Alan Nash and Bertram Ludäscher, *23rd ACM Symposium on Principles of Database Systems* (PODS 2004) Paris, France, 2004.

"Processing Unions of Conjunctive Queries with Negation under Limited Access Patterns" Alan Nash and Bertram Ludäscher, *9th International Conference on Extending Database Technology* (EDBT 2004) Heraklion, Greece, 2004.

"Universal Languages and the Power of Diagonalization" Alan Nash, Russell Impagliazzo, and Jeff Remmel, *18th IEEE Conference on Computational Complexity* (CCC 2003) Aarhus, Denmark, 2003.

"A Generalized Parallelogram Law" Alan Nash, *American Mathematical Monthly*, January 2003.

ABSTRACT OF THE DISSERTATION

Foundations of Information Integration

by

Alan Nash
Doctor of Philosophy in Mathematics and Computer Science
University of California San Diego, 2006
Professor Jeff Remmel, Chair
Professor Russell Impagliazzo, Co-Chair
Professor Victor Vianu, Co-Chair

We study three fundamental problems in information integration:

1. the data integration query problem,
2. the data exchange core computation problem, and
3. the schema mapping composition problem.

The first problem consists of computing the *certain answers* to a query over a target schema for a source instance under constraints which relate the source and target schemas. We show how to compute certain answers for a larger family of constraints and queries than those previously addressed. One of the main tools is the *chase*, which we study and extend significantly.

The second problem deals with inserting data from one database into another database having a different schema. Fagin, Kolaitis, and Popa have shown that among the universal solutions of a solvable data exchange problem, there exists – up to isomorphism – a most compact one, "the core", and have convincingly argued that this core should be the database to be materialized. We show how to compute the core in the general setting where the mapping between the source and target schemas is given by source-to-target constraints which are arbitrary tuple generating dependencies (TGDs) and target constraints consisting of equality generating dependencies (EGDs) and weakly-acyclic TGDs.

The third problem, composition of mappings between schemas, is essential to support schema evolution, data exchange, data integration, and other data management

tasks. We study the issues involved in composing schema mappings given by embedded dependencies that need not be source-to-target and we concentrate on obtaining (first-order) embedded dependencies. We provide a composition algorithm and several negative results. In particular, we show that even full dependencies that are not limited to be source-to-target are not closed under composition and that determining whether the composition can be given by these kinds of dependencies is undecidable. These negative results carry over to mappings given by embedded dependencies.

1

Introduction

1.1 Introduction

Information integration is concerned with putting together information from different data sources for various purposes. Problems in information integration have existed for a long time, most likely predating computers and certainly since soon after their introduction. There are many different parts to this endeavor including, for example, extracting structured data from sources which do not have much structure to begin with, such as HTML pages. We concentrate on the case where the data sources are relational databases and do not address such issues as "structure extraction" or the related problem of "data cleaning."

Relational databases consist of *relations*, which are sets of tuples of elements of some domain D. A relation has *arity* r if it is a subset of D^r. The data in a relational database is organized according to a *schema*, which is a list of relation names and their arities.

Integration can be achieved in many different ways, but many of these ways can be subsumed under variations of the following scenario, which is the main one we consider. Suppose there are several relational databases to be integrated. We can first pretend that these source schemas are disjoint, and then further pretend that there is a single source schema consisting of the union of the "real" source schemas and a single database obtained by simply putting together the "real" source databases.

Integration can be achieved by creating a *public* or *target* schema and allowing users (both human and systems) to interact with this target schema as if there was a database over that target schema. In some cases this target database may actually be constructed (*materialized*). The connection between the source schema and the target

schema is specified by a *schema mapping*, which is one of the fundamental objects of our study. The name "schema mapping" is misleading, but unfortunately well established. Schema mappings do not actually map schemas to schemas, but rather databases over the source schema to databases over the target schema. Secondly, schema mappings are often not functions; that is, to each source there may correspond several (even infinitely many) databases over the target schema that conform to the schema mapping. Formally, a schema mapping is a binary relation on databases, over the source schema for the domain and over the target schema for the range. When a schema mappings happens to be a function, we say that it is *functional*.

To simplify our considerations, we will pretend that the source schema σ and the target schema τ are disjoint. The schema mappings we consider are given by finite sets of sentences (*constraints*) in some logic over $\sigma \cup \tau$. A mapping m is *given* by constraints Σ when

$$(S, T) \in m \text{ iff } (S, T) \models \Sigma$$

where S is a database over σ, T is a database over τ, (S, T) is the database over $\sigma \cup \tau$ obtained by putting together the relations in S and T, and $(S, T) \models \Sigma$ means that (S, T) satisfies the constraints Σ. We will consider several different families of constraints, but the ones of most immediate interest to us are those constraints ξ of the form

$$\phi(\bar{u}) \rightarrow \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

where ϕ and ψ are conjunctions of atoms and may include equalities. We adopt the standard convention that all variables which are not otherwise quantified are universally quantified and we require that all variables $\bar{u}$ appear in ϕ. Such constraints are known as *embedded (implicational) dependencies* [15]. We call ϕ the *premise* and ψ the *conclusion*. If $\bar{v}$ is empty, then ξ is a *full dependency*. If ψ consists only of equalities, then ξ is an *equality-generating dependency (EGD)*. If ψ consists only of relational atoms, then ξ is a *tuple-generating dependency (TGD)*. *Source-to-target* constraints are those where ϕ is over σ and ψ is over τ. Similarly, *target-to-target* or simply *target* constraints are those over τ. In parts of these thesis we will also consider constraints where ϕ and ψ are allowed to include negation over relational atoms, inequality, and disjunction.

Tasks in information integration include the design of suitable target schemas and the specification of suitable schema mappings. We do not address these tasks, but rather concentrate on the case where the target schemas and the schema mappings are

given to us.

We are interested in investigating basic operations on and properties of schema mappings. It is our hope that in time, a general toolkit consisting of algorithms to perform such basic operations and to check such properties is developed. Of course, for such development it is also important to develop appropriate lower bounds and other limitative results which help to outline what is possible and what is not. Such study must necessarily be parametrized by the logical complexity of the constraints used to define schema mappings and should also help in the selection of an appropriate logic for such constraints that strikes a reasonable balance between expressive power and efficiency of the corresponding algorithms.

In this thesis, we concentrate on three basic problems (all three are operations that involve schema mappings and possibly an additional source and query):

- *The data integration query problem*: given a source S over σ, a schema mapping m between σ and τ given by constraints Σ, and a query Q over the target schema τ, answer Q.

- *The data exchange problem*: given a source S over σ, and a schema mapping m between σ and τ given by constraints Σ, create a database T over τ such that $(S,T) \in m$; that is, such that $(S,T) \models \Sigma$. A database T satisfying these conditions is called a *solution* for S under m (or Σ). In particular, we concentrate on the *core computation problem*: compute a *minimal* solution which is also *universal* (see the preliminaries below).

- *The composition problem*: given mappings m_{12} between σ_1 and σ_2 and m_{23} between σ_2 and σ_3 given by constraints Σ_{12} and Σ_{23}, find a set of constraints Σ_{13} defining the mapping m_{13} obtained from composing m_{12} and m_{23}; that is, such that

$$(R,T) \models \Sigma_{13} \text{ iff } \exists S\ (R,S) \models \Sigma_{12} \text{ and } (S,T) \models \Sigma_{23}.$$

In the course of our study of these three fundamental problems, we develop results and techniques which apply to other areas of database research, including query containment under constraints, testing implication of constraints, and query minimization. In particular, in Chapter 3 we investigate and extend the *chase*, introduce the closely-related notion of *template*, and we show that the chase produces templates. The chase has been used in databases for over 25 years to check query containment under constraints and implication of constraints.

Many kinds of schema mappings are used in information integration. For example, there are very complex mappings given by Pearl scripts. Such mappings are very hard to analyze, and we do not consider them here. Among those mappings which are more amenable to theoretical analysis, there has been a progression towards more and more expressive mappings, which can be roughly summarized as follows:

- *global-as-view (GAV)* mappings and *local-as-view (LAV)* mappings,
- *global-local-as-view (GLAV)* mappings, and
- *global-local-as-view (GLAV)* mappings + target constraints.

GAV mappings [ACPS96,GPQ+97] are those where the target instance is given by a set of queries $\mathcal{V}$ over the source schema. Such queries are known as views. The target schema is sometimes known as the *global* schema, which explains the name "global-as-view." GAV mappings are functional. All three problems given above are easy for GAV mappings. Problem 1 reduces to evaluating the query Q on the result of computing views $\mathcal{V}$ over the source S, which is straightforward. Problem 2 is trivial, since minimization is only hard when databases have variables (or *labelled nulls*) in them and the unique solution $T := \mathcal{V}(S)$ has no variables. Problem 3 is also easy, since composition of GAV mappings reduces to composition of the queries defining the views, which is usually straightforward.

LAV mappings [LRO96,FW97,DGL99] are not a generalization of GAV mappings, but simply a different kind of mapping. They are given by inclusion of the source relations in views over the target schemas. When the views are given by conjunctive queries with equalities. they are those given by source-to-target embedded dependencies where ϕ contains a single relational atom. They are generally non-functional.

GLAV mappings [FLM99] are a generalization of both GAV and LAV mappings. In the general case they are given by inclusions of queries over the source schema in queries over the target schema. When these queries are given by conjunctive queries with equalities. GLAV mappings correspond to mappings given by source-to-target embedded dependencies. A further generalization is GLAV mappings with additional target constraints, usually given by embedded dependencies. In this case, it is often necessary to have a condition on the target constraints to guarantee termination of the chase and the condition usually considered is *weak acyclicity* (see the preliminaries).

In some parts of this thesis we go beyond GLAV mappings given by conjunctive queries with equality and also consider

- general mappings given by embedded dependencies and

- general mappings given by universal-existential (Π_2) constraints.

The latter are those given by constraints that are similar to embedded dependencies, except that we also allow negation over relational atoms, inequality, and disjunction in ϕ and ψ.

The Data Integration Query Problem

In data integration, the target schema is the schema of a mediator, against which client applications can pose their query Q. The target instance is virtual, i.e. not materialized, and the task of the mediator is to compute the answer to Q using the source instance S. The answer to Q is defined as the intersection of the results of Q on all solutions: $\bigcap_{\text{solution } T} Q(T)$. This intersection is called the *certain answers* to Q.

[11] reveals a beautiful connection between the data exchange and the data integration query problems: if the client query Q is a union of conjunctive queries and the constraints in Σ are source-to-target and target-to-target embedded dependencies, then any universal solution U of the data exchange problem provides a solution to the data integration query problem: the certain answers to Q are the tuples in $Q(U)$ over the active domain of the source. Moreover, [11] shows that a universal solution can be computed using the chase procedure. The importance of these results is that they eliminate the challenge posed by the non-algorithmic definition of certain answers, which is based on infinitely many possible solutions. All one needs to do to find the certain answers is to compute a universal solution using the chase, and run the client query over it.

Unfortunately, the connection between the two problems breaks down as soon as more expressive queries or constraints are considered. As already observed in [11], if Q contains even one inequality, then its result on a universal solution may *strictly contain* the certain answers. Furthermore, it is shown in [16] that, when the constraints in Σ are not source-to-target, there simply may not be any universal solution, yet the set of certain answers may be non-empty. Looking at constraints beyond the source-to-target class is motivated in [16, 22] in the setting of peer data management. This case is also relevant to incorporating materialized warehouses and cached queries into the mediator for data integration [8].

As we will explain in the contributions section, we restore and strengthen the connection between data exchange and data integration by introducing the notion of a *universal set solution* and by considering alternative notions of universality.

The Data Exchange Core Computation Problem

While data integration usually deals with query translation and with query processing among multiple databases, data exchange aims at *materializing* a target database stemming from some source databases. Clearly, once transferred to the target database, the data can be queried according to the target schema. While data exchange has been recognized as an important problem for several decades, systematic research on foundational and algorithmic issues of this problem has started only a few years ago with the fundamental work of Fagin, Kolaitis, Miller, and Popa [11].

There can be several universal solutions to a data exchange problem and these solutions can noticeably differ in size. However, as observed in [12], there is – up to isomorphism – one particular universal solution, called *the core* (of any universal solution), which is the most compact one. More specifically, the core of a universal solution U is (up to isomorphism) the smallest subset V of U such that V is homomorphically equivalent to U. Fagin, Kolaitis, and Popa [12] argue that the core of a data exchange problem should be the solution of choice.

Computing the core of an arbitrary instance is known to be NP-complete. However, when the instance is a universal solution, we may use the constraints to guide the minimization problem in order to make it more efficient. The problem we tackle here is how to compute the core of a universal solution in polynomial time, for mappings given by source-to-target TGDs and target weakly-acyclic TGDs.

The Schema Mapping Composition Problem

Mapping composition refers to combining two mappings into a single one. If m_{12} is a mapping between schemas σ_1 and σ_2 and m_{23} is a mapping between schemas σ_2 and σ_3, then the composition $m_{12} \circ m_{23}$ of m_{12} and m_{23} is a mapping that captures the same relationship between σ_1 and σ_3 as the two mappings m_{12} and m_{23}.

Composition of mappings generalizes composition of queries, which is implemented in most commercial database systems. It is known that composition of two first-order queries (a.k.a. view unfolding) is a first-order query, that is, first-order queries are closed under composition. The same holds for conjunctive queries and unions of conjunctive queries.

Query composition corresponds to composition of functional mappings. In a more general setting, the mappings to be composed may be non-functional and this makes the problem of composing them harder. For example, answering queries using

views involves the composition of the query and the inverse of the view. But inverting a functional mapping often yields a non-functional mapping.

1.2 Organization

Chapter 2 introduces the notation used throughout this thesis. Then we cover three basic problems in information integration:

- The data integration query problem (Chapter 3).
- The data exchange core computation problem (Chapter 4).
- The schema mapping composition problem (Chapter 5).

Chapter 3 is organized as follows. In Section 3.1 we show how to compute certain answers using universal set solutions. Universal set solutions are special kinds of templates; we introduce the latter in Section 3.2. In Section 3.3 we show how to compute templates using the chase. In Section 3.5 we extend our results to constraints beyond embedded dependencies and in Section 3.6 to mappings other than homomorphisms and queries beyond UCQ. In Section 3.7 we show how templates apply to checking containment under constraints and implication. Then in Section 3.8 we show how to compute certain answers to queries with k-variables, even when universal set solutions do not exist.

Chapter 4 is organized as follows. Section 4.1 summarizes the relevant previous work and introduced the main ideas used in our solution on an informal level. The detailed proof of our main result is carried out in Sections 4.2-4.4. In particular, in section 4.2 we deal with cores and retractions (i.e., idempotent endomorphisms), in Section 4.3 we prove that cores of data exchange problems can be computed in polynomial time in case the set Σ_t of target dependencies consists is a weakly acyclic set of TGDs. This tractability results gives rise to the new FindCore algorithm. In Section 4.4 we extend the tractability of core finding to sets of target constraints that may contain, in addition, EGDs and arrive thus at the solution of the main open problem posed in [12]. In Section 4.5, we explain how the tractability result can be extended to a yet larger classes of target constraints Σ_t provided they satisfy a certain property, called the *bounded ancestor property.*

Chapter 5 is organized as follows. In Section 5.1 we present the deductive system which we use to as the basis of our composition algorithm. In Sections 5.2, 5.3, and 5.4 we study the composition of mappings given by, respectively, full, second-

order, and embedded dependencies. In Section 5.5 we introduce a tool for proving lower bounds on the number of embedded dependencies needed to axiomatize some classes of structures, which we use to prove one of our negative results. In Section 5.6 we examine the semantics of Skolemized constraints and study unrestricted composition. In Section 5.7 we briefly consider how some other basic operators on mappings such as domain and range relate to composition.

Chapter 6 is the conclusion.

1.3 Contributions and Related Work

Data Integration Query Problem: Chapter 3

This chapter makes the following contributions.

1. Universal set solutions. We restore and strengthen the connection between data exchange and data integration by introducing the notion of a *universal set solution*. This is a finite set U of solutions, sufficient to compute the certain answers to a query Q of arity r for a source S as follows:

$$\text{certain answers of } Q = \text{dom}(S)^r \cap \bigcap_{T \in U} Q(T).$$

2. Wider classes of queries and constraints. We show that such connection between data exchange and data integration holds for wider classes of queries than unions of conjunctive queries and for wider classes of constraints than source-to-target and target-to-target embedded dependencies. In particular, we show how to compute certain answers for

- arbitrary monotonic queries,
- unions of conjunctive queries with inequality and negation (UCQ$^{\neg,\neq}$or $\exists$-queries), and
- embedded dependencies extended with disjunction, inequality, and negation ($\forall\exists$-constraints), arbitrarily expressed over the combined schemas σ and τ (i.e. not restricted to be source-to-target and target-to-target) which satisfy the terminating conditions in item 8 below.

3. Data exchange. Our work provides solutions to the data exchange problem beyond the setting of source-to-target embedded dependencies studied in the literature [11, 16] and in Chapter 4. In our more general setting, the single universal

solution may not exist, or may not be useful for correctly computing certain answers to queries beyond conjunctive queries. Previous research does not provide any guidelines on what solution to materialize in that case. A natural choice is to materialize one solution from the universal set solution, if it exists. We show how to compute such solutions using the chase.

4. Relaxing universality. We address the case when there is no universal set solution, showing that certain answers can sometimes be computed for unions of conjunctive queries with at most k variables by finding a "not quite universal" set solution which we call k-universal. We introduce an interesting variant of the chase to compute k-universal set solutions.

5. Query containment, etc. Our results, concepts, and techniques are widely applicable beyond the data exchange and integration settings. In particular, they provide a unified solution for deciding query containment under constraints for expressive classes of queries and constraints which allow disjunction, inequalities and negated literals.

The machinery we develop to carry out the research program detailed above involves the following technical contributions.

6. Templates. We introduce the concept of a *template* of a class of instances, which generalizes the notion of a universal set solution and is relevant to the data exchange, data integration and containment problems. Given constraints Σ, we distinguish between a template for the set of all finite models of Σ which we call *weak* templates and a template for all (both finite and infinite) models of Σ which we call a *strong* template. We show that some embedded dependencies Σ have a weak template, but no strong template. It turns out that a universal set solution for S under Σ is precisely a weak template for constraints Σ_S satisfying $T \models \Sigma_S$ iff $(S, T) \models \Sigma$.

7. New chase. We show that strong templates are precisely what any "chase-like" procedure computes if it terminates. We start by formalizing the well-known ordered chase procedure to provide a unifying treatment of all chase extensions from recent prior work to cover arbitrary $\forall\exists$ constraints. We show that this ordered chase is incomplete, that is, it may fail to find a strong template even when one exists.

We then introduce a novel chase procedure, which we call the *unordered minimizing chase*, that is order-independent. We show that the unordered minimizing chase is complete for finding strong templates; that is, the unordered minimizing chase termi-

nates if and only if there is a strong template. From the point of view of completeness, this is the best any chase-like procedure can achieve. In particular, the unordered minimizing chase terminates whenever any chase-like procedure terminates and if there is a universal set solution that any chase-like procedure can find, the unordered minimizing chase will.

8. New termination conditions. We provide a widely-applicable, sufficient condition for termination of the ordered chase, which can be checked effectively on the constraints Σ. If this condition holds, we say that Σ is *stratified*. This new stratification condition is strictly more general than the best previously known such condition: weak acyclicity [11, 9].

Related Work. We have already discussed above the pioneering work in [11]. The chase was introduced in [28] where its connection to logical implication was established (an early ancestor was introduced in [2]). Related formulations of the chase for various kinds of dependencies were introduced in [27, 38]. The chase was extended to embedded dependencies in [3], to include disjunction and inequality in [9], and to arbitrary $\forall\exists$-sentences in [7].

Data Exchange Core Computation: Chapter 4

Since the core is in many contexts apparently the most satisfactory solution to a data exchange problem, it makes sense to study computational issues and the complexity of "getting to the core." This is a topic of [12] and the main topic of the Chapter 4.

Computing the core of an arbitrary relational instance with variables (which is not necessarily a universal solution of a data exchange problem) is NP-complete, as this is easily seen to be equivalent to a number of well-known NP-complete problems such as computing the core of a graph [23, 12], computing the smallest equivalent subquery contained in a conjunctive query [6], or computing the condensation of a clause [20]. Therefore, Fagin et al. [12] wondered whether the core of a universal solution of a data exchange problem whose target TGDs are weakly acyclic can be efficiently computed.

Fagin et. al formulated the following problem:

> **Problem [12]:** Given a data exchange problem whose source-to-target constraints are TGDs and whose target constraints consist of weakly-acyclic TGDs and arbitrary EGDs, can the core of a universal solution be computed in polynomial time?

This is precisely the problem we tackle in this chapter. While there has been some progress and partial answers, the problem remained open for about three years. It

was also mentioned as an important open Problem in Kolaitis' invited PODS'05 talk and paper *Schema Mappings, Data Exchange, and Metadata Management* [25]. The main result of the present chapter is the following positive answer to this question.

> **Theorem 4.9.** The core of a universal solution of a data exchange problem whose source-to-target constraints are TGDs and whose target constraints consist of weakly-acyclic TGDs and arbitrary EGDs can be computed in polynomial time.

The related work for this chapter is discussed in Section 4.1 since it is also used to introduce the main ideas used in our solution.

Composition of Schema Mappings: Chapter 5

Composition was recently studied by Madhavan and Halevy [26] and by Fagin, Kolaitis, Popa, and Tan [13] (see "Related Work" below). We address a similar set of questions as Fagin et al, but for different mapping languages. To be precise, they focus on mappings given by second-order constraints that are restricted to be source-to-target, while we we study constraints that are first-order and need not be source-to-target. (All of these terms will be defined shortly.) We have several motivations for pursuing this direction, including the following:

- Allow schema constraints. If they are present, composition yields mappings that may include functional dependencies or inclusion dependencies that are not source-to-target.
- Support mappings that express equality of views and, more generally, symmetric data exchange and peer-to-peer systems.
- Obtain closure under most basic mapping operators; in particular, composition and inverse.
- Ensure that checking whether a pair of instances is in a mapping has low complexity.
- Be able to deploy composition mappings in existing database system products.

In this new context, we extend the results of [13] in several directions. We study the composition of three related kinds of mappings:

(1) FuD-mappings (given by full dependencies)
(2) ED-mappings (given by embedded dependencies)
(3) SkED-mappings (given by second-order constraints)

and the corresponding mappings without equality. ED-mappings subsume st-tgds, functional dependencies and inclusion dependencies, and can express view definitions. SkED constraints subsume the SO tgds of [13], which in our terminology are source-to-target SkTGD constraints.

The case of most interest to us is that of ED-mappings. We show that one way to compose them is to:

1. Skolemize the ED-mappings to get SkED-mappings,
2. find a finite SkED axiomatization of all SkED constraints that hold for the composition, and
3. de-Skolemize the finite SkED axiomatization to get a ED-mapping.

The first step is easy; the difficulties arise in Steps 2 and 3. In the work [13], the source-to-target restriction simplifies Step 2. In Step 3, our goal is to obtain embedded dependencies, which requires eliminating second-order quantifiers. By contrast, the composition considered by Fagin et al. is given by second-order constraints, so in their case Step 3 does not apply.

In the case of SkTGD we consider both *restricted* composition, in which we are not allowed to introduce new function symbols, and *unrestricted* composition, in which we are free to introduce new function symbols. Since no function symbols appear in the other kinds of constraints we consider, restricted and unrestricted composition coincides for them.

Observe that our mapping languages are capable of expressing within-schema constraints, such as inclusion and functional dependencies. In this chapter, we assume that schema constraints are part of each mapping that mentions the schema. So we do not need to refer to them explicitly.

To list our contributions, we need to refer to many classes of mappings. To simplify the presentation, we use the following convention. Whenever we refer to a class of constraints without equality (for example, TGD), we imply that the result also holds for the corresponding class of constraints with equality (for example, ED), unless otherwise stated. In contrast, whenever we refer to a class of constraints with equality, we do not imply the result holds for the corresponding class of constraints without equality. Furthermore our negative results do not require the use of constants and our positive results allow constants. Our contributions include the following.

Negative results:

1. We show that FTGD-mappings are not closed under composition and that SkTGD-mappings are not closed under restricted composition (Theorem 5.3).
2. We show that the problem of determining whether the composition of two FTGD-mappings is a FTGD-mapping is undecidable (Theorem 5.4). This result carries over to TGD-mappings and to restricted composition of SkTGD-mappings.
3. Expressing the composition of two FTGD-mappings may require FTGD constraints that are exponentially larger in size than the input mappings, even over fixed schemas (Example 5.5). This result carries over to TGD-mappings and SkTGD-mappings. We show that there are TGD-mappings that require exponentially larger expressions in TGD than in SkTGD (Theorem 5.15); that is, an exponential increase in size may occur independently at each of the Steps 2 and 3 of the procedure outlined above. We develop a novel inexpressibility mechanism that allows us to show this result (Section 5.5).

Positive results:

4. We present necessary and sufficient (but uncomputable) conditions for composition of FTGD and restricted composition SkTGD-mappings (Theorem 5.5 and Theorem 5.11).
5. We present algorithms that compute the composition of FTGD and restricted composition SkTGD-mappings whenever they terminate (Corollary 5.8). These algorithms are very similar to each other and can be seen as an extension of the algorithm in [13] to handle mappings that are not restricted to being source-to-target.
6. We introduce exponential-time sufficient conditions for the algorithms above to terminate (Theorem 5.9).
7. We present an algorithm to compute the composition of TGD-mappings, which consists of three steps as outlined above: (1) Skolemize, (2) invoke the restricted composition algorithm for SkTGD-mappings, and (3) de-Skolemize.
8. The de-Skolemization step may fail. (After all, SkTGD has more expressive power than TGD since, as shown in [13], it can encode **NP**-complete problems.) We show how to check in polynomial time whether it will succeed and whether its output will be exponentially larger than its input (Proposition 5.14).
9. We identify exponential-time recognizable subsets of FTGD and SkTGD that are closed under composition (restricted for the latter) and inverse and that include

source-to-target constraints and constraints that express view definitions (Theorem 5.10). We do not provide such a subset of TGD since the conditions on it would be very restrictive (to ensure that de-Skolemization succeeds).

Additional (minor) results for second-order constraints:

10. We identify two different kinds of composition of SkTGD-mappings: restricted and unrestricted.
11. We introduce another fragment of second-order logic, ∃SOED. We show that every finite set of source-to-target constraints SkED is equivalent, under the special semantics, to a finite set of source-to-target ∃SOED constraints under the usual semantics for ∃SO (Theorem 5.27).

Composition is only one of many useful operations on mappings. Bernstein et al. [4, 5] introduced a general framework, called *model management*, in which operators on schemas and mappings are used to simplify the development of metadata-intensive applications. The basic operators include domain, range, composition, and inverse. Further operators are discussed in [30, 29].

12. We show that domain, range, and composition are closely related and can be reduced to each other (Proposition 5.29).

This is one reason why, in this context, composition and inverse are fundamental. The latter is easy if we use symmetric restrictions on the languages that define our mappings. Thus, FuD, ED, and SkED mappings have trivial inverse mappings. On the other hand, composition turns out to be very hard and is the primary subject of this chapter.

Related Work. A first semantics for composition of mappings was proposed in the pioneering work [26] by Madhavan and Halevy. Under their definition, m_{13} is a composition of m_{12} and m_{23} if the certain answers obtained by way of m_{13} for any query in a class of queries $\mathcal{L}$ against schema σ_3 are precisely those that can be obtained by using m_{23} and m_{12} in sequence. Notice that in their definition, composition depends on the query class $\mathcal{L}$. They focused on the relational case and considered mappings given by a certain class of tgds, which they call GLAV formulas.

Madhavan and Halevy showed that the result of composition may be an infinite set of formulas when the query language $\mathcal{L}$ is that of conjunctive queries, and proposed algorithms for the cases when composition can be done. Their definition has some disadvantages. In particular, the result of composition varies depending on the choice of

the query language $\mathcal{L}$. Also, the definition is asymmetric; that is, it is based on queries over σ_3 and does not consider queries over σ_1.

An alternative, language-invariant semantics for mapping composition was proposed independently by [13] and [31, Chapter 4]. Those papers considered mappings as binary relations on instances of schemas and defined mapping composition as a set-theoretic composition of such binary relations. This semantics makes the result of mapping composition unique and does not depend on a specific logical formalism chosen for representing mappings and queries.

Fagin, Kolaitis, Popa, and Tan [13] were the first to embark on a systematic investigation of mapping composition under these natural semantics. They presented many fundamental results; we survey only some of them here. First, they showed that *full* st-tgds are closed under composition, but that *embedded* st-tgds are not. To obtain closure in a more general setting, they introduced *SO tgds*, a second-order extension of st-tgds. These arise from Skolemizing embedded st-tgds. They showed that SO tgds are strictly more expressive than st-tgds and are closed under composition. This makes them a suitable mapping language for data exchange and query-rewriting scenarios [11, 39]. Further results in [13] are that composition of mappings given by st-tgds may give mappings undefinable in $L^{\omega}_{\infty\omega}$ and that composition of FO-mappings may give uncomputable mappings.

We extend the seminal work of [13] in two principal directions: (1) we study constraints that need not be source-to-target and (2) we concentrate on obtaining embedded dependencies (which are first-order).

Very roughly speaking, the main two challenges that we face involve recursion and de-Skolemization.

1.4 Acknowledgements

The results of Chapter 3 have been obtained with Alin Deutsch and Jeff Remmel. The results of Chapter 4 have been obtained with Georg Gottlob, partly during a visit to the Technical University of Vienna, and will appear in [17]. The results of Chapter 5 have been obtained with Phil Bernstein and Sergey Melnik, partly during two internships at Microsoft Research, and have appeared in [32]. We (Phil Bernstein, Sergey Melnik, and I) are grateful to Ron Fagin, Phokion Kolaitis, Lucian Popa, and Wang-Chiew Tan for fruitful discussions on mapping composition and for their feedback on a preliminary version of the paper [32] and we thank Andreas Blass who provided the

formula ψ in the proof of Theorem 5.3 and contributed to Proposition 5.29.

2

Preliminaries

2.1 Basics

A *schema* or *signature* σ is a finite list of relation symbols and their arities. An instance A *over* σ has one relation for every relation symbol in σ, of the same arity. We write arity(R) for the arity of a relation R. We define the domain $\mathrm{dom}(A)$ of an instance A as the set of *values* which appear in A. Unless otherwise specified, we consider finite instances with two types of values: constants and variables. The latter are also known as *labeled nulls*. An instance without variables is *ground*. If X is a set of values, we write $R|X$ for the relation R restricted to those tuples which have values in X and we $A|X$ for the subinstance of A obtained by restricting the relations in A to only those tuples which have values in X. For an instance A, we write $\mathrm{dom}(A)$ for the domain of A, $|A|$ for the size of $\mathrm{dom}(A)$, and R^A for the value of the relation R in A. If A, B are both over σ, we write $A \subseteq B$ if for every relation symbol $R \in \sigma$, $R^A \subseteq R^B$ and $A \sqsubseteq B$ if for every relation symbol $R \in \sigma$, $R^A = R^B|\mathrm{dom}(A)$.

We write $\bar{a}$ for a *tuple* $(a_1, \ldots, a_r)$ where the arity r is usually clear from the context. We write $a \in \bar{a}$ whenever $a = a_i$ for some i satisfying $1 \leq i \leq r$. In some cases, we overload the notation $\bar{a}$ to denote just a list of variables $a_1, \ldots, a_r$; which use is intended should be clear from context. In particular, sometimes we write $\bar{a} \in X$ to denote $a_1, \ldots, a_r \in X$ which is equivalent to $(a_1, \ldots, a_r) \in X^r$. This is a useful convention since it is cumbersome to write instead $\bar{a} \in X^r$ when the arity r of $\bar{a}$ has not been explicitly introduced. Similarly, we write $\bar{a} \notin X$ to denote $a_1, \ldots, a_r \notin X$, which is different from $(a_1, \ldots, a_r) \notin X^r$.

The *Gaifman graph* $\mathcal{G}(T)$ of instance T is the undirected graph G with vertex set $V^G := \mathrm{dom}(T)$ where there is an edge between $x, y \in \mathrm{dom}(T)$ iff x and y appear together

in some tuple of some relation of T. The *Gaifman graph of variables* $\mathcal{G}_V(T)$ of instance T is $\mathcal{G}(T)$ restricted to the variables in T. A *block* of T is a connected component of $\mathcal{G}_V(T)$. We write blocks(T) for the set of all blocks of T. If $v \in \text{var}(T)$, then block(v, T) denotes the block of T containing v. If $V \subseteq \text{var}(T)$, then blocks$(V, T) = \bigcup_{v \in V}\{\text{block}(v, T)\}$, i.e., blocks$(V, T)$ is the set of all blocks of T that contain at least one variable from V. The *block size* of an instance T, denoted by blocksize(T) is the maximum number of variables appearing in a block of T. We say that a set of instances K has *bounded block size* if there is a bound b such that the block size of every $T \in K$ is $\leq b$.

We adopt the RAM model for our complexity bounds.

2.2 Homomorphisms, Retractions, and Cores

Given two instances A and B, a function $h : \text{dom}(A) \to \text{dom}(B)$ is a *homomorphism* if whenever $R(\bar{a})$ holds in A, $R(h(\bar{a}))$ holds in B and if $h(c) = c$ for every constant in A. We write $A \to B$ in case there is a homomorphism between A and B. If $A \to B$ and $B \to A$, we say that A and B are homomorphically equivalent, which we write $A \leftrightarrow B$. We extend $\to$ to sets of structures K, L as follows:

$$K \to L \text{ iff } (\forall B \in L)(\exists A \in K)(A \to B).$$

It is easy to verify that this extension of $\to$ is reflexive and transitive. Notice that $K \subseteq L$ implies $L \to K$. We say that a structure or set of structures T is *universal* for K if $T \to K$.

A homomorphism $h : A \to A$ is an *endomorphism* of A. Since we only consider finite instances, notice that an endomorphism is surjective iff it is injective. An endomorphism r on A is a *retraction* if r is the identity on its range; that is, if r is idempotent ($r \circ r = r$). If also $r(A) = B \subseteq A$, we say that B is a *retract* of A and we write $A \hookrightarrow B$. If $A \hookrightarrow B$, then $A \leftrightarrow B$; that is, A and B are homomorphically equivalent since r is a homomorphism $A \to B$ and the identity is a homomorphism $B \to A$. A retraction is *proper* if it is not surjective (which in the case of finite instances is the same as not injective). An instance is a *core* if it has no proper retractions. A core C of an instance A is a retract of A which is a core; that is, C is a minimal retract of A.

An *isomorphism* is an injective homomorphism whose inverse is also a homomorphism. We write $A \cong B$ if there is an isomorphism from A to B. Cores of an instance A are unique up to isomorphism and therefore we can talk about *the core* of A, which we

denote core(A). It follows that A and B are homomorphically equivalent iff their cores are isomorphic. In symbols: $A \leftrightarrow B$ iff core(A) $\cong$ core(B).

2.3 Queries

A *query* of arity r is a function Q from instances over a schema to relations of arity r which furthermore has the following property, known as *genericity*: if i is an isomorphism $A \to B$, then $i(Q(A)) = Q(i(A))$. We consider the class CQ of conjunctive queries (consisting only of existential quantification and conjunction) and the class UCQ of unions of conjunctive queries (consisting also of disjunction) and their extensions to include inequality ($\text{CQ}^{\neq}$,$\text{UCQ}^{\neq}$), negation ($\text{CQ}^{\neg}$,$\text{UCQ}^{\neg}$), or both ($\text{CQ}^{\neg,\neq}$,$\text{UCQ}^{\neg,\neq}$). Notice that $\text{UCQ}^{\neg,\neq}$ is the same as the class of existential queries $\exists$Q. We write CQ_0 for the set of queries with conjunction but no existential quantifiers[1] and $\text{CQ}_0^{=}$ for the corresponding query class with equality. A query Q is *monotonic* if $A \subseteq B$ implies $Q(A) \subseteq Q(B)$. We write MonQ for the class of monotonic queries.

2.4 Constraints

We denote sets of constraints with capital Greek letters and individual constraints with lowercase Greek letters. In this paper, we only consider constraints ξ of the form

$$\phi(\bar{u}) \to \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

where ϕ and ψ are conjunctions of atoms, which may include equalities. We adopt the standard convention that all variables which are not otherwise quantified are universally quantified and we require that all variables $\bar{u}$ appear in ϕ. Such constraints are known as *embedded (implicational) dependencies* [15]. We call ϕ the *premise* and ψ the *conclusion*. If $\bar{v}$ is empty, then ξ is a *full dependency*. If ψ consists only of equalities, then ξ is an *equality-generating dependency (EGD)*. If ψ consists only of relational atoms, then ξ is a *tuple-generating dependency (TGD)*. Every set Σ of embedded dependencies is equivalent to a set of TGDs and EGDs. We write $A \models \Sigma$ and we say that A is a *model* of Σ if the instance A satisfies all the constraints in Σ We consider more expressive constraints in Section 3.5. Unless otherwise specified, we only consider *finite* sets of constraints and to simplify the presentation we do not say this explicitly in the statement of every result.

[1] We are not aware of any standard notation for this class.

We define the *width* of ξ to be the arity of $\bar{u}$ and the *height* of ξ to be the arity of $\bar{v}$. For a set Σ of constraints, the width of Σ is the maximum width of a constraint in Σ and the height of a Σ is the maximum height of a constraint in Σ.

In Chapter 5, we will need to refer to the following kinds of constraints:

Name	Abbreviation	Form
Full tuple-generating dependency	FTGD	$\forall\bar{x}(\phi(\bar{x}) \rightarrow \psi(\bar{x}))$
Tuple-generating dependencies	TGD	$\forall\bar{x}(\phi(\bar{x}) \rightarrow \exists\bar{y}\,\xi(\bar{x},\bar{y}))$
Skolemized TGD	SkTGD	$\exists\bar{f}\forall\bar{x}(\phi(\bar{x}),\chi(\bar{x}) \rightarrow \psi(\bar{x}))$
Existential second-order TGD	$\exists$SOTGD	$\exists\bar{R}\forall\bar{x}(\phi(\bar{x}) \rightarrow \exists\bar{y}\,\xi(\bar{x},\bar{y}))$

where $\phi(\bar{x})$ is a conjunction of relational atoms with variables in $\bar{x}$, $\chi(\bar{x})$ is a set of equalities between variables or between a variable and a term, $\psi(\bar{x})$ is a conjunction of relational atoms with variables in $\bar{x}$, $\xi(\bar{x},\bar{y})$ is a conjunction of relational atoms with variables in $\bar{x}$ and $\bar{y}$, $\bar{f}$ is a sequence of function symbols, and $\bar{R}$ is a sequence of relation symbols.

Terms are built from variables and functions in $\bar{f}$. We further require that every variable in $\bar{x}$ be *safe*. A variable is safe if it appears in a relational atom in $\phi(\bar{x})$ or alone on one side of an equation in χ where the other side is a term constructed from safe variables.

We define *full dependency* (FuD), *embedded dependency* (ED), *Skolemized embedded dependency* (SkED), and *existential second-order embedded dependency* ($\exists$SOED) like, respectively, FTGD, TGD, SkTGD, and $\exists$SOTGD, except we also allow equalities on the right hand side of $\rightarrow$.

The *Skolemization* of a universal-existential formula is the result of applying the following replacement: for every occurrence of a first-order existential quantifier v, remove $\exists v$ and replace the quantified variable wherever it appears in the scope of the quantifier with a new term of the form $f(\bar{x})$ where f is a new function symbol and $\bar{x}$ are the universally-quantified variables. In addition, introduce a second-order existential quantifier f just outside the scope of $\bar{x}$. For example, the Skolemization of $\forall xy(R(x,y) \rightarrow \exists z\, S(y,z))$ is $\exists f\forall xy(R(x,y) \rightarrow S(y,f(x,y))$. The classes of constraints in SkTGD we define here are in *unnested* form, so we would rewrite this as $\exists f\forall xyz(R(x,y), z = f(x,y) \rightarrow S(y,z))$. Skolemizing TGD constraints gives SkTGD constraints (but not all SkTGD constraints correspond to Skolemized TGD constraints). $\exists$SOTGD constraints are obtained by adding existential second-order quantification over relations to TGD.

The existential second-order quantification in SkTGD and $\exists$SOTGD apply to a finite set of constraints, not necessarily just one (this is not easy to illustrate in the table

above). Formally, we achieve this through a single sentence, which is the conjunction of the constraints in this finite set.

Given a source signature σ_S and a target signature σ_T disjoint from σ_S, a constraint is *source-to-target (ST)* if all the relational atoms in ϕ are over σ_S or $\bar{R}$ and all relational atoms in ψ or ξ are over σ_R or $\bar{R}$.

A *signature* is a function from a set of relation symbols to positive integers which give their arities. In this paper, we use the terms signature and schema synonymously. Given a set of constraints Σ over the signature $\sigma_1 \cup \sigma_2$, $\Sigma|_{\sigma_1}$ is the set of constraints in Σ which contain only relation symbols in σ_1.

2.5 Mappings

A schema *mapping* is a binary relation on instances of database schemas. Given a class of constraints $\mathcal{L}$, we associate to every expression of the form $(\sigma_1, \sigma_2, \Sigma_{12})$ the mapping

$$\{\langle A, B\rangle : (A, B) \models \Sigma_{12}\}.$$

Here Σ_{12} is a finite subset of $\mathcal{L}$ over the signature $\sigma_1 \cup \sigma_2$, σ_1 is the *input (or source) signature*, σ_2 is the *output (or target) signature*, A is a database with signature σ_1, and B is a database with signature σ_2. To simplify the presentation, we require that σ_1 and σ_2 be disjoint (otherwise, we do some renaming). (A, B) is the database with signature $\sigma_1 \cup \sigma_2$ obtained from taking all the relations in A and B together. Its domain is the union of the domains of A and B.

We say that m is *given* by expression E if the mapping that corresponds to E is m. Furthermore, we say that m is an *$\mathcal{L}$-mapping* if m is given by an expression $(\sigma_1, \sigma_2, \Sigma_{12})$ where Σ_{12} is a finite subset of $\mathcal{L}$.

2.6 Chase

In this section we introduce some basic concepts related to the chase. The chase is studied in more detail in Chapter 3.

Basics. The chase is a well-known algorithm which proceeds to build a sequence A_0^Σ, A_1^Σ, A_2^Σ, ... step by step as follows. Assume Σ is a set of TGDs and EGDs. We set $A_0^\Sigma = A$. To obtain A_{s+1}^Σ from A_s^Σ we proceed as follows. If there is some constraint $\xi \in \Sigma$ of the form

$$\phi(\bar{u}) \rightarrow \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

such that $A_s^\Sigma \not\models \xi$, we say that ξ *applies* to A_s^Σ. In this case, there must be some $\bar{a}$ such that $A_s^\Sigma \models \phi(\bar{a})$, but no $\bar{b}$ such that $A_s^\Sigma \models \psi(\bar{a}, \bar{b})$. For every such $\bar{a}$, we say that ξ *applies* to A^Σ on $\bar{a}$. There may be several constraints in Σ that apply to A_s^Σ and for each of them, several tuples they apply on. Of these, a constraint ξ and a tuple $\bar{a}$ are chosen by some total order on the pairs $(\xi, \bar{a})$. Then A_{s+1}^Σ is obtained from A_s^Σ as follows. If ξ is a TGD, then we add to $\mathrm{dom}(A_s^\Sigma)$ new variables $\bar{b}$ and to the relations in A_s^Σ the tuples that make up the conclusion $\psi(\bar{a}, \bar{b})$ to get $A_{s+1}^\Sigma \models \psi(\bar{a}, \bar{b})$. If ξ is an EGDs, we may assume that ψ consists of a single equality $u_i = u_j$. If a_i and a_j are two distinct constants, the chase *fails*. Otherwise, $A_{s+1}^\Sigma := h(A_s^\Sigma)$ where h satisfies $h(a_i) = h(a_j) = a_i$ and is the identity on all other values. Either way, $A_{s+1}^\Sigma \models \xi(\bar{a})$ and $A_s^\Sigma \to A_{s+1}^\Sigma$.

Parents, ancestors and siblings. Assume, the variables $\bar{b}$ are new variables introduced in a chase step that corresponds to a firing of a rule whose precondition is $\phi(\bar{a})$ and which makes $\psi(\bar{a}, \bar{b})$ true, as above. If $b \in \bar{b}$, we call every value in $\bar{a}$ a *parent* of b and any other variable in $\bar{b}$ a *sibling* of b (see Example 4.2 below). If Σ has width w and height e, then every $x \in \mathrm{dom}(A^\Sigma)$ has at most $e - 1$ siblings and at most w parents. We take the *ancestor* relation to be the transitive closure of the parent relation. We define the *depth* $\mathrm{depth}(x)$ of a value x to be one more than the depth of its deepest parent and the depth of the values in $\mathrm{dom}(A)$ to be zero. Notice that constants have no ancestors, no siblings, and depth zero. Also notice that a variable may have few ancestors yet may not be introduced by any short chase sequence, for example, because it may not be introduced until many full TGDs fire.

Order, termination, and universality. If for some A_s^Σ no constraint applies, we say that the chase *terminates* and we set $A^\Sigma := A_s^\Sigma$. If there is no such step, A^Σ is undefined. A^Σ is also undefined when the chase fails. In general, A^Σ depends on the order of the chase, but to keep the notation simple we will not explicitly indicate this order. If A^Σ is defined, then $A^\Sigma \models \Sigma$, $A \to A^\Sigma$, and A^Σ is universal for $\{B : A \to B, B \models \Sigma\}$ [28, 2, 15, 3, 1]. We write $A^{\Sigma,\Sigma'}$ for $(A^\Sigma)^{\Sigma'}$.

Since the chase does not always terminate, it is natural to ask for sufficient conditions for its termination. The following wide, sufficient condition on Σ for the termination of the chase on any instance, weak acyclicity was introduced in [9] and [11].

Definition 2.1. [11, 9] A *position* is a pair (R, i)
(which we write R^i) where R is a relation symbol of arity r and i satisfies $1 \leq i \leq r$. We say that x occurs in R^i in ϕ if there is an atom of the form $R(\ldots, x, \ldots)$ in ϕ where x appears in the ith position. The *dependency graph* of a set Σ of TGDs is a directed

graph where the vertices are the positions of the relation symbols in Σ and, for every TGD ξ of the form

$$\phi(\bar{u}) \rightarrow \exists \bar{v}\, \psi(\bar{u}, \bar{v}),$$

there is

1. an edge between R^i and S^j whenever some $u \in \bar{u}$ occurs in R^i in ϕ and in S^j in ψ and

2. an edge between R^i and S^j whenever some $u \in \bar{u}$ appears in R^i in ϕ and some $v \in \bar{v}$ occurs in S^j in ψ. Furthermore, these latter edges are labeled with $\exists$ and we call them *existential edges*.

Σ is *weakly acyclic* if its dependency graph has no cycles with an existential edge. We say that a set Σ of TGDs and EGDs is weakly acyclic if the set of TGDs in Σ is weakly acyclic.

Definition 2.2. If Σ is a weakly-acyclic set of TGDs, then the *depth* $\text{depth}(R^i)$ of a position R^i of a relation symbol R in Σ is the maximal number of existential edges in a path in the dependency graph for Σ ending at that position. The *depth* of Σ is the maximal depth of a position of a relation symbol in Σ. Notice that chasing a ground instance A with weakly-acyclic TGDs Σ of depth d, results in an instance A^Σ where the depth of every value is at most d.

Theorem 2.1 ([11, 9]). *For every weakly-acyclic set Σ of TGDs and EGDs, there are b and c such that, for any A, regardless of the order of the chase and except for the case where the chase fails due to EGDs,*

1. *A^Σ is defined, and*
2. *A^Σ can be computed in $O(|A|^b)$ steps and in time $O(|A|^c)$.*

2.7 Data Integration and Data Exchange

We consider the setting where we have two schemas σ and τ which do not share any relation symbols. Given an instance S over σ and instance T over τ, the instance (S, T) over $\sigma \cup \tau$ is the instance which has all the relations in S and all those in T. Given a set of constraints Σ over $\sigma \cup \tau$, we say that T is a *solution* for S under Σ if $(S, T) \models \Sigma$. When Σ is clear from context, we simply say that T is a solution for S. We say that U is a *universal solution* for S if it is a solution for S and if it is universal for the set of all

solutions for S. As in [11], we assume that source instances of a data exchange problem are ground.

A constraint ξ over (σ, τ) is *source-to-target* if the premise of ξ is over σ and the conclusion of ξ is over τ. Notice that any set of source-to-target TGDs is weakly acyclic. We will consider the special case of settings where $\Sigma = \Sigma_{st} \cup \Sigma_t$ with Σ_{st} a set of source-to-target TGDs and Σ_t a set of TGDs and EGDs. With these restrictions, $(\sigma, \tau, \Sigma_{st}, \Sigma_t)$ is known in the literature [11] as a *data exchange setting*.

Theorem 2.2 ([11]). *If $\Sigma := \Sigma_{st} \cup \Sigma_t$ where*

- *Σ_{st} is a set of source-to-target embedded dependencies and*
- *Σ_t is a weakly-acyclic set of TGDs,*

and $(S, \emptyset)^\Sigma$ is defined and is equal to (S, U) for some U, then U is a universal solution for S under Σ.

Given source instance S, we are also interested in finding the *certain answers* to Q for S under Σ, denoted $\text{cert}_Q^\Sigma(S)$ and defined as

$$\text{cert}_Q^\Sigma(S) = \bigcap_{(S,T) \models \Sigma} Q(T).$$

3

The Data Integration Query Problem

In this chapter, we study the problem of computing certain answers to a query over a target schema for a source instance under constraints which relate the source and target schemas. Prior work has shown that, for restricted constraints (source-to-target and target-to-target embedded dependencies) and unions of conjunctive queries, there is a special instance, known as a *universal solution*, such that running the query on it essentially yields the certain answers. Such a universal solution does not always exist for even slight extensions of these classes of constraints and queries.

We show that there may be a finite set of instances, which we call a *universal set solution*, which suffices to compute the certain answers. We also introduce the notion of a *k-universal set* solution, which is sufficient to compute the certain answers to queries with at most k variables, even when no universal set solution exists. We show how to compute such universal and k-universal set solutions for universal-existential constraints and existential queries.

The main algorithm for computing the universal set solution is an extended chase. We provide a completeness result for this chase and sufficient conditions for termination, which strictly extend the best previously known conditions (such as weak acyclicity). We also introduce a new kind of chase to compute k-universal set solutions.

3.1 Certain Answers

In this section, we show that, even when a single universal solution does not exist or when a universal solution exists but is insufficient for computing certain answers,

there may exist an adequate *universal set solution* which does allows us to compute certain answers.

Definition 3.1. Given a data exchange setting, we say that U is a *universal set solution* for source instance S under constraints Σ iff U is finite, contains only solutions, and is universal for the set of all solutions: $U \rightarrow \{T \ : \ (S,T) \models \Sigma\}$.

The following example shows that, if the queries are unions of conjunctive queries and the constraints are standard embedded dependencies beyond the source-to-target class, then it can be the case that there is no universal solution, so the certain answers cannot be computed in this way. However, there may be a universal *set* solution U, which suffices to compute certain answers correctly to any union of conjunctive queries Q, by computing $\bigcap_{T \in U} Q(T)$.

Example 3.1. Let the source schema and target schema consist of the binary relation symbol E, respectively the quaternary F, and consider a source instance S:

$$S = \{E(a, b_1), E(b_1, c), E(a, b_2), E(b_2, c)\}.$$

The constraint set $\Sigma = \{\xi_{st}, \xi_{ts}\}$ connects the two schemas as follows:

$$\begin{aligned} \xi_{st}: &\quad E(x,y), E(y,z) \rightarrow \exists u \exists w \ F(x,u,z,w) \\ \xi_{ts}: &\quad F(x,u,z,w) \rightarrow E(x,u), E(u,z) \end{aligned}$$

Consider the target query $Q(x,z) :- F(x,b_1,z,w) \vee F(x,b_2,z,w)$.
It is easy to check that the set of solutions for S contains, among others, $T_1 = \{F(a,b_1,c,$ $w_1)\}$, $T_2 = \{F(a,b_2,c,w_2)\}$, where w_1, w_2 are distinct. Indeed, (S,T_1) and (S,T_2) satisfy Σ. Note that there are infinitely many solutions, since w_i can take infinitely many values. However, it can be shown that each solution must include either T_1 or T_2, for some value of w_1, respectively w_2. Therefore, Q has the certain answer (a,c).

Note that there is no single universal solution C yielding the certain answers to Q. This is because by universality, C would have to map homomorphically into both T_1 and T_2 and therefore cannot contain the values b_1 or b_2. The answer to Q on C would therefore be empty and thus not coincide with the certain answers.

However, the certain answers can be computed from a universal *set* solution. It turns out (as will be detailed later) that a universal set solution U in this setting contains precisely two elements, $U = \{\{F(a,b_1,c,w_1)\}, \{F(a,b_2,c,w_2)\}\}$, where w_1, w_2

are variables (labelled nulls) which can in principle take any value. It is easy to check that the certain answers to Q can be obtained by computing $\bigcap_{T \in U} Q(T)$. Indeed,

$$Q(\{F(a, b_1, c, w_1)\}) \cap Q(\{F(a, b_2, c, w_2)\}) = \{(a, c)\} \cap \{(a, c)\} = \{(a, c)\}.$$

Example 3.1 is not a fortunate accident: Theorem 3.1 below shows that universal set solutions always yield the certain answers.

Theorem 3.1. *If W is a universal set solution for S under Σ and $Q \in$ UCQ has arity r, then*

$$\mathrm{cert}_Q^\Sigma(S) = \mathrm{dom}(S)^r \cap \bigcap_{T \in W} Q(T).$$

Proof. The inclusion $\subseteq$ is clear, since W consists only of solutions for S under Σ and since $\mathrm{cert}_Q^\Sigma(S) \subseteq \mathrm{dom}(S)^r$ by genericity of Q. For the opposite inclusion it is enough to show that for every solution T', there is $T \in W$ such that $Q(T) \cap \mathrm{dom}(S)^r \subseteq Q(T')$. Accordingly, pick a solution T'. Then there must be $T \in W$ and a homomorphism h such that $h : T \to T'$ and h is the identity on $\mathrm{dom}(S)$. Since UCQ is closed under homomorphisms by Theorem 3.17 below, we have $\bar{a} \in Q(T)$ implies $h(\bar{a}) \in Q(T')$. Furthermore, since h is the identity on $\mathrm{dom}(S)$, we have $\bar{a} \in \mathrm{dom}(S)^r$ implies $h(\bar{a}) = \bar{a}$ and therefore

$$\mathrm{dom}(S)^r \cap Q(T) \subseteq \mathrm{dom}(S)^r \cap Q(T') \subseteq Q(T')$$

as desired. □

Other kinds of universality. In Definition 3.1, universality of a set solution is defined with respect to homomorphisms. We mention here that it is very useful to consider universality with respect to other kinds of mappings, since the resulting universal set solutions yield the certain answers for more expressive queries. We consider such mappings in Section 3.6.

Theorem 3.2. *If W is a universal set solution for S under Σ for*

1. *injective homomorphisms and $Q \in$ MonQ,*
2. *full homomorphisms and $Q \in \mathrm{UCQ}^\neg$, or*
3. *embeddings and $Q \in \mathrm{UCQ}^{\neg,\neq}$,*

and Q has arity r, then

$$\mathrm{cert}_Q^\Sigma(S) = \mathrm{dom}(S)^r \cap \bigcap_{T \in W} Q(T).$$

Proof. Essentially the same as that of Theorem 3.1, also using Theorem 3.17 below. □

The following example (adapted from [11]) pertains to part 1 in Theorem 3.2. It shows that, even if the constraints are source-to-target embedded dependencies, if the query contains even one non-equality, the universal solution is insufficient for correctly computing the certain answers. However, there is a universal *set* solution U for injective homomorphisms which, according to Theorem 3.2, suffices for correct computation of certain answers.

Example 3.2. Let the source schema consist of the binary relation symbol E, and the target schema contain binary relations F, G. Consider a source instance

$$S = \{E(a_1, b_1), E(a_2, b_2)\}.$$

The two schemas are connected by $\Sigma = \{\xi_{st}\}$, where

$$\xi_{st} : E(x, z) \rightarrow \exists y \; F(x, y), G(y, z)$$

Consider the query $Q(x, z) :- F(x, y), G(y', z), (y \neq y')$.
Q has no certain answers, since its result on the solution

$$T_1 = \{F(a_1, y), G(y, b_1), F(a_2, y), G(y, b_2)\}$$

is already empty. However, a universal solution according to [11] is

$$T_0 = \{F(a_1, y_1), G(y_1, b_1), F(a_2, y_2), G(y_2, b_2)\}$$

and, as observed there, the result of Q on T_0 is non-empty: $Q(T_0) = \{(a_1, b_2), (a_2, b_1)\}$. As shown in Section 3.3, there exists a universal *set* solution U for injective homomorphisms, which correctly captures the certain answers to Q. Indeed, U contains two instances, $U = \{T_0, T_1\}$. Then $Q(T_0) \cap Q(T_1)$ correctly yields the empty set of certain answers. Notice that there is a homomorphism, but no injective homomorphism, from T_0 into T_1.

Universal set solutions do not always exist:

Example 3.3. Consider the constraints Σ:

$$\begin{aligned} S(x,y) &\rightarrow T(x,y) \\ T(x,y) &\rightarrow \exists z\, T(y,z) \\ T(x,y), T(y,z) &\rightarrow T(x,z) \end{aligned}$$

and a source S containing a single edge. Then any solution, since it must be finite, must have a cycle. But for any finite set of solutions U, there must be a solution C_n which is a cycle larger than any cycle in U. Then $U \not\rightarrow C_n$. Therefore, there is no universal set solution for S under Σ.

3.2 Templates

Universal set solutions depend not only on the constraints Σ, but also on the source S. In this section, we generalize universal set solutions by introducing the concept of *templates*. Templates have other applications outside of data exchange and data integration, including checking query containment under constraints. We will see in Section 3.3 that the chase is a general algorithm for producing templates, not just universal set solutions, so the extra generality gives a better understanding of the power and limitations of the chase.

The connection between templates and universal set solutions is as follows. A universal set solution is a finite approximation to the set of all solutions. Similarly, a template is a finite approximation to the set of all models of a set of constraints. Given a set of constraints Σ and a source instance S, we define a set of constraints Σ_S so that the models of Σ_S are precisely the solutions for S under Σ. Then a universal set solution for S under Σ is precisely a template for Σ_S. We will gain some insight on the nature of the universal set solution for S under Σ by looking at the form of the constraints Σ_S. In particular, it will turn out that for embedded dependencies Σ, Σ_S is not always equivalent to a set of embedded dependencies, but when it is, then there is a universal solution if and only if there is a universal set solution.

Definition 3.2. We define Σ_S so that

$$T \models \Sigma_S \text{ iff } (S,T) \models \Sigma$$

by replacing every occurrence in Σ of $R\bar{x}$ where R is a relational symbol in σ with

$\bigvee_{\bar{c} \in R^S} \bar{x} = \bar{c}$. This may give constraints which have disjunction in the premise, but we "normalized away" such disjunctions. On the other hand, when there are any constraints with relation symbols from σ in the conclusion, we may get disjunction in the conclusion which can not be normalized away.[1]

Example 3.4. We revisit Example 3.2, where $\Sigma = \{\xi_{st}\}$,

$$\xi_{st} : E(x, z) \rightarrow \exists y \; F(x, y), G(y, z)$$

and the source S is $\{E(a_1, b_1), E(a_2, b_2)\}$. The set Σ_S contains the single constraint

$$x = a_1, z = b_1 \vee x = a_2, z = b_2 \quad \rightarrow \quad \exists y \; F(x, y), G(y, z)$$

which is equivalent to the set of TGDs

$$\begin{aligned} x = a_1, z = b_1 \quad &\rightarrow \quad \exists y \; F(x, y), G(y, z) \\ x = a_2, z = b_2 \quad &\rightarrow \quad \exists y \; F(x, y), G(y, z) \end{aligned}$$

It is easy to see that T is a solution for S under Σ iff $T \models \Sigma_S$.

Definition 3.3. A set of finite structures T is a *template* for a set of structures K if it satisfies the following conditions:

1. (universality) $T \rightarrow K$,
2. (conformance) $T \subseteq K$,
3. (finiteness) T is finite, and
4. (minimality) there is no $T' \subset T$ such that $T' \rightarrow T$.

These conditions imply that there is no T' satisfying 1, 2, and 3 such that $|T'| \leq |T|$. We write $[K]$ for the template of K, if it exists and we write $[\Sigma]$ for $[\text{mod}(\Sigma)]$, where $\text{mod}(\Sigma)$ is the class of all finite models of Σ.

The connection between universal set solutions and templates is as follows: a template for Σ_S is a universal set solution for S under Σ. (We give the precise statement in Proposition 3.8 below.) Therefore, templates can also be used to compute certain answers.

[1] Since here we start chasing with an empty instance, we will need to allow for chase steps (see Section 3.3) of the form $A_n \overset{\xi, \bar{a}}{\rightarrow} A_{n+1}$ where $\bar{a}$ is not necessarily in A_n for these constraints to fire once we chase.

Theorem 3.3. *If $[\Sigma_S]$ exists and Q is a conjunctive query of arity r, then*

$$\begin{aligned} \mathrm{cert}_Q^{\Sigma}(S) &= \mathrm{dom}(S)^r \cap \bigcap_{T \in [\Sigma_S]} Q(T) \\ &= \{\bar{c} : \Sigma_S \models Q(\bar{c})\}. \end{aligned}$$

Proof. The proof of the first equation is very similar to that of Theorem 3.1, except that we use the fact that if T' is a solution for S under Σ, then $T' \models \Sigma_S$ and therefore there is $T \in [\Sigma_S]$ such that $T \to T'$. The second equation follows immediately from the definition of certain answers and Σ_S. □

Why standard data exchange admits a universal solution. We have defined templates for arbitrary constraints, which are not necessarily source-to-target, and may contain disjunction. This case goes beyond the data exchange setting in the literature [11]. We now prove from the basic properties of templates the fact that in the particular case of embedded source-to-target dependencies, the template is a singleton (or, equivalently, there exists a universal solution).

Proposition 3.4. *If Σ is a set of embedded dependencies where all conclusions are over τ, then Σ_S is a set of embedded dependencies. In particular, this holds in the case where Σ is a set of source-to-target TGDs and target-to-target TGDs and EGDs, as in [11].*

Embedded dependencies have some nice closure properties.

Theorem 3.5. *If Σ is a set of embedded dependencies, then Σ is closed under retractions and products. That is:*

1. *If $A \models \Sigma$ and $A \hookrightarrow B$, then $B \models \Sigma$.*
2. *If $A, B \models \Sigma$, then $A \times B \models \Sigma$.*

Proof. Assume ξ is an embedded dependency of the form

$$\phi(\bar{u}) \to \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

with $\bar{v}$ possibly empty.

1. Assume that $A \models \xi$ and $h : A \hookrightarrow B$. Then if $B \models \phi(\bar{b})$ for $\bar{b} \in \mathrm{dom}(B)$, then also $A \models \phi(\bar{b})$. Therefore, since $A \models \xi$, there are $\bar{a} \in \mathrm{dom}(A)$ such that $A \models \psi(\bar{b}, \bar{a})$. This implies $B \models \psi(h(\bar{b}), h(\bar{a}))$ and, since h is a retraction and $\bar{b} \in \mathrm{dom}(B)$, $h(\bar{b}) = \bar{b}$, so $B \models \psi(\bar{b}, \bar{c})$ for $\bar{c} = h(\bar{a})$. That is, $B \models \xi$.

2. Assume that $A, B \models \xi$ and set $C := A \times B$. Then if $C \models \phi(\bar{c})$ for $\bar{c} \in \mathrm{dom}(C)$, by definition of C and because ϕ is a conjunction of relational atoms, we must have $\bar{a}, \bar{b}$ such that for every i, $c_i = (a_i, b_i)$ and such that $A \models \phi(\bar{a})$ and $B \models \phi(\bar{b})$. Since $A, B \models \xi$, there must be $\bar{d}$ and $\bar{e}$ such that $A \models \psi(\bar{a}, \bar{d})$ and $B \models \psi(\bar{b}, \bar{e})$. Then $\bar{f}$ obtained by setting $f_i = (d_i, e_i)$ satisfies $C \models \psi(\bar{c}, \bar{f})$. That is, $C \models \xi$.

□

In particular, closure under products is enough to guarantee that universal set solutions are in fact simply universal solutions.

Proposition 3.6. *If $[K]$ exists and K is closed under products, then $[K]$ is a singleton. Furthermore, in this case* $\mathrm{core}(\{\prod_{A \in K} A\})$ *is a template for K.*

Corollary 3.7. *If there is a universal set solution for S under Σ and Σ is a set of source-to-target and target-to-target TGDs and EGDs, Σ_S is a set of embedded dependencies, or Σ_S is closed under products, then there is a universal solution for S under Σ.*

Templates for finite and infinite solutions. We say that T is a *strong template* for Σ if T is a template for $\mathrm{imod}(\Sigma)$, the class of all models (finite and infinite) of Σ. We say that T is a *weak template* for Σ if T is a template for $\mathrm{mod}(\Sigma)$, the class of all finite models of Σ.

Proposition 3.8. *A weak template for Σ_S is precisely a universal set solution for S under Σ.*

Clearly, any strong template is also a weak template. However, the converse is not true, as shown by the following separation result. We discuss in Section 3.3 how we compute both template flavors.

Theorem 3.9.

1. *There is a set Σ of TGDs which has a weak template, but no strong template.*
2. *There is a set Σ of TGDs which has no weak template.*

Proof. (1) Consider the set of axioms Σ_1:

$$\begin{array}{lrcl} \xi_1\text{:} & & & \exists x, y\, E(x, y) \\ \xi_2\text{:} & E(x, y) & \rightarrow & \exists z\, E(y, z) \\ \xi_3\text{:} & E(x, y), E(y, z) & \rightarrow & E(x, z) \end{array}$$

Any model of axioms ξ_1 and ξ_2 must have an infinite walk. Therefore, if the model is finite, it must have a cycle. If it has a cycle, then by axiom ξ_3 it must have a self-loop. Since the structure with a single self-loop satisfies these axioms, it is a weak template for Σ_1. On the other hand, the transitive closure of an infinite path also satisfies axioms ξ_1, ξ_2, and ξ_3, but no finite structure with a cycle has a homomorphism into it. Therefore Σ_1 has no strong template.

(2) Now consider $\Sigma := \{\xi_1, \xi_2\}$. As we have seen above, any finite model of Σ must have a cycle. But for any finite set U of such models, there is another model C_n, a cycle larger than any cycle in U and therefore $U \not\to C_n$. □

3.3 Computing Templates

Given a set of constraints Σ, we are interested in computing the template $[\Sigma]$. In this section we concentrate on computing templates for embedded dependencies. To this end, we start from the well-known chase procedure [2, 28, 27, 15, 3, 1], extending it to a novel procedure called the *unordered minimizing chase* which turns out to be strictly better at computing templates than the standard chase. We will show in Section 3.5 how to extend the chase to compute templates for larger classes of constraints and under more general universality assumptions, as required by queries which are more expressive than unions of conjunctive queries.

By Theorem 3.5 and Proposition 3.6, if they exist, templates for embedded dependencies consist of a single structure. To simplify the presentation we refer to this single structure also as a template.

Definition 3.4. (Chase Step) If ξ is a TGD or EGDs, we write $A \overset{\xi,\bar{a}}{\to} B$ if

1. $A \models \exists P_\xi(\bar{a})$,
2. $A \not\models \exists C_\xi(\bar{a})$, and
3. $B = \begin{cases} A\bar{a} \oplus C_\xi & \text{if } \xi \text{ is a TGD} \\ h(A) & \text{if } \xi \text{ is an EGD} \end{cases}$

where $A\bar{a} \oplus C_\xi$ is the result of attaching to A a copy of C_ξ by identifying $\bar{a}$ with the free variables of C_ξ and where $h(a_i) = h(a_j) = a_i$ and h is the identity elsewhere in case $\exists P_\xi$ has as free variables $\bar{u}$ and C_ξ is $u_i = u_j$. If 1 and 2 hold, we say that ξ *applies* to A on $\bar{a}$. We do not require $\bar{a} \in \mathrm{dom}(A)$, which is important in case the premise has constants.

Definition 3.5. (**Chase Sequence**) Assume Σ is a set of EGDs and TGDs.

1. A Σ-*chase sequence* S (or just *chase sequence* if Σ is clear from context) is a sequence of structures $A_0, A_1, \ldots$ such that every structure A_{s+1} in it is obtained from the previous one A_s by a chase step. That is, there are $\xi \in \Sigma$ and $\bar{a}$ such that $A_s \overset{\xi,\bar{a}}{\to} A_{s+1}$. We say that S starts with A if $A_0 = A$.

2. A chase sequence $A = A_0, \ldots, A_n$ is *terminating* if $A_n \models \Sigma$. In this case we say that $A^\Sigma = A_n$ is the *result* of the chase.

3. We say that the chase *terminates* if there is a terminating chase sequence. A^Σ is defined whenever there is some terminating chase sequence starting with A, but its value may depend on the chase sequence. We will see later that all chase results are homomorphically equivalent, so we can often speak about A^Σ without referring to a particular chase sequence.

4. We say that an infinite chase sequence is *fair* if whenever ξ applies to A_s on $\bar{a}$ there is some $r > s$ such that $A_r \models \exists C_\xi(\bar{a})$.

5. If Σ consists of TGDs only and $A = A_0, A_1, \ldots$ is an infinite chase sequence, we set $A^\Sigma_\omega = \bigcup_i A_i$. If no sequence is specified, we take A^Σ_ω to be obtained as above from any fair infinite chase sequence. Then A^Σ_ω is only defined up to homomorphic equivalence as in the case of A^Σ.

The following theorem lists some essential properties of the chase, which we will generalize for the extended chase. These results are considered folklore or appear implicitly in proofs related to the chase [2, 28, 27, 15, 3, 1].

Theorem 3.10. *If Σ is a set of TGDs and EGDs, $A = A_0, A_1, \ldots$ is a finite or infinite chase sequence, and*

$$K = \{B \in \mathrm{imod}(\Sigma) : A \to B\},$$

then:

1. $A_0 \to A_1 \to A_2 \to \ldots$
2. $A_0, A_1, A_2, \ldots \to K$
3. *If A^Σ is defined, then $A^\Sigma \models \Sigma$.*
4. *If Σ consist of TGDs only, then A^Σ_ω is universal for K and $A^\Sigma_\omega \models \Sigma$.*
5. *If $B \models \Sigma$ and there is a homomorphism $h : A_n \to B$, then there is a homomorphism $h' : A_{n+1} \to B$ extending h (if ξ is a TGD) or identifying some values in the domain of h (if ξ is an EGD).*

6. *If $B \models \Sigma$, A^Σ is defined, and there is a homomorphism $h : A \to B$, then there is a homomorphism $h' : A^\Sigma \to B$.*

Proof. 1. Follows from the definition of $\overset{\xi,\bar{a}}{\to}$.

2. Follows from 5 below.
3. Follows immediately from the definition of A^Σ.
4. If $A^\Sigma_\omega \not\models \Sigma$, then for some $\bar{a}$ and $\xi \in \Sigma$, we must have $A^\Sigma_\omega \models \exists P_\xi(\bar{a})$ and $A^\Sigma_\omega \not\models \exists C_\xi(\bar{a})$. But then this must also hold for A_s for some s, the former because the range of any homomorphism $h : \exists P_\xi \to A^\Sigma_\omega \bar{a}$ is finite and the latter by monotonicity of $\exists C_\xi$. But then, by definition of fairness, there must be A_r for some $r > s$ such that $A_r \models \exists C_\xi(\bar{a})$, contradicting $A^\Sigma_\omega \not\models \exists C_\xi(\bar{a})$.
5. Assume $A_n \overset{\xi,\bar{a}}{\to} A_{n+1}$. Then since $A_n \models \exists P_\xi(\bar{a})$, $B \models \exists P_\xi(h(\bar{a}))$. Therefore, there is $\bar{b}$ (possibly empty) such that $B \models C_\xi(h(\bar{a}), \bar{b})$. If ξ is a TGD, then we can map C_ξ to $C_\xi(h(\bar{a}), \bar{b})$ to get the desired extension h'. If ξ is an EGD, then h must map the equated values to the same value in B, so the restriction $h|\mathrm{dom}(A_{n+1})$ is also a homomorphism.
6. Follows from 5 above.

□

Corollary 3.11.

1. *If A^Σ is defined, then A^Σ is a template for*

$$K = \{B \in \mathrm{imod}(\Sigma) : A \to B\}.$$

2. *If $\emptyset^\Sigma$ is defined, then $\emptyset^\Sigma$ is a strong template for Σ.*

Proof. Immediate from parts 2 and 3 in Theorem 3.10. □

We say that any sequence is *chase-like* if it satisfies the conditions of Theorem 3.10. We say that an algorithm is *chase-like* if it produces chase-like sequences. It would be nice to have an algorithm that satisfied the conditions of Theorem 3.10 only for finite instances, but it is not clear at all how such algorithm can be obtained. Certainly, it can not be obtained by simply adding a copy of the conclusion of some constraints to the next instance in the sequence. Therefore, any chase-like algorithm, if it terminates, produces a strong template. A natural question, then, is the following:

Is the chase complete for strong templates?

That is, for any Σ with a strong template, will the chase always find it? More precisely, will any (long enough) Σ-chase sequence terminate? For the chase outlined above, the answer is no, as the following example shows.

Example 3.5. Consider the set Σ consisting of following TGDs:

$$\begin{array}{lrcl} \xi_1\colon & & & \exists u, v\, E(u,v), E(v,u) \\ \xi_2\colon & E(x,y), E(y,x) & \rightarrow & \exists u\, E(u,u) \\ \xi_3\colon & E(x,y) & \rightarrow & \exists u\, E(x,u), E(u,y) \end{array}$$

The singleton template consisting of the self-loop is a template for Σ, yet any Σ-chase sequence starting with $\emptyset$ must be infinite. This is because ξ_1 must fire first to give a cycle of length 2. Assume ξ_2 fires next to give a disjoint self-loop. From now on, ignore this loop. Set $A_0 := C_2$, the cycle of length 2. Now it is easy to show that if $A_s \overset{\xi_3,a,b}{\longrightarrow} A_{s+1}$ where $a \neq b$, then two new edges ac and cb are added to A_{s+1} and that

$$A_{s+1} \not\models \exists C_{\xi_3}(ac) \text{ and } A_{s+1} \not\models \exists C_{\xi_3}(cb).$$

Therefore, if $A_{s+1} \not\models \xi_3$, which results in an infinite chase sequence.

A Complete Chase. We now define a novel chase procedure, the *unordered minimized chase* or *um-chase*, which is order-independent and which, more importantly, is complete for strong templates, and therefore superior to the chase introduced above (which we also call the *ordered chase*) for the task of finding templates, which is what chase-like algorithms are all about.

Intuitively, the unordered chase step proceeds by first firing all applicable standard chase steps simultaneously, then minimizing the resulting structure by computing its core. We formalize the procedure below.

Definition 3.6. (**Unordered-chase step**)
If Σ is a set of TGDs, we write $A \overset{\Sigma}{\rightarrow} B$ if

1. $A \not\models \Sigma$ and
2. $B = \bigcup_{\xi \in \Sigma, \bar{a} \in \mathrm{dom}(A), A \overset{\xi,\bar{a}}{\hookrightarrow} D} D$.

That is, B is the structure obtained from A by simultaneously firing all applying chase steps. If Σ also contains EGDs, then we also identify all elements which have been identified by every constraint ξ and any tuple $\bar{a}$ such that ξ applies to A on $\bar{a}$.

For the minimization part of the next step, we need to define cores. An instance is a *core* if it has no proper retractions. A core C of an instance A is a retract of A

which is a core. Cores of an instance A are unique up to isomorphism and therefore we can talk about *the core* of A. It follows that A and B are homomorphically equivalent iff their cores are isomorphic.

Definition 3.7. (**Unordered-minimizing-chase step**)
We write $A \xrightarrow{\Sigma\downarrow} B$ if $A \xrightarrow{\Sigma} B'$ and $B = \text{core}(B')$.

We extend the definition of chase sequence to um-chase sequence in the obvious way. Notice that um-chase sequences are determined up to isomorphism, since cores are determined up to isomorphism. We use the notation A^Σ to refer also to the result of a terminating um-chase sequence. Such result is unique up to isomorphism. Similarly, we use A^Σ_ω also for the um-chase. Which chase we have in mind should be clear from context. Theorem 3.10 also holds for the um-chase.

Theorem 3.12. *If Σ is a set of EGDs and TGDs, then*

1. *The unordered minimizing chase terminates iff there is a strong template for Σ.*
2. *The result of the unordered minimizing chase is a strong template for Σ.*

Proof. Part 2 follows from Theorem 3.10. Therefore, if the unordered minimizing chase terminates, then there is a strong template for Σ, namely the result of the chase.

For the converse of part 1, assume first that Σ consists only of TGDs and assume there is a strong template $\{T\}$ for Σ and there is no finite unordered chase with minimization sequence starting with $\emptyset$. Then there must be an infinite unordered chase sequence starting with $\emptyset$: $\emptyset = A_0, A_1, A_2, \ldots$. Set $A^\Sigma_\omega = \bigcup_i A_i$, which is well defined because for all i, $A_i \subseteq A_{i+1}$. Since $\{T\}$ is a strong template for Σ and $A^\Sigma_\omega \to T$ by Theorem 3.10, $T \to A^\Sigma_\omega$. Since T is finite, $T \to A_n$ for some n and, by Theorem 3.10, $A_n \to T$. But then core(T) and core(A_n) are isomorphic and therefore both satisfy Σ. Now consider the unordered chase with minimization sequence starting with $\emptyset$: $A_0, A_1, \ldots$. It is easy to verify by induction that for every s, $A_s = \text{core}(A_s)$. In particular, $A^\Sigma_n = \text{core}(A_n)$ and therefore $A^\Sigma_n \models \Sigma$ and this sequence is finite.

If Σ consists of EGDs and TGDs, we can simulate the EGDs with TGDs to obtain $\bar{\Sigma}$ as explained in [17]. Then also core(T) and core(A_n) are isomorphic for some n as above and the rest of the argument goes through unchanged. □

Corollary 3.13. *If Σ is a set of embedded dependencies and any chase-like algorithm terminates on input A under Σ to give U, then the unordered minimizing chase terminates on input A and gives the strong template A^Σ which is homomorphically equivalent to U.*

Proof. If such chase-like algorithm terminates on input A under Σ giving U, then U is a strong template by Corollary 3.11. Therefore, by Theorem 3.12 part 1, the unorderd minimizing chase terminates and by part 2 gives a strong template A^{Σ}. Since both U and A^{Σ} satisfy Σ by Theorem 3.10, we have $U \to A^{\Sigma}$ and $A^{\Sigma} \to U$ by their universality. □

3.4 Conditions for Termination

A widely-applicable, sufficient condition on Σ for the termination of the chase, is weak acyclicity, defined in the preliminaries.

Now consider the following example.

Example 3.6. Consider the set $\Sigma = \{\xi\}$ where ξ is the following TGD:

$$\exists y\, R(x,y), R(y,x) \to \exists u, v\, R(x,u), R(u,v), R(v,x).$$

It is easy to check that Σ is not weakly acyclic, yet it is clear that A^{Σ} is defined for the ordered chase for any chase order since introducing 3-cycles will never create any new 2-cycles.

We introduce a strictly wider, effectively checkable condition which is also sufficient for termination of the chase, which is motivated by the example above.

Definition 3.8. (**Stratified**) Given EGDs or TGDs α and β we write $\alpha \prec \beta$ if there exists A, B, $\bar{a} \in \mathrm{dom}(A)$, and $\bar{b} \in \mathrm{dom}(B)$ such that

1. β does not apply to A on $\bar{b}$, possibly because $\{\bar{b}\} \not\subseteq \mathrm{dom}(A)$,
2. $A \overset{\alpha,\bar{a}}{\to} B$ for some B, and
3. β applies to B on $\bar{b}$

The *chase graph* G of a set of TGDs Σ has as vertices the constraints in Σ and there is an edge between two constraints $\alpha, \beta \in \Sigma$ if $\alpha \prec \beta$. We say that C is a *cycle-component* of G if C is a connected component of the graph G' consisting only of those edges which participate in cycles. A set of EGDs and TGDs Σ is *stratified* if every cycle component of the chase graph of Σ is weakly-acyclic.

Theorem 3.14. *If Σ is a stratified set of EGDs and TGDs, then there exists c such that for every A, the length of every chase sequence $A_0^{\Sigma}, A_1^{\Sigma}, \ldots$ is bounded by $|A|^c$.*

Proof. (sketch) Consider the chase graph G of Σ and its associated graph G' where every cycle component has been replaced by a single vertex. G' is acyclic. Each vertex in G' is a set of weakly-acyclic TGDs or a single TGD. Now chase as follows. Pick a vertex v of indegree 0 in G', chase with the associated TGD or weakly-acyclic set of TGDs until this subchase terminates, remove v from G' and repeat. Each subchase must terminate because β fires after α only if $\alpha \prec \beta$ and because a subchase with a weakly-acyclic set of TGDs terminates by Theorem 2.1. □

Theorem 3.15. *Given TGDs α and β, checking whether $\alpha \prec \beta$ holds is decidable.*

Proof. Assume $\alpha \prec \beta$. Then, by the definition, there are $A, B, \bar{a}, \bar{b}$ satisfying conditions 1, 2, and 3 of the definition. In particular, there is a homomorphism $h : \exists P_\beta \to B\bar{b}$. Set B' to be a minimal instance such that $h(B) \subseteq B'$ and such that there is $A' \subseteq B'$ satisfying $A' \overset{\alpha,\bar{a}}{\to} B'$. Then $A', B', \bar{a}, \bar{b}$ also satisfies condition 2 of the definition by construction, and conditions 1 by monotonicity of P_β and condition 3 by monotonicity of C_β together with $h(B) \subseteq B'$. Furthermore, such B' must satisfy $|B'| \leq |C_\alpha| + |P_\beta|$ so we only need to examine a finite set of candidates A', B'. In fact, it is enough to consider unions of homomorphic images of P_β with C_α as candidates for B and remove from them an induced substructure isomorphic to C_α to get candidates for A. □

Theorem 3.16. *Weakly-acyclic sets of EGDs and TGDs are stratified, but not conversely.*

Proof. If a set of EGDs and TGDs is weakly acyclic, then it is stratified by the definition. The set $\Sigma = \{\xi\}$ from Example 3.6 satisfies $\xi \not\prec \xi$ and therefore is stratified, yet not acyclic. To see this, notice if $A \overset{\alpha,\bar{a}}{\to} B$ then B has no new 2-cycles or 1-cycles, that is, no such cycles which are not already in A. □

3.5 Beyond Embedded Dependencies

In this section we extend our consideration to constraints of the form

$$\bigvee_{1 \leq i \leq p} \phi_i(\bar{u}, \bar{w}) \to \exists \bar{v} \bigvee_{1 \leq i \leq c} \psi_i(\bar{u}, \bar{v})$$

where each ϕ_i and ψ_i is a conjunction of relational atoms, negated relational atoms, equalities, or inequalities. We call such constraints *negation disjunctive embedded dependencies* or *NDEDs* to be consistent with the name *disjunctive embedded dependencies* or

DEDs for the same class of constraints without negation introduced in [9]. It is easy to check that every $\forall\exists$ constraint is equivalent to an NDED. We extend the results of the previous sections to these kinds of constraints. To do this, we first show how to handle disjunction and then we show how to handle negation.

Adding disjunction. First of all, notice DEDs that are not closed under products as the following example shows. This is in constrast to embedded dependencies (cf. Theorem 3.5).

Example 3.7. The following disjunctive TGD ξ

$$\exists u, v Euv, Evu \text{ or } \exists u, v, w Euv, Evw, Ewu$$

is not closed under products. We have $C_2, C_3 \models \xi$ where C_k is the directed cycle of length k, yet $C_2 \times C_3 = C_6$ and $C_6 \not\models \xi$.

As a consequence, a template for a set Σ of such constraints is not necessarily a singleton set. We now explain an extended chase for DEDs introduced in [9]. The comments above imply that such chase must have at each step not a single instance, but instead a set of instances. Therefore, we aim to define $K \xrightarrow{\xi} L$ where K and L are finite sets of instances and ξ is an DED to parallel Definition 3.4. Then we extend the results from Section 3.3 and we show how to extend them to NDEDs.

Definition 3.9. (**Extended Chase Step**) First assume that ξ is a DED of the form shown above. Set

$$\xi_i := \bigvee_{1 \le i \le p} \phi_i(\bar{u}, \bar{w}) \rightarrow \exists \bar{v}, \psi_i(\bar{u}, \bar{v})$$

so that $P_\xi = P_{\xi_1} = \ldots = P_{\xi_p}$ and $C_\xi = \bigvee_{1 \le i \le c} C_{\xi_i}$. We write $A \overset{\xi,\bar{a}}{\rightarrow} \{B_1, \ldots, B_c\}$ if

1. $A \models \exists P_\xi(\bar{a})$,
2. $A \not\models \exists C_\xi(\bar{a})$, and
3. for each i, $A \overset{\xi_i,\bar{a}}{\rightarrow} B_i$.

If 1 and 2 hold, we say that ξ *applies* to A on $\bar{a}$. Notice that this is consistent since if 2 holds, then also $A \not\models \exists C_{\xi_i}(\bar{a})$ for every i. That is, we create one new instance for every disjunct in the conclusion.

We write $K \overset{\xi,\bar{a}}{\rightarrow} L$ where K and L are finite sets of instances if

$$L = K_1 \cup \bigcup_{A \in K_2, A \overset{\xi,\bar{a}}{\rightarrow} M} M$$

where

$$K_1 := \{A \in K : \{\bar{a}\} \not\subseteq A \text{ or } A \not\models \exists P_\xi(\bar{a}) \text{ or } A \models \exists C_\xi(\bar{a})\}$$

and $K_2 := K - K_1$. That is, K_1 is the set of instances in K to which ξ does not apply on $\bar{a}$ and K_2 is the set of instances in K to which ξ does apply on $\bar{a}$. The instances in L are those obtained by a chase step with ξ and $\bar{a}$ from an instance in K or those instances in K to which ξ does not apply on $\bar{a}$.

Given this definition of a chase step, the definitions of chase sequence, chase result, etc. from Section 3.3 extend naturally to the situation where at each step we have a finite set of instances instead of a single instance. This extension was made in [9]. We keep the notation of Section 3.3 since this makes the presentation simpler and more intuitive. Notice that A^Σ_ω may now be an infinite set of instances. Then Theorems 3.10 and Theorems 3.12 go through with DEDs instead of EGDs and TGDs. Only minor and straightforward changes are needed in their proofs.

Adding negation. We now explain a further extension to handle negation, introduced in [7]. We first extend DEDs with constraints that may have $\perp$ as their conclusion and we extend the definition of a chase step as follows. If ξ has $\perp$ as its conclusion, then

$$K \overset{\xi,\bar{a}}{\rightarrow} L \text{ iff } L = \{A \in K, A \not\models \exists P_\xi(\bar{a})\}.$$

That is, if ξ applies to A on $\bar{a}$, it "kills" A. We call such constraint *DEDFs* for *DEDs with falsehood*. Implicitly, such constraints are already needed for a much smaller kinds of constraints in order to handle contradictions which arise, for example, from equating two constants.

Using DEDFs, we can simulate NDEDs. The details of this simulation are given in the next section; here we only provide a rough sketch. Given a set Σ of NDEDs over signature σ, we compute a set $\hat{\Sigma}$ of DEDFs over the signature

$$\hat{\sigma} := \sigma \cup \{\hat{R} : R \in \Sigma \cup \{N\}$$

where $\hat{R}\bar{x}$ "stands for" $\neg R\bar{x}$ and Nxy "stands for" $x \neq y$. If the chase terminates, the result will be a template under embeddings, rather than homomorphisms. From such result, it is straightforward to extract a template under homomorphisms (cf. Theorem 3.19).

3.6 Beyond UCQ and Homomorphisms

In order to be able to compute certain answers to queries which are not unions of conjunctive queries, we need set solutions which are universal not under homomorphisms, but under other kinds of mappings. Universality under other kinds of mappings is useful also for other applications, including checking containment under constraints.

We are interested in the following kinds of homomorphisms. We say that a homomorphism $h : A \to B$ is *full* if $A \models R(\bar{x})$ iff $B \models R(h\bar{x})$ for all relations R in A and B. An *embedding* is a full injective homomorphism. We write hom for the set of homomorphisms on finite instances, ihom for injective homomorphisms, fhom for full homomorphisms, and emb for embeddings.

Theorem 3.17.

1. UCQ *is closed under homomorphisms.*
2. MonQ *is closed under injective homomorphisms.*
3. $\text{UCQ}^{\neg}$ *is closed under full homomorphisms.*
4. $\text{UCQ}^{\neg,\neq}$ *is closed under embeddings.*

That is, for any of these classes of queries and the corresponding class of mappings,

$$h : A \dashrightarrow B \text{ and } \bar{a} \in Q(A) \text{ implies } h(\bar{a}) \in Q(B).$$

Proof. 1. Assume that $h : A \to B$ is a homomorphism, $\bar{a} \in Q(A)$, $Q \in$ UCQ, and $Q := \bigvee_{1 \le i \le k} Q_i$ where each $Q_i \in$ CQ. Then $\bar{a} \in Q_i(A)$ for some i and this happens iff there is a homomorphism $g : Q_i \to A\bar{a}$, that is, a homomorphism from Q_i to A which maps the free variables $\bar{x}$ of Q_i to $\bar{a}$. But then $h \circ g : Q_i \to Bh(\bar{a})$ and therefore $h(\bar{a}) \in Q(B)$.

2. Assume that $h : A \to B$ is an injective homomorphism, $\bar{a} \in Q(A)$, and $Q \in$ MonQ. Set $A' = h(A)$; that is, for every relation in A, set $R^{A'} = h(R^A)$. Then $A' \subseteq B$ and h is an isomorphism between A and A'. Therefore, by genericity, $h(\bar{a}) \in Q(A')$ and by monotonicity, $h(\bar{a}) \in Q(A)$.
3. Simlar to part 1. If $\bar{a} \in Q_i(A)$ then there is a homomorphism $g : Q_i \to A\bar{a}$ which also preserves the absence of some tuples. Composing h with g gives a homomorphism which preserves the absence of those tuples.
4. Similar to part 3, but with embeddings.

□

We fix some family F of mappings on finite instances such as injective homomorphisms and we write $A \dashrightarrow B$ if there is $h : A \to B$ in F. We extend $\dashrightarrow$ to sets of structures as we did for $\to$. We say that a structure or set of structures T is *F-universal* for K if $T \dashrightarrow K$.

Definition 3.10. A set of finite structures T is an *F-template* for a set of structures K if it satisfies the following conditions:

1. (F-universality) $T \dashrightarrow K$,
2. (conformance) $T \subseteq K$,
3. (finiteness) T is finite, and
4. (minimality) there is no $T' \subset T$ such that $T' \to T$.

Therefore, a plain template is a hom-template.

It turns out that we can compute strong templates under F in case $F \in \{\mathrm{ihom}, \mathrm{fhom}, \mathrm{emb}\}$ by using a reduction to the case of homomorphisms. Other families of mappings may be handled by similar reductions.

Theorem 3.18. *Given $F \in \{\mathrm{ihom}, \mathrm{fhom}, \mathrm{emb}\}$ and a signature σ, there is a signature $\hat{\sigma} \supset \sigma$ and constraints Λ such that for every A, B over σ, there are unique expansions $\hat{A}, \hat{B}$ over $\hat{\sigma}$ of A, B satisfying $\hat{A}, \hat{B} \models \Lambda$. Furthermore,*

$$h : A \dashrightarrow B \text{ iff } h : \hat{A} \to \hat{B}$$

Proof. (a) If F consists of injective homomorphisms, set $\hat{\sigma} := \sigma \cup \{N\}$ where N is a new binary relation symbol and set Λ to contain the constraints

$$x = y \vee N(x, y) \qquad x = y, N(x, y) \to \bot$$

where N stands for $\neq$. Now assume 1, 2, and 3 hold. If $h : \hat{A} \to \hat{B}$ is a homomorphism, then also h is a homomorphism $A \to B$. Now if $x \neq y$, we must have $A \models Nxy$ by the first constraint in Λ and therefore $B \models Nh(x)h(y)$. Then the second constraint in Λ implies $h(x) \neq h(y)$; that is, h is injective. The converse is obvious.

(b) If F consists of full homomorphisms, set $\hat{\sigma} := \sigma \cup \{\hat{R} : R \in \sigma\}$ where each relation symbol $\hat{R}$ is new and of the same arity as R. Set Λ to contain all constraints of the form

$$R(\bar{x}) \vee \hat{R}(\bar{x}) \qquad R(\bar{x}), \hat{R}(\bar{x}) \to \bot$$

for every relation symbol $R \in \sigma$. The rest of the proof is similar to case (a).

(c) Embeddings are precisely full injective homomorphisms, so this case is handled by combining (a) and (b).

The one-to-one correspondence between A and $\hat{A} \models \Lambda$ is clear. □

Theorem 3.19. *If Σ is a set of NDEDs over signature σ and $F \in \{\text{ihom}, \text{fhom}, \text{emb}\}$, then there is a signature $\hat{\sigma}$ and a set of DEDFs $\hat{\Sigma}$ such that the following are equivalent*

1. *There is a strong template for Σ.*
2. *There is a strong F-template for Σ.*
3. *There is a strong template for $\hat{\Sigma}$.*
4. *The unordered chase with minimization terminates for $\hat{\Sigma}$, producing a strong template for $\hat{\Sigma}$.*

Furthermore, $\hat{\Sigma}$ can be obtained efficiently from Σ and each of these strong templates can be efficiently obtained from the other.

Proof. Set $\hat{\sigma}$ and Λ as in the proof of Theorem 3.18 and set $\hat{\Sigma} := \Sigma \cup \Lambda$.

1 implies 3: Assume there is a strong template T for Σ. Set $\hat{T}$ to consist of all expansions of all homomorphic images of instances in T to $\hat{\sigma}$ that satisfy Λ. Then $\hat{T}$ satisfies all conditions for a template except minimality as follows. Conformance and finiteness are obvious. If $\hat{B} \models \hat{\Sigma}$, then its reduction B to σ satisfies Σ and therefore there is $A \in T$ and a homomorphism $h : A \to B$. Then there is $A' = h(A) \in T$ such that $g : A' \to B$ is injective.

3 implies 2: Assume there is a strong template $\hat{T}$ for $\hat{\Sigma}$. Then the set T of structures in $\hat{T}$ reduced to the signature σ all conditions for an F-template except minimality as follows. Conformance and finitiness are obvious. Universality is satisfied by Theorem 3.18, since if $B \models \Sigma$, there is an expansion $\hat{B}$ of B such that $\hat{B} \models \hat{\Sigma}$. Then there is $\hat{A} \in \hat{T}$ such that $\hat{A} \to \hat{B}$ and therefore the reduction A of $\hat{A}$ to σ satisfies $A \dashrightarrow B$. It is easy to obtain $T' \subseteq T$ which satisfies minimality as well.

2 implies 1: Assume there is a strong F-template T for Σ. Since every F-mapping is a homomorphism, T satisfies all conditions for a template, except for minimality. It is easy to obtain $T' \subseteq T$ which satisfies minimality as well.

3 iff 4: This follows from Theorem 3.12.

The proof above shows how to obtain each of the strong templates from the others. □

Theorem 3.19 allows us to compute templates and universal set solutions for NDED constraints.

3.7 Containment

Templates are useful for checking containment, containment under constraints, and implication as we show next.

Definition 3.11. We write $P \sqsubseteq Q$ in case $P(A) \subseteq Q(A)$ for all finite structures A. and $P \sqsubseteq_\Sigma Q$ in case $P(A) \subseteq Q(A)$ for all finite structures $A \models \Sigma$. The *containment problem* consist of, given P and Q, deciding whether $P \sqsubseteq Q$. The *containment under constraints problem* consist of, given P, Q, and Σ, deciding whether $P \sqsubseteq_\Sigma Q$.

In order to state general results in a simple manner we consider some fixed class of mappings F closed under composition, such as hom, fhom, ihom, or emb and we consider the corresponding F-templates. We write $A \dashrightarrow B$ if there is a mapping $h : A \to B$ such that $h \in F$. We say that a class of structures K is *closed* under $\dashrightarrow$ if $A \in K, A \dashrightarrow B$ imply $B \in K$. Similarly, we say that Σ is *closed* under $\dashrightarrow$ if $\mod(\Sigma)$ is.

In a sense, an F-template is an incomplete, but finite description of a class K, unless that class K is closed under F. The importance of the result below, is that reduces an infinite problem (part 1) to checking finitely many instances for the existences of a mapping (part 3).

Theorem 3.20 (Containment). *If K, L are sets of structures, L is closed under $\dashrightarrow$, and $[K]$ and $[L]$ exists, then the following are equivalent.*

1. $K \subseteq L$.
2. $[K] \subseteq L$.
3. $[L] \dashrightarrow [K]$.

Proof. If (1) holds, then $[K] \subseteq K \subseteq L$ so (2) holds. If (2) holds, then $[L] \dashrightarrow L$ and therefore $[L] \dashrightarrow [K]$ so (3) holds. Now assume (3) holds and pick $A \in K$. Since $[K] \dashrightarrow K$, there is $B \in [K]$ such that $B \dashrightarrow A$. Since $[L] \dashrightarrow [K]$, there is $C \in [L]$ such that $C \dashrightarrow B \dashrightarrow A$. Since $[L] \subseteq L$, $C \in L$. Since F is closed under composition, $C \dashrightarrow A$ and since L is closed under $\dashrightarrow$, $A \in L$. This shows that (1) holds. □

We want to be able to speak of a set of instances associated with a query Q. Therefore, given a query Q, we define a sentence $\hat{Q}$ obtained from Q by replacing the

free variables $\bar{x}$ of Q with new constants $\bar{c}$. Notice that with this definition, we have by Theorem 3.17:

1. If $Q \in$ UCQ, then $\hat{Q}$ is closed under homomorphisms.
2. If $Q \in$ MonQ, then $\hat{Q}$ is closed under inj. homomorphisms.
3. If $Q \in \text{UCQ}^{\neg}$, then $\hat{Q}$ is closed under full homomorphisms.
4. If $Q \in \text{UCQ}^{\neg,\neq}$, then $\hat{Q}$ is closed under embeddings.

To simplify the notation, we write $[Q]$ instead of $[\hat{Q}]$. If $Q \in$ CQ, then $[Q]$ is precisely what is known as the *frozen instance* of Q. Therefore, for the case of queries, templates generalize the notion of frozen instances. Furthermore, we have the following natural generalization of what is known as *the homomorphism theorem* for conjunctive queries [1]:

Corollary 3.21 (Query Containment). *If $P, Q \in \mathcal{L}$ and $\mathcal{L}$ is closed under the mappings F, then*

$$P \sqsubseteq Q \text{ iff } [Q] \to [P].$$

The following important result follows from Theorems 3.20 and 3.10.

Theorem 3.22 (Implication). *If $P, Q \in \mathcal{L}$, $\mathcal{L}$ is closed under the mappings F, and Σ is a set of sentences, then the following are equivalent whenever all templates mentioned exist:*

1. $P \sqsubseteq_\Sigma Q$.
2. $\Sigma \models \forall\bar{x}(P \to Q)$.
3. $\Sigma, \hat{P} \models \hat{Q}$.
4. $\text{mod}(\Sigma \cup \{\hat{P}\}) \subseteq \text{mod}(\hat{Q})$.
5. $[Q] \to [\Sigma \cup \{\hat{P}\}]$

Furthermore, if some Σ-chase terminates on P, then 1-5 are also equivalent to:

6. $P^\Sigma \sqsubseteq Q$

Partial results similar to Theorems 3.20 and 3.22, except for the mention of templates, are known for several special cases including conjunctive queries and embedded dependencies. Our contribution is identifying the crucial roles of templates in them and therefore providing a uniform generalization to any kinds of constraints, mappings, and queries closed under such mappings for which we can find templates.

Notice also that in some cases, we do not need to chase to obtain a template, but can simply asserts its existence to obtain the desired result. In particular, we can then 'use' a weak template, which we know can not be found by chasing. An example of a result of this kind is the following generalization of Theorem 5 in [7]:

Theorem 3.23. *For any set Σ of universal-existential constraints, if there is some c such that for any $P, Q \in \mathrm{UCQ}^{\neg,\neq}$, a weak template $T = [\Sigma \cup \{\hat{P}\}]$ exists and every instance A in it satisfies $|A| \leq |P|^c$, then we can check whether $P \sqsubseteq_\Sigma Q$ holds in $\mathbf{\Pi_2^P}$.*

Proof. Assume the hypotheses. Now for every A satisfying $|A| \leq |P|^c$, $A \models \Sigma$, and $A \models \hat{P}$, check whether $A \models \hat{Q}$. If this holds, then we must have $T \subseteq \mathrm{mod}(\hat{Q})$, and therefore $P \sqsubseteq_\Sigma Q$ by Theorem 3.22. Otherwise, we have a counterexample. □

3.8 Queries with k-Variables

As we have seen in Theorem 3.9, there are sets of TGDs with no templates and, more specifically, there are data integration settings which have no universal set solutions as Example 3.3 shows. Can we still compute certain answers in such situations? We show that in many situations we can, by relaxing the notion of universal set solution as follows. We write $A \xrightarrow{k} B$ if

$$(\forall A' \sqsubseteq A)(|A'| \leq k \rightarrow A' \rightarrow B).$$

That is, every restriction of A to at most k elements has a homomorphism into B. This notion can easily be generalized to other kinds of mappings and it also applies to checking containment.

If $A \xrightarrow{k} B$, we say that there is a k-homomorphism from A to B, even though this fact is not usually witnessed by a single mapping. Nevertheless, the binary relation given by $\xrightarrow{k}$ is reflexive and transitive, and therefore $\xrightarrow{k}$ gives a preordering. It turns out this is all we need to define templates in the most general way: we simply replace universality in the definition of template with universality under some preordering. Here we define k-templates for homomorphisms; a similar definition applies to other mappings.

Definition 3.12. A set of finite structures T is a *k-template* for a set of structures K if it satisfies the following conditions:

1. (k-universality) $T \xrightarrow{k} K$,
2. (conformance) $T \subseteq K$,

3. (finiteness) T is finite, and
4. (minimality) there is no $T' \subset T$ such that $T' \to T$.

We define k-universal solutions similarly, using k-universality instead of plain universality. It turns out that k-universality is all we need to answer existential queries with k variables (we assume variables are not reused). More precisely, CQ_k is the set of conjunctive queries with at most k variables (the existential quantification has been pushed out). and UCQ_k are unions of CQ_k queries. We define the other families of existential queries in a similar way.

Theorem 3.24.

1. *UCQ_k is closed under k-homomorphisms.*
2. *MonQ_k is closed under injective k-homomorphisms.*
3. *$\mathrm{UCQ}_k^{\neg}$ is closed under full k-homomorphisms.*
4. *$\mathrm{UCQ}_k^{\neg,\neq}$ is closed under k-embeddings.*

That is, for any of these classes of queries and the corresponding class of mappings,

$$h : A \xrightarrow{k} B \text{ and } \bar{a} \in Q(A) \text{ implies } h(\bar{a}) \in Q(B).$$

Proof. (1) Assume that $A \xrightarrow{k} B$, $\bar{a} \in Q(A)$, $Q \in \mathrm{UCQ}$, and $Q := \bigvee_{1 \le i \le k} Q_i$ where each $Q_i \in \mathrm{CQ}$. Then $\bar{a} \in Q_i(A)$ for some i and this happens iff there is a homomorphism $g : Q_i \to A\bar{a}$, that is a homomorphism from Q_i to A which maps the free variables $\bar{x}$ of Q_i to $\bar{a}$. Since $|g(Q_i)| \le k$, we have $A' \sqsubseteq A$ and homomorphism $h : A' \to B$. But then $h \circ g : Q_i \to Bh(\bar{a})$ and therefore $h(\bar{a}) \in Q(B)$. The proof of 2, 3, and 4 is similar to the corresponding parts in Theorem 3.24. □

Theorem 3.25. *If W is a k-universal set solution for S under Σ for*

1. *homomorphisms and $Q \in \mathrm{CQ}_k$,*
2. *injective homomorphisms and $Q \in \mathrm{MonQ}_k$,*
3. *full homomorphisms and $Q \in \mathrm{UCQ}_k^{\neg}$, or*
4. *embeddings and $Q \in \mathrm{UCQ}_k^{\neg,\neq}$,*

and Q has arity r, then

$$\mathrm{cert}_Q^{\Sigma}(S) = \mathrm{dom}(S)^r \cap \bigcap_{T \in W} Q(T).$$

Proof. (sketch) Similar to the proof of Theorems 3.1 and 3.2 using Theorem 3.24 instead of Theorem 3.17. □

Theorem 3.26. *For every k, there is a set Σ_{k+1} of TGDs such that Σ has a weak template and a strong k-template, but no strong $(k+1)$-template*

Proof. (sketch) Consider the set of axioms Σ_{k+1}:

$$\begin{array}{lllll} \xi_1\text{:} & & & & \exists x, y\, E(x,y) \\ \xi_2\text{:} & E(x,y) & & \rightarrow & \exists z\, E(y,z) \\ \xi_3^{k+2}\text{:} & E(u_0,u_1), \ldots, E(u_{k+1},u_{k+2}) & & \rightarrow & E(u_0,u_{k+2}) \end{array}$$

$\{C^{k+1}\}$ is a weak template for Σ_{k+1} and also a strong k-template. If $G \models \Sigma_{k+1}$, then G must contain a cycle of length at most $k+1$, so it can not be a strong $(k+1)$-template. □

To compute k-templates for Σ when Σ is closed under products, we can use the following variant of the chase. Compute a chase sequence $\emptyset = A_0, A_1, \ldots$ using some chase-like algorithm, for example the unordered minimizing chase. Simulatenously, compute a chain of models of Σ: $B_0 \leftarrow B_1 \leftarrow \ldots$. Such a chain can be obtained, for example, by picking some enumeration of the models of Σ and setting B_n to be the product of the first $n+1$ models. Alternatively, given B_n, we can set $B_{n+1} := M_{n+1}$ were M_{n+1} is the $(n+1)$th model in this enumeration in case $M_{n+1} \rightarrow B_{n+1}$ and $B_{n+1} := B_n$ otherwise. If we find some n, m such that $B_m \xrightarrow{k} A_n$ we stop. The result is B_m, which has the desired universality property since $B_m \xrightarrow{k} A_n \rightarrow \mathrm{mod}(\Sigma)$ by Theorem 3.10. This can be generalized to universal-existential constraints by considering the situation where each A_i and B_j is a finite set of instances instead of a single instance. We can show the following.

Theorem 3.27. *If there is a strong k-template for Σ, then the procedure above using the unordered minimizing chase will terminate and find it.*

This chapter is based on "Data Exchange, Data Integration, and the Chase" by Alan Nash, Alin Deutsch, and Jeff Remmel [34]. I was responsible for developing all the concepts in this paper, except as follows. Jeff Remmel and Alin Deutsch contributed towards the proof of completeness, towards the overall presentation, introduced the unordered minimizing chase, and participated in many discussions in which the concepts discussed here were clarified.

4

The Data Exchange Core Computation Problem

Data exchange deals with inserting data from one database into another database having a different schema. Fagin, Kolaitis, and Popa have shown that among the universal solutions of a solvable data exchange problem, there exists – up to isomorphism – a most compact one, "the core", and have convincingly argued that this core should be the database to be materialized. They stated as an important open problem whether the core can be computed in polynomial time in the general setting where the mapping between the source and target schemas is given by source-to-target constraints which are arbitrary tuple generating dependencies (TGDs) and target constraints consisting of equality generating dependencies (EGDs) and weakly-acyclic TGDs. In this chapter we solve this problem by developing new methods for efficiently computing the core of a universal solution. This positive result shows that data exchange based on cores is feasible and applicable in a very general setting.

4.1 Computing the Core of a Universal Solution: Outline

In this section, we outline the methods for core computation used in previous work and in the next few sections of this chapter.

Previous Results. Fagin, Kolaitis, and Popa [12] proved that the core of a universal solution can be computed in polynomial time in two restricted settings:

- When the set Σ_t of target constraints is empty.
- When the set Σ_t of target constraints contains EGDs only.

They provided two different methods to obtain these results, of which one is directly relevant to our present work, namely, the *blocks method*. This method is based on the observation that the Gaifman graph of the variables of the result T of applying the source-to-target TGDs to a ground source instance S consists of connected components whose size is bounded by a constant b. Such instances T have a nice property: checking whether there is a homomorphism from any $T \in K$ where K is any set of instances with such bound b into any arbitrary other instance T' is feasible in polynomial time. In fact, this test essentially boils down to check whether each of these blocks has a homomorphism to T'. They give the following result in somewhat different form.

Theorem 4.1 ([12]).

1. *If Σ is a set of source-to-target constraints of height e, S is ground, and $(S,T) = (S,\emptyset)^{\Sigma}$, then* $\text{blocksize}(T) \leq e$.

2. *If* $\text{blocksize}(A) \leq c$, *then we can check whether $A \to B$ holds in time $O(|B|^c)$.*

The core of T can be obtained by checking whether T admits a proper endomorphism h and, if so, replacing T by $h(T)$. This process is repeated until T cannot be further shrunk via endomorphisms. The result is the core.

Fagin et al. then extended this method to the case where Σ_t consists of EGDs. The difficulty here is that EGDs, when applied, can merge blocks by equating variables from different blocks. Thus, after chasing EGDs over T, the result T^{Σ_t} has, in general, lost the bounded block-size property. However Fagin et al., by an insightful *Rigidity Lemma* [12] show that after equating a sequence of variables while enforcing EGDs, the final remaining variable is *rigid*, i.e., can be mapped only to itself in every endomorphism. The resulting instance T^{Σ_t} has thus the bounded block-size property if we treat such variables as constants.

In [19] Gottlob has shown that computing cores is tractable if the target dependencies Σ_t consist of *full* TGDs, i.e., TGDs without existentially quantified variables, and arbitrary EGDs. Note that a set of full TGDs is weakly acyclic.

For full TGDs the situation is rather complex. While $T = S^{\Sigma_{st}}$ has bounded block size, T^{Σ_t} has in general neither bounded block size nor rigid variables. In fact, while T^{Σ_t} contains no additional variables, different blocks of T can be merged through the creation of new atoms. A full TGD of the form $r(x,y) \wedge r(z,t) \to r(y,t)$ may obviously merge blocks. This situation is depicted in figure 4.1, where the original blocks of T are depicted as ovals, and where some new atoms created via full TGDs are depicted as

black boxed. These atoms may connect previously separate blocks and as a result, very large blocks may arise.

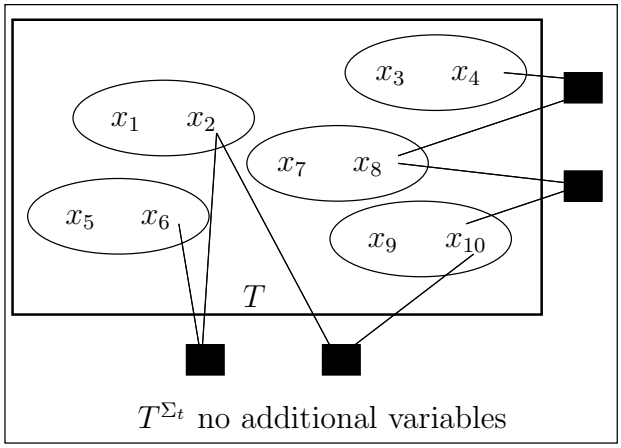

Figure 4.1 Structure of the target instance T^{Σ_t} in case Σ_t consists of full TGDs only

Generalizing the results of [12], it was shown in [19] that for checking whether a universal solution $B \subseteq T^{\Sigma_t}$ is not yet the core, it suffices to look for a homomorphism $h : T \to B$ such that $h(T) \subset B$.[1] This is because it was further shown that this homomorphism $h : T \longrightarrow B$ is actually a non-injective endomorphism of T^{Σ_t} (recall that T and T^{Σ_t} have the same domains). Moreover, h can be transformed in polynomial time into a non-injective *retraction* r of T^{Σ_t} satisfying $r(T^{\Sigma_t}) \subset B$. This ensures that $B' = r(T^{\Sigma_t}) \subset B$ satisfies Σ_t. If such a mapping h exists, it can be found in polynomial time by exploiting the bounded block-size of T.

We needed to consider retractions and not arbitrary endomorphisms, because for a retraction r it holds that $r(T^{\Sigma_t}) \models \Sigma_t$, while this is not always true for endomorphisms in general (as will be made clear through Example 4.1 in Section 4.2). Starting from $B = T^{\Sigma_t}$, by successively replacing B with B' as described, we eventually reach the core.

This tractability result was then extended in [19] to the setting where Σ_t contains EGDs in addition to full TGDs. This was achieved by a simulation of EGDs through full TGDs. Note that with full TGDs and EGDs we can express functional, join, and multivalued dependencies, but not inclusion dependencies or foreign key constraints.

Another relevant class of data exchange problems arises when the set Σ_t of target constraints is restricted to contain weakly acyclic *simple TGDs* and arbitrary

[1] In [12] similar mappings where called *useful*.

EGDs. A simple TGD is one whose left side consists of a single atom with no repeated variables. In [19] it was shown that for this class of problems, the core can be computed in polynomial time, too. The proof is based on the observation that chasing with simple TGDs does not change hypertree width [21]. This class is practically relevant, because it covers as target constraints the important class of functional dependencies and acyclic inclusion dependencies, and thus also foreign key constraints (as long as they do not destroy weak acyclicity). However it does not include multivalued or join dependencies. Fagin [14] has shown tractability for a slightly larger class by completely different means.

The tractability results of the present chapter subsume those of [19] and are more general. Therefore we will not further discuss the specific classes of tractable data exchange problems studied in [19].

Outline of Main New Ideas for Tractability. We outline the main ideas underlying our solution of the core computation problem for the general case, when the target constraints Σ_t may consist of weakly-acyclic TGDs and arbitrary EGDs. Such constrains encompass all major types of data dependencies. We will first deal with weakly-acyclic TGDs as target constraints and then show how EGDs can be added.

For weakly-acyclic TGDs the situation is yet more complicated than for full TGDs. Again, let T denote a target instance obtained by chasing the source instance S with the source-to-target constraints Σ_{st} (for an arbitrary chase order). As before, T has bounded block-size. However, while in the case of full TGDs, T already contained *all* variables of T^{Σ_t}, now there can be further variables outside T and these variables can appear in large (unbounded) blocks of T^{Σ_t}, see Figure 4.2. We cannot proceed,

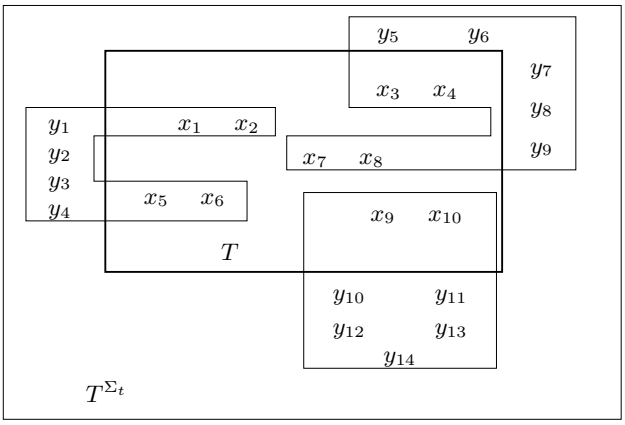

Figure 4.2 Structure of the target instance T^{Σ_t} in case Σ_t consists of weakly acyclic TGDs

as in the case of full TGDs, by trying to find a non-injective homomorphism h from T to some intermediate instance $B \subset T^{\Sigma_t}$ and hope that h extends automatically to a non-injective endomorphism of $T^{\Sigma_t} \to B$. In fact, there may not exist any non-injective homomorphism $T \to B$, while there may well exist a non-injective endomorphism h of $T^{\Sigma_t} \to B$ such that $h(T^{\Sigma_t}) \subset B$, where $h(x) = h(y)$ for two distinct values x and y that are *not both in* T. But variables outside T may appear in large blocks and it is thus not at all obvious how homomorphisms involving them can be found in polynomial time.

Our solution of this problem is based on the following ideas.

Idea 1. For technical reasons, we compute *non-injective retractions* $T^{\Sigma_t} \to B$ rather than arbitrary (i.e., not necessarily idempotent) non-injective endomorphisms. The reason is that if h is a retraction, then $h(T^{\Sigma_t}) \models \Sigma_t$ (Theorem 3.5). Note that this is *not true* for arbitrary endomorphisms (see Example 4.1). In order to find a non-injective retraction, we can always find a non-injective endomorphism of T^{Σ_t} first and then transform it in polynomial time into a suitable retraction (Theorem 4.2).

Recall that if x is a variable of T^{Σ_t} that was introduced at some chase step via an existential TGD ξ from Σ_t, then the *parents* of x are all values occurring in the atoms that made ξ fire and the *siblings* are all other new variables introduced at the same chase step. Recall further, that the ancestors of a variable are defined in the usual way by the transitive closure of the parent relation.

Idea 2. We observe that each variable of T^{Σ} has a bounded set of ancestors and a bounded set of siblings of ancestors. Our idea is to exploit this fact.

The next idea deals with *how* to exploit the bounded set of ancestors.

Idea 3. Assume that we have already constructed a retraction $h : T^{\Sigma_t} \to B \subseteq T^{\Sigma_t}$ and we want to see whether B can be further shrunk. This means that we need to see whether two distinct values x, y of B can be further "lumped together" by a homomorphism h' such that $h'(x) = h'(y)$. To this aim, we define, for all pairs of distinct values x, y of B, the sub-instance $T_{xy} \subset T^{\Sigma_t}$ which contains all atoms over the set of values of T, x y, and their siblings, and all ancestors of x and y and the siblings of these ancestors. Given that T has bounded block-size and that the number of ancestors and siblings of each variable is bounded by a constant, these sets T_{xy} all have bounded block-size. We can then check for each T_{xy} in polynomial time whether there is a homomorphism $T_{xy} \to B$ such that x and y are mapped to the same element z, see Figure 4.3.

If this is possible for some T_{xy}, then this homomorphism h can be extended in

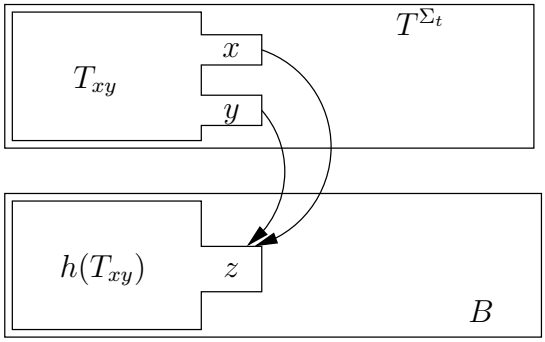

Figure 4.3 Improvement of a homomorphism

polynomial time to an non-injective endomorphism $h : T^{\Sigma_t} \to B$ such that $h(T^{\Sigma_t}) \subset B$. Then, as explained before, from such a h we can compute in polynomial time a retraction h' of T^{Σ_t} such that $h'(T^{\Sigma_t}) \subset h(B) \subset B$ and thus B can be replaced by a smaller instance $h(B)$ which also satisfies Σ_t. Otherwise B is already the core.

In order to establish that each homomorphism $T_{xy} \to B \subseteq T^{\Sigma_t}$ mapping x and y to the same element can be extended in polynomial time to to an endomorphism $T^{\Sigma_t} \to B$ such that $h(B) \subset B$, we show a slightly stronger result, described in Idea 4.

Idea 4. We show that whenever a subset A of T^{Σ_t} contains T and is closed under ancestors and siblings, and whenever B satisfies Σ_t, then any homomorphism $h : A \to B$ can be extended in polynomial time to an homomorphism from $T^{\Sigma_t} \to B$. (Theorem 4.5)

These are the four main new ideas we used in order to show that the core can be computed in polynomial time in case Σ_t consists of TGDs only. To cover, in addition EGDs, we need two further ideas

Idea 5. We simulate EGDs by full TGDs by introducing an additional binary relation E which stands for 'equal' and by adding some consistency rules which say that values which are marked as equal in the E relation are indistinguishable by the other relations of the target database.

This simulation introduces a big new problem. Adding the new full TGDs may create new cycles and will in general yield a set of TGDs which is not weakly acyclic. Therefore, there is a risk that the chase will not terminate. Here comes our final idea that solves this problem, too.

Idea 6. We observe that for a particular chase order which can be statically determined, TGDs with existentially quantified variables will never fire on premises contain-

ing variables whose ancestor-tree exceeds a certain depth. Thus the chase terminates in polynomial time, and the crucial bounded-ancestors property is still guaranteed.

This concludes the rather superficial presentation of our main ideas. In the rest of this chapter we will make these ideas more concrete and provide the glue for putting them together.

4.2 Retractions and Cores

We have seen in Theorem 3.5 that embedded dependencies are closed under retractions. On the other hand, even full dependencies are not closed under endomorphisms, as the following example shows.

Example 4.1. Assume A is an instance with a single binary relation R containing the tuples $\{(x,z),(x,a),(z,y),(a,z),(a,a)\}$ where x,y,z are variables and a is a constant, Σ consists of the single constraint

$$R(u,w), R(w,w), R(w,v) \rightarrow R(u,v),$$

and $h(x) = x, h(y) = z, h(z) = a, h(a) = a$. Then $A \models \Sigma$ and h is an endomorphism of A, but $h(A)$, which consists of R with tuples $\{(x,a),(a,z),(a,a)\}$ does not satisfy Σ since R does not contain (x,z).

In general, one can obtain the core of an instance A by successively applying non-surjective endomorphisms. However, if an instance A satisfies some constraints Σ, then even though its core C satisfies Σ by Theorem 3.5, the image $h(A)$ of A under an endomorphism h may not satisfy Σ as Example 4.1 shows. In Theorem 4.7 we will compute the core of an instance U which satisfies some constraints Σ by computing a chain $U = U_0 \supset U_1 \supset U_2 \supset \ldots \supset U_n$, but we will need each U_i to satisfy Σ. Therefore, we will ensure that for each i, $U_i \hookrightarrow U_{i+1}$ and in order to do this, we use the following result.

Theorem 4.2. *Given an endomorphism $h : A \rightarrow A$ such that $h(x) = h(y)$ for some $x, y \in \mathrm{dom}(A)$, there is a proper retraction r on A such that $r(x) = r(y)$. Moreover, such retraction can be found in time $O(|\mathrm{dom}(A)|^2)$.*

Proof. Assume h is as in the hypothesis. We write h^q for the composition of h with itself q times. Since h is an endomorphism, $h(A) \subseteq A$. Since also $B \subseteq A$ implies $h(B) \subseteq h(A)$, it follows that $h^1(A) \supseteq h^2(A) \supseteq h^3(A) \supseteq \ldots$ This sequence can not decrease forever,

so there must be some $q \leq |\mathrm{dom}(A)|$ such that $h \circ h^q(A) = h^q(A)$. Set $g := h^q$ and $B := g(A)$. Then $h(g(A)) = g(A)$, and thus $g(B) = h^q(g(A)) = g(A) = B$. That is, g is an automorphism on B. Denote by G the graph of g on B, i.e., the digraph whose set of vertices is B and whose set of arcs is $\{(u, v) : u, v \in B \wedge g(u) = v\}$. Since g is a permutation, G consists of a collection $C_1, C_2, \ldots, C_k$ of disjoint cycles or loops C_i of respective lengths $c_1, \ldots, c_k$. These cycles correspond to the strongly connected components of G and can thus be identified in linear time. Obviously, we have

$$\Sigma_{i=1}^{k} c_i = |\mathrm{dom}(B)| \leq |\mathrm{dom}(A)|.$$

Let $g_0 := g$ and for $0 \leq i \leq k-1$, let $g_{i+1} := g_i^{c_i}$. Set $r := g_k$. We thus have:

$$r = ((((g^{c_1})^{c_2})^{c_3}) \cdots)^{c_k} = g^{c_1 \cdot c_2 \cdots c_k}.$$

Note that for $1 \leq i \leq k$, g_i is the identity on the vertices of all cycles $C_1, \ldots, C_i$. Thus r restricted to B is the identity on B. Moreover $r(A) = g(A) = B$, so r is a retraction of A. Note that r, arising from compositions of endomorphisms, is itself an endomorphism. Furthermore, $r = (h^q)^{c_1 \cdot c_2 \cdots c_k}$ and therefore, since $h(x) = h(y)$, it also holds that $r(x) = r(y)$. Thus r is the desired retraction. Computing r as described above requires no more than $|\mathrm{dom}(A)| + \Sigma_{i=1}^{k} c_i \leq 2|\mathrm{dom}(A)|$ compositions. Each single composition is feasible in linear time in $|\mathrm{dom}(A)|$ and therefore r can be done in time quadratic in $|\mathrm{dom}(A)|$. □

4.3 Weakly-Acyclic TGDs

In this section we prove the following result.

Theorem 4.3. *For every $\Sigma := \Sigma_{st} \cup \Sigma_t$ where*

- *Σ_{st} is a set of source-to-target embedded dependencies and*
- *Σ_t is a weakly-acyclic set of TGDs*

and every ground instance S, a core of a universal solution U for S under Σ can be computed in time $O(|\mathrm{dom}(S)|^b)$ for some b which depends only on Σ.

In order to prove Theorem 4.3, we need several intermediate results. In the proof we chase $(S, \emptyset)$ with Σ_{st} to obtain (S, T), then chase T with Σ_t to obtain U. Then (idea 1) we compute the core of U by successively applying proper retractions

(Theorem 4.7) instead of non-surjective endomorphisms. In order to find such proper retractions efficiently, we (idea 3) identify a set of fragments T_{xy} of U: one such fragment T_{xy} only slightly larger (idea 2) than T satisfying $T \subseteq T_{xy} \subseteq U$ for every pair of distinct values $x, y \in \mathrm{dom}(U)$ (Lemma 4.4). Finding a homomorphism from T_{xy} to any instance B is easy, since it is easy for T. Furthermore (idea 4), such a homomorphism $T_{xy} \to B$ can be extended to a homomorphism $U \to B$ when $B \models \Sigma$ (Theorem 4.5). This is enough to show that we can check efficiently whether any $U' \subseteq U$ satisfying $U' \models \Sigma$ has a proper retraction (Theorem 4.6) since U' has a proper retraction iff there are distinct values $x, y \in \mathrm{dom}(U')$ such that there is a homomorphism $T_{xy} \to U'$. Notice that we need $U' \models \Sigma$ and therefore we compute the core by successively applying proper retractions (cf. Theorem 3.5) instead of non-surjective endomorphisms (cf. Example 4.1). At the end of this section, we present an algorithm that performs all these steps.

The following lemma corresponds to idea 2 in Section 4.1.

Lemma 4.4. *For every weakly-acyclic set Σ of TGDs of depth d, width w, and height e, instance T, and $x, y \in \mathrm{dom}(T^\Sigma)$, there is T_{xy} satisfying*

1. $x, y \in \mathrm{dom}(T_{xy})$,
2. $|\mathrm{dom}(T_{xy})| \leq |\mathrm{dom}(T)| + 2edw^d$,
3. $T \subseteq T_{xy} \subseteq T^\Sigma$,
4. $\mathrm{dom}(T_{xy})$ *is closed under parents and siblings, and*
5. T_{xy} *can be computed in time $O(|\mathrm{dom}(T)|^c)$ for some c which depends only on Σ.*

Proof. Assume Σ, x, y, and T satisfy the hypotheses. Then every value in T^Σ has depth at most d. If x has nonzero depth, then it was introduced into relations in T^Σ by some step of the chase, which must have fired on some set of tuples of T^Σ.

For any x, set A_x to be x and all its ancestors and E_x to be A_x and all siblings of elements in A_x. That is, E_x is the smallest set containing x which is closed under parents and siblings. An easy induction on depth shows that $|A_x| \leq dw^d$ and $|E_x| \leq edw^d$. We can compute A_x and E_x in time $O(|\mathrm{dom}(T)|^d)$ where d depends only on Σ. Similarly, compute A_y and E_y. Set $T_{xy} := T \cup (T^\Sigma | (E_x \cup E_y))$. Clearly, T_{xy} can be computed in time $O(|\mathrm{dom}(T)|^c)$ for some c that depends only on Σ and it satisfies requirements 1 through 4. □

Example 4.2. Consider the set Σ consisting of the single constraint

$$E(x, y), E(y, z) \to \exists u, v\, F(x, u), F(u, v), F(v, z)$$

which says that for every path of length 2 in E there is a path of length 3 in F. Consider the instance T which consists of a path $c_1, c_2, c_3, c_4, c_5, c_6$ of length 5 in E. Then T^Σ contains, in addition to this path of length 5 in E, two paths of length 6 in F: $c_1, v_1, v_2, c_3, v_3, v_4, c_5$ and $c_2, v_5, v_6, c_4, v_7, v_8, c_6$. Here v_1 and v_2 are siblings and c_1, c_2 and c_3 are their parents. Therefore, if $x = v_1$, then $A_x = \{c_1, c_2, c_3, v_1\}$ and $E_x = \{c_1, c_2, c_3, v_1, v_2\}$. Similarly, if $y = v_7$, then $A_y = \{c_4, c_5, c_6, v_7\}$ and $E_y = \{c_4, c_5, c_6, v_7, v_8\}$. Therefore, T_{xy} consists of the path of length 5 in E: $c_1, c_2, c_3, c_4, c_5, c_6$ and two path of length 3 in F: c_1, v_1, v_2, c_3 and c_4, v_7, v_8, c_6.

Notice that even though, as we discuss in the proof of Lemma 4.4, $|A_x| \leq dw^d$ (that is, the set of ancestors of x is bounded by dw^d), this does not mean that every tuple $\bar{b}$ introduced by the chase has a "bounded proof length" in the sense that only a bounded number of tuples $\bar{b}_1, \ldots, \bar{b}_k$ are needed to force the introduction of $\bar{b}$. To see this, consider for example the single constraint

$$E(x, y), E(y, z) \rightarrow E(x, z)$$

which says that E is transitively closed. Since this is a full TGD, no variables are added during the chase and therefore, $A_x = E_x = \emptyset$. However, there are E-paths of arbitrarily large length which cause the introduction of a pair into E and therefore such pairs have unbounded proof length.

The following theorem corresponds to idea 4 in Section 4.1.

Theorem 4.5. *If Σ is a set of weakly-acyclic TGDs, and B, T, and W are instances satisfying*

1. $B \models \Sigma$,
2. $T \subseteq W \subseteq T^\Sigma$, *and*
3. $\mathrm{dom}(W)$ *is closed under ancestors and siblings,*

then any homomorphism $h : W \rightarrow B$ can be extended in time $O(|\mathrm{dom}(T)|^b)$ to a homomorphism $h' : T^\Sigma \rightarrow B$ where b depends only on Σ.

Proof. Assume 1, 2, and 3 hold and there is a homomorphism $h : W \rightarrow B$. Then h can be extended to the desired h' in time $O(|\mathrm{dom}(T)|^b)$ where b depends only on Σ as follows. Assume that the chase of T with Σ terminates in t steps; that is, $T^\Sigma = T^\Sigma_t$. We will compute a sequence of homomorphisms $h_0 := h \subseteq h_1 \subseteq \ldots \subseteq h_t$ such that $h_s : T_s \rightarrow B$ where $T_s := T^\Sigma_s \cup W$. Since $T^\Sigma_0 = T \subseteq W$, the homomorphism $h_0 = h$ is

a homomorphism $T_0 = W \to B$. Since $W \subseteq T^\Sigma$ we have $T^\Sigma = T_t^\Sigma = T_t$ and therefore $h' := h_t$ is the desired extension.

To obtain the homomorphism h_{s+1} from the homomorphism h_s we proceed as follows. Assume that T_{s+1}^Σ is obtained from T_s^Σ by firing a constraint ξ of the form

$$\phi(\bar{u}) \to \exists \bar{v}\, \psi(\bar{u}, \bar{v})$$

on $\bar{a} \in \mathrm{dom}(T_s^\Sigma)$ where in the case of full TGDs $\bar{v}$ is empty. That is $T_s^\Sigma \models \phi(\bar{a})$ and $T_{s+1}^\Sigma \models \psi(\bar{a}, \bar{b})$ for some $\bar{b} \in \mathrm{dom}(T_{s+1}^\Sigma) - \mathrm{dom}(T_s^\Sigma)$. Since h_s is a homomorphism $T_s \to B$ and ϕ is monotonic, $B \models \phi(h_s(\bar{a}))$. The only tuples introduced into relations in T_{s+1}^Σ are those in $\psi(\bar{a}, \bar{b})$. Therefore it is sufficient to define $h_{s+1} \supseteq h_s$ so that $B \models \psi(h_{s+1}(\bar{a}), h_{s+1}(\bar{b}))$.

If ξ is a full TGD, we set $h_{s+1} := h_s$. Since $B \models \Sigma$, we have $B \models \psi(h_s(\bar{a}))$. Otherwise, since T_s is closed under siblings, there are only two cases to consider:

1. $\bar{b} \in \mathrm{dom}(T_s)$ and
2. $\bar{b} \notin \mathrm{dom}(T_s)$.

In case (1) we set $h_{s+1} := h_s$. Since $\bar{b} \in \mathrm{dom}(T_s) - \mathrm{dom}(T_s^\Sigma)$ and $T_s = T_s^\Sigma \cup W$, we must have $\bar{b} \in \mathrm{dom}(W)$. Since W is closed under parents, $\bar{a} \in \mathrm{dom}(W)$ and therefore $W \models \psi(\bar{a}, \bar{b})$. Since $h_0 : T_0 = W \to B$ is a homomorphism, $B \models \psi(h_0(\bar{a}), h_0(\bar{b}))$ and therefore, since $h_{s+1} \supseteq h_0$, $B \models \psi(h_{s+1}(\bar{a}), h_{s+1}(\bar{b}))$.

In case (2) we set $h_{s+1}(x) := h_s(x)$ for any $x \in \mathrm{dom}(T_s)$ and $h_{s+1}(\bar{b}) := \bar{c}$ for some $\bar{c}$ such that $B \models \psi(h_s(\bar{a}), \bar{c})$. Such $\bar{c}$ exists because $B \models \Sigma$. Then $B \models \psi(h_{s+1}(\bar{a}), h_{s+1}(\bar{b}))$.

Since Σ is weakly-acyclic then, by Theorem 2.1 t is $O(|\mathrm{dom}(T)|^p)$ for some p which depends only on Σ and this implies that the extension $h' = h_t$ can be obtained in time $O(|\mathrm{dom}(T)|^b)$ where b depends only on Σ. □

Theorem 4.6. *For any weakly-acyclic set Σ of TGDs and instance T, we can check whether any retract U' of $U = T^\Sigma$ has a proper retraction (i.e., whether U' is not a core) and find it in time $O(|\mathrm{dom}(T^\Sigma)|^b)$ where b depends only on Σ and* blocksize(T).

Proof. For every $x, y \in \mathrm{dom}(U')$, compute T_{xy} with the properties given in Lemma 4.4 and test whether there is a homomorphism $h : T_{xy} \to U'$ such that $h(x) = h(y)$. Then U' has a proper retraction iff there are such x, y, h by Claims 1 and 2 below.

Such T_{xy} exist and can be computed in time $O(|\mathrm{dom}(T)|^c)$ for some c which depends only on Σ by Lemma 4.4. Therefore, since there are at most $|\mathrm{dom}(U')|^2$ pairs (x, y), the result follows from Claims 3 and 4 below.

Claim 1: *r is a proper retraction on U' iff there are $x, y \in \mathrm{dom}(U')$ such that $r(x) = r(y)$.* This is obvious.

The following claim corresponds to idea 3.

Claim 2: *If $x, y \in \mathrm{dom}(U')$, then there is a homomorphism $h : T_{xy} \to U'$ such that $h(x) = h(y)$ iff there is a retraction r on U' such that $r(x) = r(y)$.*

Proof. Since U' is a retract of U and $U \models \Sigma$, we have $U' \models \Sigma$ by Theorem 3.5. Therefore, if there is a homomorphism $h : T_{xy} \to U'$ such that $h(x) = h(y)$, then h can be extended to a homomorphism $h' : U \to U'$ by Theorem 4.5 (remember we have $U = T^\Sigma$ and we can set $W := T_{xy}$ and $B := U'$ to satisfy the hypotheses of the theorem). Then $h'' := h'|U'$ is an endomorphism of U' with $h''(x) = h''(y)$. By Theorem 4.2, there is a retraction r of U' such that $r(x) = r(y)$.

Conversely, if there is a retraction r on U' such that $r(x) = r(y)$, then since U' is a retract of U, we know that there is a retraction $r' : U \hookrightarrow U'$. If there is also a retraction r on U' such that $r(x) = r(y)$, then $r'' = r \circ r'$ satisfies $r''(x) = r''(y)$. Therefore $h := r''|T_{xy}$ is a homomorphism $T_{xy} \to U'$ satisfying $h(x) = h(y)$.

Claim 3: *Given $x, y \in \mathrm{dom}(U')$ and a homomorphism $h : T_{xy} \to U'$ such that $h(x) = h(y)$, a retraction r on U' such that $r(x) = r(y)$ can be found in time $O(|\mathrm{dom}(U)|^c)$ for some c which depends only on Σ.*

Proof. This follows directly from the proof of Claim 2, since Lemma 4.4, Theorem 4.5, and Theorem 4.2 guarantee that h', h'', and r can all be found in time $O(|\mathrm{dom}(U)|^{c'})$ for some c' which depends only on Σ.

Claim 4: *Checking whether, for any $x, y \in \mathrm{dom}(U')$, there is a homomorphism $h : T_{xy} \to U'$ such that $h(x) = h(y)$ (and if so, finding it) can be done in time $O(|\mathrm{dom}(U)|^{c''})$ for some c'' which depends only on Σ and blocksize(T).*

Proof. Set $s :=$ blocksize(T). Since $|\mathrm{dom}(T_{xy})| \leq |\mathrm{dom}(T)| + 2edw^d$ by Lemma 4.4, the set

$$\{T_{xy} : x, y \in \mathrm{dom}(U'), T \in K\}$$

has block size bound $s + 2edw^d$ and therefore Theorem 4.1 implies the claim except for the additional requirement that $h(x) = h(y)$. Handling this additional requirement is straightforward. □

Theorem 4.7. *For every s and weakly-acyclic set Σ of TGDs, there is b such that for any T with* $\text{blocksize}(T) \leq s$, *the core of T^Σ can be computed in time $O(|\text{dom}(T)|^b)$.*

Proof. Set $U := T^\Sigma$. By Theorem 2.1, U can be computed in time $O(|\text{dom}(T)|)^d$ for some d that depends only on s and Σ. To compute the core C of U efficiently we set $U_0 := U$ and we compute a sequence $U_0, U_1, \ldots, U_n$ such that

1. $U_0 \supset U_1 \supset \ldots \supset U_n$,
2. $U_0 \hookrightarrow U_1 \hookrightarrow \ldots \hookrightarrow U_n$,
3. $U_0, U_1, \ldots, U_n \models \Sigma$, and
4. U_n is a core.

Then U_n is the core of U. Given U_m satisfying 1 and 2 above, we compute U_{m+1} as follows. We check whether U_m has a proper retraction r and find it. By Theorem 4.6, this can be done in time $O(|\text{dom}(U)|^c)$ for some c which depends only on s and Σ. If so, we set $U_{m+1} := r(U_m)$. Then U_{m+1} satisfies 1 and 2 above. Since Σ is closed under retractions by Theorem 3.5, U_{m+1} also satisfies 3 above. If U_m has no proper retraction, set $n := m$. By 1, we must have $n \leq |\text{dom}(U)|$ and therefore we can compute U_n, the core of U, in time $O(|\text{dom}(U)|^{c+1})$ from which the result follows. □

Proof. (**Theorem 4.3**) Assume Σ and S satisfy the hypotheses. First compute $(S, \emptyset)^{\Sigma_{st}}$, which is equal to (S, T) for some T. Next compute $U := T^{\Sigma_t}$. By Theorem 2.1, $(S, \emptyset)^{\Sigma_{st}}$ and T^{Σ_t} are well defined and can be computed in time $O(|\text{dom}(S)|^c)$ where c depends only on Σ. We have

$$(S, \emptyset)^\Sigma = (S, \emptyset)^{\Sigma_{st}, \Sigma_t} = (S, T)^{\Sigma_t} = (S, T^{\Sigma_t}) = (S, U)$$

and therefore, by Theorem 2.2, U is a universal solution for S under Σ. By Theorem 4.1 for any fixed Σ, the set of all T obtained as above has bounded block size. Therefore, by Theorem 4.7 there exists b such that for any S the core of U can be computed in time $O(|\text{dom}(T)|^b)$. The result follows from Theorem 2.1. □

The procedure FindCore computes the core of a universal solution for a ground instance S under a set Σ as above in time $O(|\text{dom}(S)|^b)$ for some b which depends only on Σ. The correctness and efficiency of FindCore follow from Theorem 4.3 above.

Procedure FindCore

Input: Source ground instance S of a data exchange problem

Output: Core of a universal solution for S

1. Chase $(S, \emptyset)$ with Σ_{st} to obtain $(S, T) = (S, \emptyset)^{\Sigma_{st}}$.
2. Chase T with Σ_t to obtain $U := T^{\Sigma_t}$.
3. For every x, y where $x \in \mathrm{var}(U)$, $y \in \mathrm{dom}(U)$, and $x \neq y$ do
4. Compute T_{xy} as in Lemma 4.4.
5. Look for a homomorphism $h : T_{xy} \to U$ such that $h(x) = h(y)$.
6. If there is such h, then
7. Extend h to an endomorphism h' on U. (Theorem 4.5)
8. From h', compute a proper retraction r on U. (Theorem 4.2)
9. Set $U := r(U)$.
10. Repeat from step 3.
11. Return U.

The following theorem states a concrete upper bound for the runtime of the FindCore algorithm. We believe that there is room for improvements regarding both the algorithm itself and its analysis. In particular, the algorithm can be improved by a factor of $|\mathrm{dom}(T^{\Sigma_t})|$ because for every pair $x, y \in \mathrm{dom}(T^{\Sigma_t})$ we need to check only once whether there is a homomorphism $h : T_{xy} \to U$ such that $h(x) = h(y)$, instead of once for every iteration starting at step 3. This is because if there is no such homomorphism $h : T_{xy} \to U$, then there can not be such homomorphism $h : T_{xy} \to U'$ for any $U' \subseteq U$. To simplify the presentation, we do not incorporate this improvement into FindCore.

Theorem 4.8. *For every $\Sigma := \Sigma_{st} \cup \Sigma_t$ where*

- *Σ_{st} is a set of source-to-target embedded dependencies and*
- *Σ_t is a weakly-acyclic set of TGDs of depth d, width w, and height e*

every variable-free source instance S and T such that $(S, T) = (S, \emptyset)^{\Sigma_{st}}$ a core of the universal solution $U = T^{\Sigma_t}$ for S under Σ can be computed in time

1. *$O(t^{s+4})$ and*
2. *$O(a|\tau|^2|\Sigma_t|p^{2r+e+w+3} + p^{s+4}))$*

where

- *p is either $|\mathrm{dom}(U)|$ or $(f|\mathrm{dom}(T)|)^{w^d}$ (a bound for $|\mathrm{dom}(U)|$),*

- *t is the time to compute T^{Σ_t},*
- *$|\tau|$ is the number of relation symbols in the target schema,*
- *$|\Sigma_t|$ is the number of constraints in Σ_t,*
- *a is the maximal number of atoms in any one constraint in Σ,*
- *$f = (e|\Sigma_t| + 1)$,*
- *r is the maximal arity of a relation symbol in τ, and*
- *$s = e + 2e(d + 2w^d)$ is a bound on the blocksize of T_{xy}.*

Finally, $|\mathrm{dom}(T)| \leq |\mathrm{dom}(S)| + p|\mathrm{dom}(S)|^q$ where p is the number of existentially quantified variables in the conclusions of Σ_{st} and q is the width of Σ_{st}. This gives a bound in terms of $|\mathrm{dom}(S)|$.

Proof. This was already proved in Theorem 4.3 except for the time bounds 1 and 2 which we give here. Our analysis follows the algorithm FindCore. We use the improved bound $|\mathrm{dom}(T_{xy})| \leq 2e(d + 2w^d)$, which is easy to verify.

Assume that $|\mathrm{dom}(U)| = n$ and the number of tuples in U is m. Then m is $O(|\tau|n^r)$ where r is the maximal arity of a symbol in τ. Notice that each step of the chase with Σ_t takes time

$$O(|\Sigma_t|n^w n^e am)$$

which is

$$O(a|\tau||\Sigma_t|n^{r+e+w})$$

where a is the maximal number of atoms in a constraint in Σ_t. This is because for each step we must check each constraint $\xi \in \Sigma$ and n^w possible tuples $\bar{a}$ to see whether ξ applies on $\bar{a}$. Furthermore, we need to check at most n^e tuples for the existentially-quantified variables in the conclusion. This consists of checking whether a atoms are in U, which has m tuples. Since at least one atom is introduced by each step of the chase with Σ_t and there are m tuples in U and m is $O(n^r)$, step 2 must execute in time t which is

$$O(a|\tau|^2|\Sigma_t|n^{2r+e+w})$$

We disregard step 1 whose time is insignificant compared to the rest of FindCore.

Notice that in order to compute T_{xy} it is enough to keep track of the ancestor and sibling graphs during steps 1 and 2, then given x and y, compute T_{xy} by following these graphs. Therefore step 4 takes time $O(n^2)$, since it is enough to iterate through U to find each element in T_{xy} that is not in T and $T_{xy} \subseteq U$. Notice also that the

block size of T_{xy} is bounded by the block size of T, which is $\leq e$ plus the number of elements added, which is $\leq 2e(d + 2w^d)$. Therefore, the block size of T_{xy} is bounded by $s := e + 2e(d + 2w^d)$. To check for a homomorphism from T_{xy} to U, it is enough to look at n^s possible setting for the variables in T_{xy} and check whether these map correctly into U. Therefore, the total time for step 5 is $|\mathrm{dom}(T_{xy})|n^s \leq n^{s+1}$. Step 7 essentially amounts to redoing the chase with Σ_t and therefore takes time $O(t)$. By Theorem 4.2, step 8 takes time $O(n^2)$. In summary,

Step 4	$O(n^2)$
Step 5	$O(n^{s+1})$
Step 7	$O(t)$
Step 8	$O(n^2)$

where t is the time to do the chase T^{Σ_t} and s is the bound on the block size of T_{xy}. Therefore one iteration of the inner loop (steps 4 through 9) takes time

$$O(t + n^{s+1}).$$

Since we repeat steps 4 through 9 for all values of x, y in U, we may go through n^2 iterations. Since line 10 may be executed at most n times, so we iterate the inner loop at most n^3 times. Lines 3 through 10 dominate the total time and therefore the total time for FindCore is

$$O(n^3(t + n^{s+1})).$$

Since n is $O(t)$, we can simplify the bound above to $O(t^3(t + t^{s+1}))$ which is $O(t^{s+4})$ where t is the time to compute T^{Σ_t}.

Alternatively we can get a more precise bound by substituting the value computed above for t which gives

$$O(n^3(a|\tau|^2|\Sigma_t|n^{2r+e+w} + n^{s+1}))$$

where $n = |\mathrm{dom}(U)|$.

Now we compute a bound for $|\mathrm{dom}(U)|$ in terms of $|\mathrm{dom}(T)|$. There are at most $e|\Sigma_t|$ existentially-quantified variables in conclusions of constraints in Σ_t. If $n_0 := |\mathrm{dom}(T)|$, then the number n_{i+1} of values of depth at most $i+1$ in U is at most $n_i + e|\Sigma_t|n_i^w$ since at most one value is introduced for every existentially quantified variable, of which there are at most $e|\Sigma_t|$ and every tuple on which a constraint fires, of which there are at most n_i^w. Therefore, $n_{i+1} \leq f n_i^w$ which implies that $n = n_d \leq f^{w^d} n_0^{w^d}$ and therefore

$|\mathrm{dom}(U)| \leq (e|\Sigma_t| + 1)^{w^d} |\mathrm{dom}(T)|^{w^d}$.

The bound for T in terms of S is computed similarly. Since Σ_{st} is source-to-target, T only has values of depth 0 and 1. □

4.4 Adding EGDs

In this section we prove the following result.

Theorem 4.9. *For every $\Sigma := \Sigma_{st} \cup \Sigma_t$ where*

- *Σ_{st} is a set of source-to-target embedded dependencies and*
- *Σ_t is a weakly-acyclic set of TGDs and EGDs.*

and every ground source instance S, a core of a universal solution U for S under Σ can be computed in time $O(|\mathrm{dom}(S)|^b)$ where b depends only on Σ.

In order to prove Theorem 4.9, we introduce a set $\bar{\Sigma}$ of TGDs which 'simulates' a set Σ of TGDs and EGDs (idea 5), in particular in the sense that the core of a solution for S under Σ is the same as the core of a solution for S under $\bar{\Sigma}$ (Lemma 4.10). If Σ is a weakly-acyclic set of TGDs and EGDs, $\bar{\Sigma}$ is not necessarily weakly-acyclic and Example 4.4 shows that the chase with $\bar{\Sigma}$ does not terminate on all orders. Nevertheless, we show that, for a certain class of chase orders which we call *nice* (idea 6), the chase is guaranteed to terminate whenever Σ is weakly-acyclic (Theorem 4.14). This is enough to complete the proof of Theorem 4.9.

The procedure to compute the core of a universal solution for S under a set Σ as in Theorem 4.3 is the same as FindCore (see Section 4.3) except that it includes an initial step to compute $\bar{\Sigma}$. After that, it proceeds with $\bar{\Sigma}$ instead of Σ. It appears towards the end of this section.

Given a set of TGDs and EGDs Σ over some signature τ, we define $\bar{\Sigma}$ over the signature $\tau \cup \{E\}$ where E is a new binary relation symbol by the following replacements on Σ.

1. Replace all equalities with E-atoms; that is, replace $x = y$ with $E(x, y)$.
2. Add the *equality* constraints:
   (a) $E(x, y) \rightarrow E(y, x)$
   (b) $E(x, y), E(y, z) \rightarrow E(x, z)$

(c) $R(x_1, \ldots x_k) \rightarrow E(x_i, x_i)$
for every $R \in \tau$ and $i \in \{1, \ldots, k\}$ where k is the arity of R

3. Add the *consistency* constraints

(d) $R(x_1, \ldots x_k), E(x_i, y) \rightarrow R(x_1, \ldots, y, \ldots x_k)$ for every $R \in \tau$ and $i \in \{1, \ldots, k\}$ where k is the arity of R and where y appears in the same position in R as x_i.

In step 1, EGDs are replaced by full TGDs which simulate them.

Example 4.3. If Σ consists of the following two constraints

$$\begin{aligned} D(x) &\rightarrow \exists y F(x, y) \\ F(x, y), F(x, z) &\rightarrow y = z \end{aligned}$$

then $\bar{\Sigma}$ consists of the following constraints

$$\begin{aligned} D(x) &\rightarrow \exists y F(x, y) && (4.1) \\ F(x, y), F(x, z) &\rightarrow E(y, z) && (4.2) \\ E(x, y) &\rightarrow E(y, x) && (4.3) \\ E(x, y), E(y, z) &\rightarrow E(x, z) && (4.4) \\ D(x) &\rightarrow E(x, x) && (4.5) \\ F(x, y) &\rightarrow E(x, x) && (4.6) \\ F(x, y) &\rightarrow E(y, y) && (4.7) \\ D(x), E(x, z) &\rightarrow D(z) && (4.8) \\ F(x, y), E(x, z) &\rightarrow F(z, y) && (4.9) \\ F(x, y), E(y, z) &\rightarrow F(x, z) && (4.10) \end{aligned}$$

Here, 1 and 2 are obtained from Σ by replacing equalities with E-atoms, 3-7 are equality constraints, and 8-10 are consistency constraints.

The auxiliary relation E is not essential; $\bar{\Sigma}$ can be replaced with the constraints obtained from $\bar{\Sigma}$ by removing the equality constraint (2b) and then using resolution through E on the remaining constraints. However, the presentation is easier with an explicit relation E.

Σ and $\bar{\Sigma}$ are over different signatures, but to simplify the presentation we will pretend that a model A of Σ also contains the relation E given by the identity relation on its domain. As a result of this convention, $A \models \Sigma$ implies $A \models \bar{\Sigma}$. Conversely, if E^A is the identity on $\mathrm{dom}(A)$ and $A \models \bar{\Sigma}$, then $A \models \Sigma$.

Lemma 4.10. *Under the hypotheses of Theorem 4.9, $\bar{\Sigma}$ consists only of source-to-target embedded dependencies and a set (not necessarily weakly-acyclic) of target TGDs such that C is the core of a universal solution for Σ iff C is the core of a universal solution for $\bar{\Sigma}$.*

Proof. If U is a universal solution for S under Σ, then U is also a solution for S under $\bar{\Sigma}$. Furthermore, U is universal for the solutions for S under $\bar{\Sigma}$ as follows. Assume that T is a solution for S under $\bar{\Sigma}$. Pick an element $\hat{x}$ from every equivalence class $[x]$ under E and define e so that $e(y) = \hat{x}$ whenever $y \in [x]$. It is easy to check that e is a retraction. Then $T' = e(T)$ is a solution for S under $\bar{\Sigma}$ and $E^{T'}$ is the identity on $\mathrm{dom}(T')$ and therefore T' is also a solution for S under Σ. Therefore, there is a homomorphism $h : U \to T' \subseteq T$, so h is also a homomorphism $U \to T$. This shows that U is also a universal solution for S under $\bar{\Sigma}$.

Conversely, assume U is a universal solution for S under $\bar{\Sigma}$. Define e as above for U instead of T. Then $U' = e(U)$ is a solution for S under Σ. Furthermore, U' is universal for the solutions for S under Σ as follows. Assume that T is a solution for S under Σ. Then T is also a solution for S under $\bar{\Sigma}$ and therefore there is a homomorphism $h : U \to T$. It follows that $h' := h|U'$ is a homomorphism $U' \to T$.

Then if C is the core of U, it must also be the core of U' since U' is a retract of U and therefore homomorphically equivalent to U. □

If Σ is a weakly-acyclic set of TGDs and EGDs, $\bar{\Sigma}$ is not necessarily weakly-acyclic. It is natural to wonder whether $A^{\bar{\Sigma}}$ is defined for any A; that is, whether the chase with $\bar{\Sigma}$ terminates *for all orders* as it does for Σ (see Theorem 2.1). The following example shows that this is not so.

Example 4.4. Assume that Σ consists of the constraints:

$R(x,y)$	$\rightarrow$	$\exists z\, T(x,y,z)$
$S(x,z)$	$\rightarrow$	$\exists y\, T(x,y,z)$
$T(x,y,z), T(x,u,v)$	$\rightarrow$	$u = y$
$T(x,y,z), T(x,u,v)$	$\rightarrow$	$v = z$

Then $\bar{\Sigma}$ consists of the constraints:

ξ_1	$R(x,y)$	$\rightarrow$	$\exists z\, T(x,y,z)$
ξ_2	$S(x,z)$	$\rightarrow$	$\exists y\, T(x,y,z)$
ξ_3	$T(x,y,z), T(x,u,v)$	$\rightarrow$	$E(u,y)$
ξ_4	$T(x,y,z), T(x,u,v)$	$\rightarrow$	$E(v,z)$
ξ_5	$E(x,y)$	$\rightarrow$	$E(y,x)$
ξ_6	$E(x,y), E(y,z)$	$\rightarrow$	$E(x,z)$

together with 7 equality constraints $\alpha_1, \ldots, \alpha_7$ of the kind (c) and 7 consistency constraints $\beta_1, \ldots, \beta_7$ corresponding to positions R^1, R^2, S^1, S^2, T^1, T^2, and T^3.

It is easy to verify that Σ is a weakly-acyclic set of TGDs and EGDs. Yet, $\bar{\Sigma}$ is not weakly acyclic. For example, there is a cycle of length 4 through the positions R^1, T^3, and E^2 where the edges are given by constraints ξ_1, ξ_4, and β_1 and the first edge is existential.

If the instance A contains only the tuples $R(1,2)$ and $S(1,3)$, then the chase of A with $\bar{\Sigma}$ does not terminate for the chase order that applies ξ_1, ξ_2, ξ_3, ξ_4, β_2, and β_4 repeatedly in the pattern shown in Figure 4.4. Each line in the table indicates a constraint that fired on some tuple and the new tuple that was introduced as a result. We consider variables in alphabetic order. For example the third line below indicates that ξ_3 fired under the assignment $(u,v,x,y,z) := (2,a,1,b,3)$ introducing the tuple $(2,b)$ into the relation E. This chase continues forever.

New tuple	constraint	fired on
$T(1,2,a)$	ξ_1	$(1,2)$
$T(1,b,3)$	ξ_2	$(1,3)$
$E(2,b)$	ξ_3	$(2,a,1,b,3)$
$E(3,a)$	ξ_4	$(b,3,1,2,a)$
$R(1,b)$	β_2	$(1,2,b)$
$S(1,a)$	β_4	$(1,3,a)$
$T(1,b,c)$	ξ_1	$(1,b)$
$T(1,d,a)$	ξ_2	$(1,a)$
$E(2,d)$	ξ_3	$(2,c,1,d,3)$
$E(3,c)$	ξ_4	$(d,3,1,2,c)$
$R(1,d)$	β_2	$(1,2,d)$
$S(1,c)$	β_4	$(1,3,c)$
$\ldots$		

Figure 4.4 Non-terminating chase

Fix some weakly-acyclic set Σ of TGDs and EGDs. Consider the dependency graph associated with the TGDs in Σ. If R is a relation, we say that a tuple $\bar{a}$ is *good* for R if the depth of every value in it is smaller than or equal to the depth of the

corresponding position in R. That is, $\text{depth}(a_i) \leq \text{depth}(R^i)$. If $\phi(\bar{x})$ is a conjunction of atoms with variables $\bar{x}$, we say that a tuple $\bar{a}$ of the same arity as $\bar{x}$ is good for ϕ if the depth of every value a_i in it is smaller than or equal to the the depth of every position in ϕ where x_i appears. When R or ϕ are clear from context, we simply say that $\bar{a}$ is good. When we consider $\bar{\Sigma}$, we still use the dependency graph associated with the TGDs in Σ and ignore the E relation. Notice that if a TGD fires on a tuple that is good for its premise, then all tuples introduced by its conclusion are good.

Definition 4.1. We say that a chase order is *nice* if whenever several constraints apply, a constraint of the kind which appears earliest in the following list is fired:

1. an equality constraint,
2. a consistency constraint,
3. a constraint firing on a tuple which is good for its premise,
4. a constraint firing on a tuple which is bad for its premise.

Example 4.5. If we chase A from Example 4.4 with the constraints Σ from that example in a nice order, we get the terminating chase shown in Figure 4.5. After the steps shown, all constraints in $\bar{\Sigma}$ are satisfied.

New tuple	constraint	fired on
$E(1,1)$	α_1	$(1,2)$
$E(2,2)$	α_2	$(1,2)$
$E(3,3)$	α_4	$(1,3)$
$T(1,2,a)$	ξ_1	$(1,2)$
$E(a,a)$	α_7	$(1,2,a)$
$T(1,b,3)$	ξ_2	$(1,3)$
$E(b,b)$	α_6	$(1,b,3)$
$E(2,b)$	ξ_3	$(2,a,1,b,3)$
$E(b,2)$	ξ_5	$(2,b)$
$E(3,a)$	ξ_4	$(b,3,1,2,a)$
$E(a,3)$	ξ_5	$(3,a)$
$R(1,b)$	β_2	$(1,2,b)$
$S(1,a)$	β_4	$(1,3,a)$
$T(1,2,3)$	β_7	$(1,2,a,3)$
$T(1,b,a)$	β_7	$(1,b,3,a)$

Figure 4.5 Terminating Chase

It turns out that the only constraints which apply to a bad tuple are consistency constraints (Lemma 4.12), so we never fire constraints of the kind 4 in the definition of

nice order. This implies that bad tuples are only introduced by consistency constraints and Theorem 4.14 below, similar to Theorem 2.1, follows.

If A is a model of $\bar{\Sigma}$, we write $x \equiv y$ if $E(x, y)$ holds in A. We extend the equivalence relation E on elements of the universe to tuples as follows: if $\bar{a}$ and $\bar{b}$ are two r-tuples, then $\bar{a} \equiv \bar{b}$ iff $a_i \equiv b_i$ for $1 \leq i \leq r$.

Lemma 4.11. *If ϕ is a conjunction of relational atoms, A satisfies all equality and consistency constraints, $A \models \bar{a} \equiv \bar{b}$ and $A \models \phi(\bar{a})$, then also $A \models \phi(\bar{b})$.*

Proof. Assume the hypotheses and furthermore that the variables (all free) of ϕ are $\bar{x}$. Then for every atom of the form $R(x_{i_1}, \ldots, x_{i_k})$ in ϕ, we have $A \models R(a_{i_1}, \ldots, a_{i_k})$. To show that $A \models \phi(\bar{b})$, it is enough to show that also $A \models R(b_{i_1}, \ldots, b_{i_k})$. Since A satisfies the equality and consistency constraints, $a_{i_1} \equiv b_{i_1}, \ldots, a_{i_k} \equiv b_{i_k}$, and $A \models R(a_{i_1}, \ldots, a_{i_k})$, we also know that $A \models R(b_{i_1}, a_{i_2}, \ldots, a_{i_k})$, $A \models R(b_{i_1}, b_{i_2}, a_{i_3}, \ldots, a_{i_k})$, ... $A \models R(b_{i_1}, \ldots, b_{i_k})$ by the consistency constraints, as desired. □

We write $[a]$ for the equivalence class $\{b : b \equiv a\}$ of a. We write $\pi_i R$ for the ith projection $\{c_i : R(\bar{c})\}$ of relation R.

Lemma 4.12. *If Σ is a weakly-acyclic set of TGDs and EGDs, then at every step $A_s^{\bar{\Sigma}}$ of the chase of A with $\bar{\Sigma}$ using a nice order such that $A_s^{\bar{\Sigma}}$ satisfies the equality and consistency constraints, the following holds:*

1. *For every relation R, if $a \in \pi_i R$, then $[a] \subseteq \pi_i R$.*
2. *For every relation R, if $a \in \pi_i R$, then there exists $b \equiv a$ such that* $\text{depth}(b) \leq \text{depth}(R^i)$.
3. *If $\phi(\bar{a})$ holds where ϕ is a conjunction of relational atoms, then $\phi(\bar{b})$ holds for a tuple $\bar{b}$ good for ϕ such that $\bar{a} \equiv \bar{b}$.*
4. *If ξ fires on $\bar{a}$, then $\bar{a}$ is good for the premise of ξ.*

Proof. (1) follows directly from the fact that the consistency constraints are satisfied.

We show 2, 3, and 4 by induction on s. Clearly they hold for $s = 0$ since then all values are constants and have depth 0. Now assume that 2, 3, and 4 hold for $A_r^{\bar{\Sigma}}$ for all $r < s$.

(2) If all values in $A_s^{\bar{\Sigma}}$ are constants, then 2 holds trivially. Otherwise there must be a largest value $r < s$ such that a constraint ξ which is not an equality or consistency constraint fired in $A_r^{\bar{\Sigma}}$. Assume ξ fired on tuple $\bar{a}$. Since 4 holds and $A_r^{\bar{\Sigma}}$ satisfies all equality and consistency constraints because the chase order is nice, $\bar{a}$ is

good. Therefore, every new tuple introduced by the conclusion of ξ into a relation R is good for that relation R. This implies that 2 holds for $A_{r+1}^{\bar{\Sigma}}$ The equality and consistency constraints which fire after ξ only add values equivalent to those already in R, so 2 also holds for $A_t^{\bar{\Sigma}}$ for every every t such that $r < t \leq s$.

(3) For every i, pick b_i to be a value of minimal depth in $[a_i]$. Since 2 holds, $\bar{b}$ is good for ϕ. Assume the variables (all free) of ϕ are $\bar{x}$. Then for every atom of the form $R(x_{i_1}, \ldots, x_{i_k})$ in ϕ we have $A_s^{\bar{\Sigma}} \models R(a_{i_1}, \ldots, a_{i_k})$. Since 1 holds, we also have $A_s^{\bar{\Sigma}} \models R(b_{i_1}, \ldots, b_{i_k})$. It follows that $A_s^{\bar{\Sigma}} \models \phi(\bar{b})$.

(4) Assume ξ is of the form $\phi(\bar{x}) \rightarrow \exists \bar{y} \psi(\bar{x}, \bar{y})$. We must have $A_s^{\bar{\Sigma}} \models \phi(\bar{a})$. If $\bar{a}$ is not good, we get a contradiction as follows. By 3 we must have $A_s^{\bar{\Sigma}} \models \phi(\bar{b})$ for a tuple $\bar{b}$ good for ϕ such that $\bar{a} \equiv \bar{b}$. Since we are chasing with a nice order, we must have $A_s^{\bar{\Sigma}} \models \xi(\bar{b})$ and this implies that there must be $\bar{c}$ such that $A_s^{\bar{\Sigma}} \models \psi(\bar{b}, \bar{c})$. By Lemma 4.11, we have $A_s^{\bar{\Sigma}} \models \psi(\bar{a}, \bar{c})$. That is, $A_s^{\bar{\Sigma}} \models \xi(\bar{a})$ so ξ can not fire on $\bar{a}$. □

Lemma 4.13. *If Σ is a weakly-acyclic set of TGDs and EGDs of depth d, then for any s all values in $A_s^{\bar{\Sigma}}$ obtained by using a nice chase order have depth $\leq d$.*

Proof. This follows from Lemma 4.12 part 4, since constraints which fire on tuples which are good for their premise introduce tuples which are good for the relations they are introduced into. □

Theorem 4.14. *For every weakly-acyclic set Σ of TGDs and EGDs, any instance A, and any nice chase order,*

1. *$A^{\bar{\Sigma}}$ is defined, and*
2. *$A^{\bar{\Sigma}}$ can be computed in $O(|\mathrm{dom}(A)|^b)$ steps and in time $O(|\mathrm{dom}(A)|^c)$ where b and c depend only on Σ.*

Proof. Immediate from Lemma 4.13. □

Theorem 4.15. *For every weakly-acyclic set Σ of TGDs and EGDs, any instance T, and any retract U' of $U = T^{\Sigma}$, we can check whether U' has a proper retraction (i.e., whether U' is not a core) and find it in time $O(|\mathrm{dom}(T^{\Sigma})|^b)$ where b depends only on Σ and the block size of T.*

Proof. Similar to that of Theorem 4.6, except using Theorem 4.14 instead of Theorem 2.1. □

Theorem 4.16. *For every weakly-acyclic set Σ of TGDs and EGDs and any instance T, the core of T^{Σ} for a nice chase order can be computed in time $O(|\mathrm{dom}(T)|^b)$ where b depends only on Σ and the block size of T.*

Proof. To compute such a core, we replace Σ with $\bar{\Sigma}$ and proceed as in the proof of Theorem 4.7, except we use Theorem 4.15 instead of Theorem 4.6. By Lemma 4.10, the core so computed using $\bar{\Sigma}$ is the desired core. □

Proof. (**Theorem 4.9**) Similar to that of Theorem 4.3, using Theorem 4.16 instead of Theorem 4.7. □

Procedure FindCore

Input: Source ground instance S of a data exchange problem

Output: Core of a universal solution for S

1. Compute $\bar{\Sigma}_t$ from Σ_t.
2. Chase $(S, \emptyset)$ with Σ_{st} to obtain $(S, T) = (S, \emptyset)^{\Sigma_{st}}$.
3. Chase T with $\bar{\Sigma}_t$ (using a nice order) to obtain $U := T^{\bar{\Sigma}_t}$.
4. For every x, y where $x \in \mathrm{var}(U)$, $y \in \mathrm{dom}(U)$, and $x \neq y$ do
5. Compute T_{xy} as in Lemma 4.4.
6. Look for a homomorphism $h : T_{xy} \to U$ such that $h(x) = h(y)$.
7. If there is such h, then
8. Extend h to an endomorphism h' on U. (Theorem 4.5)
9. From h', compute a proper retraction r on U. (Theorem 4.2)
10. Set $U := U'$.
11. Repeat from step 4.
12. Return U.

4.5 The Bounded Ancestor Property

The crucial property of weakly-acyclic TGDS which we use in Lemma 4.4 can be formalized as follows.

Definition 4.2. A set Σ of TGDs and EGDs has the *bounded ancestor property* for chase orders $\mathcal{O}$ if there exists a such that for

1. any instance T,
2. any n, and
3. and any value $x \in \mathrm{dom}(T_n^{\Sigma})$, under a chase order in $\mathcal{O}$

x has at most a ancestors.

Similarly, Σ has *bounded depth* for chase orders $\mathcal{O}$ if there exists d such that for

1. any instance T,
2. any n, and
3. and any value $x \in \mathrm{dom}(T_n^\Sigma)$, under a chase order in $\mathcal{O}$

x has depth at most d.

Σ has the bounded ancestor property iff it has the bounded depth property since if there is a global bound a on the number of ancestors for any variables, then clearly a is also a global bound on the depth of any variables. Conversely, if Σ has width w and a global bound d on the depth of all variables then dw^d is a global bound on the number of ancestors of any variable.

Theorem 4.17. *If Σ is a set of TGDs and EGDs with the bounded ancestor property (or has bounded depth) for chase orders $\mathcal{O}$, then T^Σ is defined for all chase orders in $\mathcal{O}$.*

Proof. Assume Σ has width w, height e, and the global bound on depth is d. Assume also we chase with a chase order in $\mathcal{O}$. There are at most $e|\Sigma|$ existentially-quantified variables in conclusions of constraints in Σ. Therefore the number n_{i+1} of values of depth at most $i+1$ in T_s^Σ for any s is at most $n_i + e|\Sigma| n_i^w$ since at most one value is introduced for every existentially quantified variable in a constraint in Σ, of which there are at most $e|\Sigma|$ and every tuple on which a constraint fires, of which there are at most n_i^w. Set $f := (1 + e|\Sigma|)$. Then, $n_{i+1} \leq f n_i^w$ which implies that $|\mathrm{dom}(T_s^\Sigma)| = n_d \leq f^{w^d} n_0^{w^d}$ where $n_0 := |\mathrm{dom}(T)|$. Since the number of values in T_s^Σ is bounded independently of s, the chase must terminate and therefore T^Σ is defined. □

Example 4.6. Assume Σ consists of the following constraints

$$D(x) \rightarrow \exists y F(x, y) \tag{4.1}$$

$$D(x) \rightarrow \exists z G(x, z) \tag{4.2}$$

$$F(x, y), G(x, z) \rightarrow R(y, z) \tag{4.3}$$

$$R(w, w) \rightarrow \exists v S(w, v) \tag{4.4}$$

$$S(w, v) \rightarrow D(v) \tag{4.5}$$

It is easy to check that Σ has the bounded ancestor property, but is not weakly acyclic.

The following is a restatement of results proved earlier in terms of the bounded ancestor property:

1. If Σ is a weakly-acyclic set of TGDs, then Σ has the bounded ancestor property for all chase orders (shown in the proof of Theorem 2.1)

2. If Σ is a weakly-acyclic set of TGDs and EGDs, then $\bar{\Sigma}$ has the bounded ancestor property for nice chase orders (Lemma 4.13).

Furthermore, the concept of depth of a position which we use in the section on TGDs can be replaced with a global bound d on depth whose existence follows from the bounded ancestor property. It is straightforward to verify that the results of Section 4.3 go through under the assumption that Σ has the bounded ancestor property. In those cases where we can find some statically-determined chase order so that $\bar{\Sigma}$ has the bounded ancestor property, we can also handle EGDs as in Section 4.4. Therefore, it would be nice if we could show:

> If Σ_1 is a set of TGDs with the bounded ancestor property, Σ_2 is a set of EGDs, and $\Sigma = \Sigma_1 \cup \Sigma_2$, then $\bar{\Sigma}$ has the bounded ancestor property for some set of chase orders $\mathcal{O}$.

Unfortunately, this is false as the following example shows.

Example 4.7. Assume Σ consists of the following constraints

$$D(x) \rightarrow \exists y F(x, y) \tag{4.1}$$

$$D(x) \rightarrow \exists z G(x, z) \tag{4.2}$$

$$F(x, y), G(x, z) \rightarrow R(y, z) \tag{4.3}$$

$$R(w, w) \rightarrow \exists v S(w, v) \tag{4.4}$$

$$S(w, v) \rightarrow D(v) \tag{4.5}$$

$$F(x, y), G(x, z) \rightarrow y = z \tag{4.6}$$

where Σ_1 consists of constraints 1-5 and Σ_2 consists of constraint 6. Σ_1, which consists of the constraints in Example 4.6, has the bounded ancestor property, but is not weakly acyclic. However, chasing an instance in which D which has a single constant and all other relations are empty with Σ leads to a non-terminating chase. Therefore, Σ does not have the bounded ancestor property for any chase order. The same holds for $\bar{\Sigma}$.

This chapter is based on "Data Exchange: Computing Cores in Polynomial Time" by Georg Gottlob and Alan Nash [17] (journal version [18]) I was responsible for developing the techniques to handle equality-generating dependencies and for the generalization beyond weakly-acyclic dependencies. I also contributed towards the presentation and the development of some of the minor technical points in the section on handling tuple-generating dependencies.

5

The Schema Mapping Composition Problem

Composition of mappings between schemas is essential to support schema evolution, data exchange, data integration, and other data management tasks. In many applications, mappings are given by embedded dependencies. In this chapter, we study the issues involved in composing such mappings.

Our algorithms and results extend those of [13] who studied composition of mappings given by several kinds of constraints. In particular, they proved that full source-to-target tuple-generating dependencies (tgds) are closed under composition, but embedded source-to-target tgds are not. They introduced a class of second-order constraints, *SO tgds*, that is closed under composition and has desirable properties for data exchange.

We study constraints that need not be source-to-target and we concentrate on obtaining (first-order) embedded dependencies. As part of this study, we also consider full dependencies and second-order constraints that arise from Skolemizing embedded dependencies. For each of the three classes of mappings that we study, we provide (a) an algorithm that attempts to compute the composition and (b) sufficient conditions on the input mappings that guarantee that the algorithm will succeed.

In addition, we give several negative results. In particular, we show that full dependencies and second-order dependencies that are not limited to be source-to-target are not closed under composition (for the latter, under the additional restriction that no new function symbols are introduced). Furthermore, we show that determining whether the composition can be given by these kinds of dependencies is undecidable.

Composition and Inverse Given two mappings m_{12} and m_{23}, the composition $m_{12} \circ m_{23}$ is the mapping

$$\{\langle A, C\rangle : \exists B(\langle A, B\rangle \in m_{12} \wedge \langle B, C\rangle \in m_{23})\}.$$

We are concerned here with the following problem: given two expressions of the form specified above, find an expression for the composition. That is, we are concerned with the syntactic counterpart to the semantic operation defined above. We say that two $\mathcal{L}$-mappings given by the expressions $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_3, \sigma_4, \Sigma_{12})$ are *compatible* if $\sigma_2 = \sigma_3$ and $\sigma_1, \sigma_2, \sigma_4$ are pairwise disjoint. We only consider composition of compatible $\mathcal{L}$-mappings and therefore we have only a partial composition operation on expressions. We say that $\mathcal{L}$ is *closed under composition* if the composition of any two compatible $\mathcal{L}$-mappings is an $\mathcal{L}$-mapping.

Given a mapping m_{12} between σ_1 and σ_2, we define the inverse m_{12}^{-1} of m_{12} to satisfy

$$(A, B) \in m_{12} \text{ iff } (B, A) \in m_{12}^{-1}.$$

Closure under inverse is defined similarly to closure under composition.

Notice that under these definitions, it is possible to get a language that is closed under both inverse and composition by considering the set of SkTGD-mappings consisting of mappings of the following three kinds: mappings given by the empty set of constraints (unconstrained mappings), mappings given by a set of source-to-target constraints (ST mappings), and mappings given by a set of target-to-source constraints (TS mappings). The inverse of an ST mapping is a TS mapping and conversely. The inverse of an unconstrained mapping is an unconstrained mapping. Furthermore, the composition of ST with ST gives ST, of TS with TS gives TS, and that of any other combination gives an unconstrained mapping. (These facts are not obvious, but follow from our results in subsequent sections.)

5.1 Deductions

In some of the following results and algorithms, we will need to refer to some specific deductive system. Here we outline its basics; the details are not essential.

We write FTGD or SkTGD constraints augmented with constants as *rules* of the form $\phi(\bar{x}), \chi(\bar{x}) \rightarrow \psi(\bar{x})$ leaving the second-order quantifiers over functions $\exists \bar{f}$ and the first-order universal quantifiers $\forall \bar{x}$ implicit. We call $\phi(\bar{x}), \chi(\bar{x})$ the *premise* and $\psi(\bar{x})$

the *conclusion*. (Similarly for FuD or SkED constraints.) If the premise is empty, we write only the conclusion. We call rules of the form $\psi(\bar{c})$, where $\bar{c}$ is a tuple of constants, *facts*. In most cases we will assume, without loss of generality, that our rules have a single atom in the conclusion since every rule with k atoms in the conclusion can always be rewritten as k rules each with a single atom in the conclusion.

Definition 5.1. A *deduction* from rules Σ is a sequence of *rules*, each obtained in one of three ways:

1. by copying a rule from Σ,
2. by applying expand/rename on a rule appearing earlier in the sequence
3. by applying resolution on two rules appearing earlier in the sequence.

We call such rules *axiom* rules, *expand/rename* rules, and *resolution* rules respectively. We say that a deduction *has length* n if it consists of n lines.

A rule r obtained by expand/rename from rule r' may have additional atoms in the premise, may have variables replaced (consistently) by arbitrary terms, may have equalities of the form $v = t$ between a variable v and a term t removed whenever v does not appear elsewhere in the rule, and may have replacements in the conclusion consistent with equalities in the premise.

A rule r obtained by resolution of two rules s and t has as premise the atoms in the premises of s and t, except for those atoms in the conclusion of s and as conclusion the atoms in the conclusion of t.

A rule ξ is a *variant* of ξ' if ξ can be deduced from ξ' without using resolution and conversely. Since each rule has a single atom in the conclusion, a rule r obtained by resolution from rules p, q consists of the conclusion of q and the premises in p and q that do not appear in the conclusion of p.

To illustrate the deductions introduced in Definition 5.1, consider the following examples: The rule $R(x, y), z = f(x, y) \rightarrow S(x, z)$ is a valid result of applying expand/rename to $R(u, v) \rightarrow S(u, f(u, v))$. The rule $R(x, y), S(y, z) \rightarrow S(x, z)$ is the result of applying resolution to rules $R(x, y) \rightarrow S(x, y)$ and $S(x, y), S(y, z) \rightarrow S(x, z)$.

We call a resolution step a σ_2-*resolution* if it involves the elimination of an atom with a relation symbol from σ_2. In the example above, if σ_2 contains S, then we have a σ_2-resolution.

We annotate our deductions by numbering the rules in them in ascending order and by adding annotations to each line indicating how that line was obtained. It is

enough to annotate a resolution rule with just two numbers and an expand/rename rule with a single number and a variable assignment. Axiom rules are indicated through a lack of any other annotation. A variable assignment is a list of items of the form $x := y$ where x is a variable and y is a term.

Example 5.1. Given

- $\Delta = \{R(1,1)\}$ and
- $\Sigma = \{R(x,y) \to S(x,y),\ S(z,z) \to T(z,z)\}$,

the following is a valid deduction from $\Sigma \cup \Delta$:

1.	$R(1,1)$	
2.	$R(x,y) \to S(x,y)$	
3.	$R(1,1) \to S(1,1)$	[2] $x := 1, y := 1$
4.	$S(1,1)$	[1,3]
5.	$S(z,z) \to T(z,z)$	
6.	$S(1,1) \to T(1,1)$	[5] $z := 1$
7.	$T(1,1)$	[4,6]

Here rules 1, 2, and 5 are axioms, 3 and 6 are expand/rename, and 4 and 7 are resolution. □

We call a sequence of at most two rename-only steps followed by a resolution step on the results of these steps a *rename-resolution*. In the example above, 4 is obtained by rename-resolution from 1 and 2 and 7 is obtained by rename-resolution from 4 and 5. A σ_2-rename-resolution is a rename-resolution where the resolution step is a σ_2-resolution.

If there is a deduction γ from a set of constraints Σ where the last line of γ contains a constraint ξ, we say that ξ *is deduced from* Σ, which we write $\Sigma \vdash \xi$, and that γ *witnesses* $\Sigma \vdash \xi$. We write $\Sigma \vdash \Sigma'$ in case $\Sigma \vdash \xi$ for every $\xi \in \Sigma'$. The *$\mathcal{L}$-deductive closure* of Σ is

$$\mathrm{DC}(\mathcal{L}, \Sigma) := \{\xi \in \mathcal{L} : \Sigma \vdash \xi\}.$$

We write $\mathrm{DC}(\Sigma)$ when $\mathcal{L}$ is clear from the context.

We write $D \models \Sigma$ if all constraints in Σ are true in D. We write $\Sigma \models \Sigma'$ if, for all instances D, $D \models \Sigma$ implies $D \models \Sigma'$. It is easy to check that if $\Sigma \vdash \Sigma'$ then also $\Sigma \models \Sigma'$; i.e., the deductive system is *sound*.

We will need the following proposition.

Proposition 5.1. *If Δ is a set of facts in $\mathcal{L}$ and $\Sigma \cup \{\phi\} \subseteq \mathcal{L}$ for $\mathcal{L} \in \{FuD, SkED\}$, then the following are equivalent:*

1. $\Sigma \cup \Delta \vdash \phi$.
2. *There is* $\xi \in \mathcal{L}$ *such that* $\Sigma \vdash \xi$ *and* $\Delta, \xi \vdash \phi$ *with* ξ *over the signature of* Δ.

Proof. If (2) holds then we have deductions γ' and γ'' witnessing $\Sigma \vdash \xi$ and $\Delta, \xi \vdash \phi$. Then γ obtained by appending γ'' to γ' witnesses $\Sigma \cup \Delta \vdash \phi$.

Now assume that (1) holds and γ witnesses $\Sigma \cup \Delta \vdash \phi$. We set γ' to γ except for the following replacements, which we make rule by rule from the first rule in γ to the last:

1. If rule r is an axiom rule from Δ, remove it.
2. If rule r is an axiom rule not from Δ, keep it as it is.
3. If rule r is obtained by expand/rename from rule i, replace every constant c in r and in the variable assignment for r with a corresponding variable v_c (a different variable for every constant).
4. If rule r is obtained by resolution from rules i and j and rule i was an axiom rule from Δ, replace r with a trivial expand/rename from rule j.
5. If rule r is obtained by resolution from rules i and j and rule i was not an axiom rule from Δ, replace r with the result of applying resolution to the new rules i and j.

(See example 5.1 below.) Take ξ to be the last rule in γ'. Steps 4 and 5 ensure that ξ is over the signature of Δ. Then γ' is a valid deduction witnessing $\Sigma \vdash \xi$ since axioms from Δ are no longer used and since replacements 1 through 5 above ensure that rule r is correctly deduced from the previously replaced rules. Also $\Delta, \xi \vdash R(\bar{c})$ is witnessed by a deduction γ'' which consists of the following sequence

- the axioms from Δ that were removed from γ to obtain γ',
- ξ followed by ξ' obtained from ξ by renaming each variable v_c to the corresponding constant c, and
- a sequence of resolution steps using each axiom from Δ and starting with ξ'.

□

Example 5.2. Given the deduction γ from Example 5.1, γ' is:

1.
2. $R(x,y) \to S(x,y)$
3. $R(v_1,v_1) \to S(v_1,v_1)$ [2] $x := v_1, y := v_1$
4. $R(v_1,v_1) \to S(v_1,v_1)$ [3]
5. $S(z,z) \to T(z,z)$
6. $S(v_1,v_1) \to T(v_1,v_1)$ [5] $z := v_1$
7. $R(v_1,v_1) \to T(v_1,v_1)$ [4,6]

The following replacements have been made for each rule:

1. Axiom from Δ: removed by 1.
2. Axiom from Σ; unchanged by 2.
3. Expand rename: 1 replaced with v_1 by 3.
4. Resolution on 1,3: trivial expand/rename by 4.
5. Axiom from Σ; unchanged by 2.
6. Expand rename: 1 replaced with v_1 by 3.
7. Resolution on 4,6: resolution by 5. □

Chase. Here we define a modified chase procedure which is needed in the proof of several results below.

Definition 5.2. Given an instance D, the result of *chasing* D with constraints $\Sigma \subseteq$ SkED and the set of Skolem functions F, which we denote chase(D, Σ, F), is the database D'' obtained from

$$D' := \{R_i(\bar{c}) : \Sigma \cup \Delta \cup \Phi \vdash R_i(\bar{c})\}$$

where

- $\bar{c}$ is a tuple of constants,
- Δ is the set of facts given by D:

$$\Delta := \{R_i(\bar{c}) : D \models R_i(\bar{c})\}$$

- and Φ is the set of facts given by F:

$$\Phi := \{f(\bar{c}) = a : f \in F, f(\bar{c}) = a\}$$

as follows. Define

$$c_0 \equiv c_1 \text{ iff } \Sigma \cup \Delta \cup \Phi \vdash c_0 = c_1.$$

Now to obtain D'' from D', pick one constant c_0 from every equivalence class and replace every constant in that equivalence class with c_0. That is, $D'' := D'/\equiv$. All functions

in F are required to have the same domain which includes D. If they have finite range, then chase(D, Σ, F) is finite.

This definition is a variation on the usual definition, in which the functions in F are constructed during the chase process. In particular, for any chase sequence of the regular chase in which new witnesses are introduced, we set the functions in F accordingly and then, the result of the modified chase defined above is exactly the same the regular chase. On the other hand, we allow the function in F to "collapse witnesses" so the universality property of the regular chase does not always hold for the chase defined here. We use this definition of the chase because (1) we need to chase with SkED-constraints, for which the functions in F are given and (2) because it will be convenient to have a definition of the chase that is closely related to the deductive system we use.

The important property we need of this modified chase is (a similar property holds for the standard chase):

Proposition 5.2. chase$(D, \Sigma, F) \models \Sigma$.

Proof. Set $D'' :=$ chase(D, Σ, F). Let D', Δ, and Φ be as in Definition 5.2. Pick $\xi \in \Sigma$. Assume ξ is $\forall \bar{x}(\phi(\bar{x}), \chi(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y})$ where ϕ, χ, ψ are as in the definition of SkED. Pick any tuple of constants $\bar{c}$ and assume that $D'' \models \phi(\bar{c})$. Pick the unique tuple of constants $\bar{d}$ such that $\Phi \vdash \chi(\bar{c}, \bar{d})$. Then $\Sigma \cup \Delta \cup \Phi \vdash \phi(\bar{c}), \chi(\bar{c}, \bar{d})$. Therefore, since $\xi \in \Sigma$, we also have $\Sigma \cup \Delta \cup \Phi \vdash \psi(\bar{c}, \bar{d})$. Without loss of generality assume that $\psi(\bar{c}, \bar{d})$ is a single relational atom or an equation. In the former case, this relational atom must be in D' and therefore also in D''. In the latter case, this equation equates two constants which are equivalent in D' and therefore equal in D''. Either way, ξ holds in D'' for $\bar{c}$. □

5.2 Full Dependencies

We start by studying composition of FuD-mappings; that is, mappings given by full dependencies. All results in this section apply to both FuD and FTGD-mappings. We will see that the techniques introduced to handle these cases can be extended to handle SkED and ED-mappings. We first show that FTGD is not closed under composition (Theorem 5.3) and, furthermore, that determining whether the composition of two FuD-mappings is a FuD-mapping is undecidable (Theorem 5.4). Then we give necessary and sufficient, non-computable conditions for the composition of two FTGD-mappings to be

a FTGD-mapping (Theorem 5.5). The following algorithm computes the composition of two FuD-mappings given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$. The algorithm, when it terminates, computes the deductive closure of $\Sigma_{12} \cup \Sigma_{23}$, then restricts this deductive closure to those constraints which do not refer to σ_2. Theorem 5.5 below shows that the deductive closure so restricted gives precisely the composition.

Procedure FuDcompose(Σ_{12}, Σ_{23})

Set $\Sigma := \Sigma_{12} \cup \Sigma_{23}$
Repeat
  Set $\Sigma' := \emptyset$
  For every pair $\phi, \psi \in \Sigma$
    For every way in which ϕ, ψ can be σ_2-rename-resolved
    to yield ξ and if there is no variant of ξ in Σ
      set $\Sigma' := \Sigma' \cup \{\xi\}$
  Set $\Sigma := \Sigma \cup \Sigma'$
Until $\Sigma' = \emptyset$
Return $\Sigma_{13} := \Sigma|_{\sigma_{13}}$

FuDcompose terminates on FuD-mappings satisfying the conditions of Theorem 5.9, which can be checked in exponential time (see also Corollary 5.8). The obstacle to composition is recursion, yet recursion is not always a problem (Example 5.3). We also define good-FuD, a subset of FuD recognizable in exponential time, which is closed under composition (Theorem 5.10).

Theorem 5.3. *There are FTGD-mappings whose composition is not an* FO*-mapping. In particular, FTGD is not closed under composition.*

Proof. Consider the FTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

Σ_{12} is	$R(xy)$	$\rightarrow$	$S(xy)$
	$S(xy), S(yz)$	$\rightarrow$	$S(xz)$
Σ_{23} is	$S(xy)$	$\rightarrow$	$T(xy)$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Together, these constraints say that $R \subseteq S \subseteq T$ and that S is transitively closed. The composition $m_{12} \circ m_{23}$ is the set of all pairs (R, T) such that $\mathrm{tc}(R) \subseteq T$. Intuitively, the FTGD-constraints which express the composition are constraints of the form

$$R(x, v_1), R(v_1, v_2), \ldots, R(v_{i-1}, v_i), R(v_i, y) \rightarrow T(x, y)$$

but no finite set of them expresses $\mathrm{tc}(R) \subseteq T$.

In fact, the composition $m_{12} \circ m_{23}$ is not even expressible in first-order logic (FO), since if we had an FO sentence ϕ such that

$$\langle R, T\rangle \in m_{12} \circ m_{23} \text{ iff } (R, T) \models \phi$$

we could create an FO formula $\psi(x, y)$ obtained by replacing every occurrence of $T(u, v)$ in ϕ with $x \neq u \vee y \neq v$. Then given a domain D with $R \subseteq D^2$ we would have $R \models \psi[a, b]$ iff $(R, D^2 - \langle a, b\rangle) \models \phi$ iff $\mathrm{tc}(R) \subseteq D^2 - \langle a, b\rangle$ iff $\langle a, b\rangle \notin \mathrm{tc}(R)$. Therefore $\forall x \forall y \neg\psi(x, y)$ would say that R is a connected graph, contradicting the fact that this can not be expressed in FO (see, e.g., Example 2.3.8 in [10]). □

Theorem 5.4. *Checking whether the composition of two FTGD-mappings is a FTGD-mapping is undecidable (in fact, coRE-hard). Furthermore, the problem is undecidable even when the second mapping is a single fixed source-to-target TGD.*

Proof. We reduce Post's correspondence problem (PCP)—known to be undecidable (see, e.g., [37])—to the problem of deciding whether $m_{12} \circ m_{23}$ is a FTGD-mapping where m_{12} and m_{23} are FTGD mappings. A PCP problem consists of a finite number of *tiles*, each with two strings of 0s and 1s: a top string and a bottom string. The decision problem is to determine whether a PCP problem has a solution. A *solution* is a string S of 0s and 1s such that there is a sequence of tiles for which the concatenation of the top strings and bottom strings of the tiles are both equal to S. For example, the PCP problem with the two tiles 00/001 and 11/1 (the string before the slash is the top string and the string after the slash is the bottom string) has a solution: 0011 obtained by the sequence consisting of the first tile followed by the second tile.

The reduction is partially inspired by an undecidability proof by Christoph Koch (Theorem 3.1 in [24]). Given a PCP problem, we define m_{23} so that there is a solution to the PCP problem iff $m_{12} \circ m_{23}$ is not a FTGD-mapping. The FTGD-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

- $\sigma_1 = \{A, B, O, I, E\}$,
- $\sigma_2 = \{C, D, Q, J\}$,
- $\sigma_3 = \{T\}$

with C quaternary, all other relations binary, and with Σ_{12} and Σ_{23} as described below. We will use A to mark the beginning and end of a path made of O and I edges, which

will correspond to strings of 0s and 1s. We will use B similarly and we will use E as an undirected graph to detect cycles, which will correspond to reused variables.

Similarly, we will use C (which is quaternary) to mark the beginning and end of two path, each made of Q and J edges, which will correspond to strings of 0s and 1s (we use Q and J because they resemble O and I). We write

$$
\begin{array}{lll}
{}^{x}0011^{y} & \text{for} & A(xy), O(xu), O(uv), I(vw), I(wy), \\
{}_{x'}0011_{y'} & \text{for} & B(x'y'), O(x'u'), O(u'v'), I(v'w'), I(w'y'), \text{ and} \\
{}^{x}_{x'}0011^{y}_{y'} & \text{for} & C(xx'yy'), Q(xu), Q(uv), J(vw), J(wy), Q(x'u'), Q(u'v'), J(v'w'), J(w'y').
\end{array}
$$

(the intermediate variables are not specified in this notation). We will also use D as an auxiliary marker and T as our "target."

We will define constraints Σ_{12} and Σ_{23}, the former independent of the PCP instance and the latter encoding the tiles of a PCP instance such that the composition of m_{12} and m_{23} is given by the following set of full TGDs Σ_{13}:

$$
\begin{aligned}
A(xx), B(x'x') &\rightarrow T(xx') && (5.1)\\
A(xy), B(x'y'), E(zz') &\rightarrow T(xx') && (5.2)\\
{}^{x}S^{y}, {}_{x}S_{y} &\rightarrow T(xx') && (5.3)
\end{aligned}
$$

for every S which is a solution to the PCP problem, together with constraints 4 to 8 below. Any PCP problem with a solution has an infinite number of solutions (the concatenation of a finite number of solutions is a solution) so σ_{13} contains 7 constrains for a PCP problem with no solution and infinitely many constraints for a PCP problem with a solution. It is clear that the set containing constraints 1, 2, and all constraints of the form

$$
{}^{x}S^{y}, {}_{x'}S_{y'} \rightarrow T(xx')
$$

for all solutions S except for a solution $\hat{S}$ does not imply

$$
{}^{x}\hat{S}^{y}, {}_{x'}\hat{S}_{y'} \rightarrow T(xx')
$$

Therefore, the composition of the mappings m_{12} and m_{23} is given by a finite set of full TGDs iff the PCP problem has no solution, as desired.

Σ_{12} contains the following full TGDs:

$$A(xy), B(x'y') \rightarrow E(xx') \tag{5.4}$$
$$O(xy) \rightarrow E(xy) \tag{5.5}$$
$$I(xy) \rightarrow E(xy) \tag{5.6}$$
$$E(yx) \rightarrow E(xy) \tag{5.7}$$
$$E(xy), E(yz) \rightarrow E(xz) \tag{5.8}$$
$$A(xy), B(x'y'), E(zz') \rightarrow C(xx'xx') \tag{5.9}$$
$$A(xy), B(x'y') \rightarrow D(xx') \tag{5.10}$$
$$D(xx'), O(xy), O(x'y') \rightarrow Q(xy), Q(x'y'), D(yy') \tag{5.11}$$
$$D(xx'), I(xy), I(x'y') \rightarrow J(xy), J(x'y'), D(yy') \tag{5.12}$$
$$A(xy), B(x'y'), D(yy') \rightarrow C(xx'yy') \tag{5.13}$$

The first six constraints will be used to detect repeated variables. The last four constraints are sufficient to deduce

$${}^{x}S^{y}, {}_{x'}S_{y'} \rightarrow {}^{x}_{x'}S^{y}_{y'}$$

for an arbitrary string S, applying 10 once, 11 once for every 0 in S, 12 once for every 1 in S, and then 13 once. Intuitively, we copy one 0 or one 1 from the top and bottom strings at a time from σ_1 to σ_2 and move the D-marker correspondingly also one step at a time.

Σ_{23} depends on the PCP instance. Given the tiles 00/001 and 11/1, it contains the following full TGDs:

$$C(xx'xx') \rightarrow T(xx') \tag{5.14}$$
$$C(xx'zz'), Q(yu), Q(uz), Q(y'u'), Q(u'v'), J(v'z') \rightarrow C(xx'yy') \tag{5.15}$$
$$C(xx'zz'), J(yz), J(y'u'), J(u'z') \rightarrow C(xx'yy') \tag{5.16}$$

More generally, each of the constraints except for the first one encodes one tile, with the top string encoded as a path through Q and J edges going from y to z and the bottom string encoded as a path through Q and J edges going from y' to z'. Using these

constraints repeatedly corresponding to the tiles that make up a solution S, we obtain

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \rightarrow C(xx'xx'). $$

Intuitively, applying these constraints corresponds to removing one tile at a time from the top and bottom strings and moving the end points in the C-marker until we obtain $C(xx'xx')$. Finaly we apply constraint 14 to obtain

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \rightarrow T(xx'). $$

This shows that we can deduce the constraints 3 introduced above.

We can deduce constraint 1 using constraints 10, 13, and 14 and we can deduce constraint 2 using constraints 9 and 14.

To complete the proof we must show that all other full TGDs over $\sigma_1 \cup \sigma_3$ that can be deduced from $\Sigma_{12} \cup \Sigma_{23}$ are implied by the constraints in Σ_{13}. Clearly, any such constraint must have an atom of the form $T(xx')$ as its conclusion and therefore must be obtained using constraint 14 from a constraint with conclusion $C(xx'xx')$. Such a conclusion can only be obtained using constraints 9, 13, or the constraints in Σ_{23} encoding tiles. If obtained through constraint 9, then since A, B, and E are over σ_1, such a constraint must be implied by constraint 2. If obtained through constraint 13, then we must have $y = x$ and $y' = x'$ and we must obtain $D(xx')$ which can only be obtained through constraint 10. Any constraint so obtained must be implied by constraint 1. We are left with only the possibility of a deduction using the constraints that correspond to tiles. It is tedious, but straightforward to verify that this leads to constraints which are implied by a constraint of the form

$$ {}^{x}S^{y}, {}_{x'}S_{y'} \rightarrow T(xx') $$

where S is a solution to the PCP problem or to some constraint that has a premise of the form

$$ {}^{x}S'^{y}, {}_{x'}S''_{y'} \rightarrow T(xx') $$

where S' and S'' are not necessarily equal and neither is necessarily a solution, but where at least some variable is shared among them. (To verify this, we essentially reverse the process obtained to deduce constraints of the form 3 above.) In the latter case, however, there must be a cycle in E and therefore, by constraint 8 (transitive closure) a 1-loop in

E. Therefore, any such constraint must be implied by constraint 2.

Finally notice that we can move the constraints encoding tiles (which are over σ_2) to Σ_{12}, in which case Σ_{23} contains a single fixed source-to-target constraint. □

Theorem 5.5. *If the FuD-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ with $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$ and $\sigma_{13} = \sigma_1 \cup \sigma_3$, then the following are equivalent*

1. *There is a finite set of constraints $\Sigma_{13} \subseteq$ FuD over the signature σ_{13} such that $m := m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_{13})$.*
2. *There is a finite set of constraints $\Sigma_{13} \subseteq$ FuD over the signature σ_{13} such that*

$$\mathrm{DC}(\mathit{FuD}, \Sigma_{123})|_{\sigma_{13}} = \mathrm{DC}(\mathit{FuD}, \Sigma_{13}).$$

3. *There is k such that for every ξ over σ_{13} satisfying $\Sigma_{123} \vdash \xi$ there is a deduction of ξ from Σ_{123} using at most k σ_2-resolutions.*

Proof. The proof uses Lemmas 5.6 and 5.7 below. First we show the equivalence of (1) and (2) then we show the equivalence of (2) and (3).

Assume (2) holds. Then $\langle A, C\rangle \in m_{12} \cdot m_{23}$

iff	$\exists B\,(A, B, C) \models \Sigma_{123}$	(by definition of $\cdot$)
iff	$(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$	(by Lemma 5.6)
iff	$(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{13})$	(since (2) holds)
iff	$(A, C) \models \Sigma_{13}$.	(since DC is sound)

This shows that (1) holds.

Conversely, assume (1) holds. Then

$(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$

iff	$\exists B\,(A, B, C) \models \Sigma_{123}$	(by Lemma 5.6)
iff	$\langle A, C\rangle \in m_{12} \cdot m_{23}$	(by definition of $\cdot$)
iff	$(A, C) \models \Sigma_{13}$	(since (1) holds)
iff	$(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{13})$.	(since DC is sound)

This shows that (2) holds.

Now assume (3) holds. Set Σ to the set of all constraints in $\mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$ that can be deduced using at most k σ_2-resolutions and no other resolutions. Clearly, every constraint in Σ can be obtained by expand/rename from a finite subset $\Sigma_{13} \subseteq \Sigma$. We show that (2) holds.

Assume that there is a deduction γ witnessing $\Sigma_{123} \vdash \xi$. Since (3) holds, we can assume that γ has $m' \leq m$ σ_2-resolutions. By Lemma 5.7 there is a deduction γ' witnessing $\Sigma_{123} \vdash \xi$ also with m' σ_2-resolutions and with all of them occurring before any other resolutions.

Since the last line of γ' does not contain any symbols from σ_2 we can assume that γ' does not contain any lines containing symbols from σ_2 after the last σ_2-resolution.

Break γ' into two parts: γ'_1 the initial segment of γ up to and including the last σ_2-resolution and γ'_2 the remainder of γ'. Every constraint ψ in γ'_1 must be in Σ, by definition of Σ and therefore we must have $\Sigma_{13} \vdash \psi$. Since every constraint ψ in γ'_2 does not contain any symbols from σ_2 and since $\Sigma_{123}|_{\sigma_{13}} \subseteq \Sigma_{13}$, we also have $\Sigma_{13} \vdash \psi$. Therefore, $\Sigma_{13} \vdash \xi$ as desired.

Conversely, assume (2) holds. Take k to be the total number of σ_2-resolutions needed to deduce every $\psi \in \Sigma_{13}$ from Σ_{123}. Assume $\Sigma_{123} \vdash \xi$. Then there is a deduction γ witnessing $\Sigma_{13} \vdash \xi$. Clearly, γ has no σ_2-resolutions. From γ, we obtain γ' witnessing $\Sigma_{123} \vdash \xi$ by appending to γ a deduction of every constraint in Σ_{13} and by replacing every line where an axiom from Σ_{13} is used by a vacuous expand/rename of the line where the deduction of that axiom ends. Clearly, γ' has exactly k σ_2-resolutions as desired. This shows that (3) holds. □

Lemma 5.6. *Under the hypotheses of Theorem 5.5, the following are equivalent:*

1. $(A, C) \models \mathrm{DC}(\mathit{FuD}, \Sigma_{123})|_{\sigma_{13}}$.
2. $\exists B\, (A, B, C) \models \Sigma_{123}$.

Proof. Assume $(A, B, C) \models \Sigma_{123}$ for some B. Then $(A, B, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{123})$ (by soundness) and therefore $(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$ since B is not mentioned in $\mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$.

Conversely, assume $(A, C) \models \mathrm{DC}(\mathrm{FuD}, \Sigma_{123})|_{\sigma_{13}}$. We set

$$(A', B, C') := \mathrm{chase}((A, \emptyset, C), \Sigma_{123}).$$

If this chase terminates and $A = A'$ and $C = C'$, then we have $(A, B, C) \models \Sigma_{123}$ by Proposition 5.2 which implies $(A, B) \models \Sigma_{12}$ and $(B, C) \models \Sigma_{23}$, as desired.

It is clear that the chase terminates since no new constants are introduced. Now assume, to get a contradiction, that $A \neq A'$ or $C \neq C'$. Set Δ_{AC} to the set of facts given by A and C. Then we must have

$$\Sigma_{123} \cup \Delta_{AC} \vdash R(\bar{c})$$

where $\bar{c}$ is a tuple of constants and R is a relation in A or C not containing $\bar{c}$ or

$$\Sigma_{123} \cup \Delta_{AC} \vdash c_0 = c_1$$

where c_0, c_1 are distinct constants in A or C.

We consider the former case; the latter is similar. We must have $(A, C) \not\models R(\bar{c})$. If $\Sigma_{123} \cup \Delta_{AC} \vdash R(\bar{c})$ then by Proposition 5.1 there exists $\xi \in \text{FuD}$ over σ_{13} such that $\Sigma_{123} \vdash \xi$ and $\Delta_{AC}, \xi \vdash R(\bar{c})$. Since $(A, C) \models \Delta_{AC}$ and $(A, C) \not\models R(\bar{c})$, it follows that $(A, C) \not\models \xi$, contradicting $(A, C) \models \text{DC}(\text{FuD}, \Sigma_{123})|_{\sigma_{13}}$. □

Lemma 5.7. *Under the hypotheses of Theorem 5.5, if there is a deduction γ witnessing $\Sigma_{123} \vdash \xi$ with at most k σ_2-resolutions, then there is γ' witnessing $\Sigma_{123} \vdash \xi$ with at most k σ_2-resolutions and where furthermore all σ_2-resolutions occur before all other resolutions.*

Proof. The basic idea of the proof is to repeatedly swap the first σ_2-resolution which occurs after a non-σ_2-resolution with that resolution until all σ_2-resolutions occur first.

Assume we have $k < m$ and a deduction $\gamma_{k,\ell}$ witnessing $\Sigma_{123} \vdash \xi$ with

1. exactly m σ_2-resolutions,
2. where the first k resolutions are σ_2-resolutions, and
3. where there are exactly ℓ non-σ_2-resolutions before the $k+1$-th σ_2-resolution or the end of the deduction.

We proceed by induction on k and ℓ. Given $\gamma_{k,\ell}$ with $\ell > 0$ we show how to obtain the deduction $\gamma_{k,\ell-1}$ satisfying 1, 2, and 3. In the case where $\ell = 0$ we simply set $\gamma_{k+1,\ell'} := \gamma_{k,0}$ picking ℓ' so that $\gamma_{k+1,\ell'}$ satisfies 1, 2, and 3 above. Once we get $\gamma_{m,\ell'}$ for some ℓ' we set $\gamma' := \gamma_{m,\ell'}$ and we are done.

Consider the line s containing δ, the $(k+1)$th σ_2-resolution in $\gamma_{k,\ell}$ of, say, lines i and j containing, respectively, α and β. Consider also the line r containing λ, the ℓ-th non-σ_2-resolution in $\gamma_{k,\ell}$ of, say, lines r_1 and r_2.

Now we have to consider several cases. If $i, j < r < s$, then we can obtain $\gamma_{k,\ell-1}$ by moving line s to just before r. If lines i, j are not derived from line r, then we can obtain $\gamma_{k,\ell-1}$ by first rearranging the deduction $\gamma_{k,\ell}$ to obtain a deduction $\gamma'_{k,\ell}$ such that $i, j < r < s$, then proceeding as above.

Otherwise, either α or β has been obtained through expand/rename from line r. To simplify the presentation we assume that both have been obtained through a single expand/rename from line r (the other cases are similar). We have $r < i, j < s$. By rearranging $\gamma_{k,\ell}$ if needed, we can assume $i = r + 1$, $j = r + 2$ and $s = r + 3$. Since α and β can be obtained from r by expand/rename, α_1, α_2 and β_1, β_2 (intuitively, the "unresolved" parts of λ expand/renamed as α and β) can be obtained, respectively, from

lines r_1, r_2 by expand/rename so that α is the resolution of α_1, α_2 and β is the resolution of β_1, β_2. We replace the four contiguous lines r, i, j, s:

r	λ	$[r_1, r_2]$
$r+1$	α	$[r]$
$r+2$	β	$[r]$
$r+3$	δ	$[r+1, r+2]$

with the following seven lines:

r	α_1	$[r_1]$
$r+1$	α_2	$[r_2]$
$r+2$	β_1	$[r_1]$
$r+3$	β_2	$[r_2]$
$r+4$	λ_1	$[r+1, r+2]$
$r+5$	λ_2	$[r, r+4]$
$r+6$	δ	$[r+5, r+3]$.

The important point is that line $r+4$ now contains a σ_2-resolution since α_2 and β_1 must resolve through a relation symbol of σ_2, because α and β do. Notice that we have δ on line $r+6$ since the result of resolution on $\alpha_1, \alpha_2, \beta_1$ and β_2 is the same as the result of resolution on α and β (this is because resolution is "associative"). □

Corollary 5.8. *Under the hypotheses of Theorem 5.5,* FuDcompose$(\Sigma_{12}, \Sigma_{23})$*, whenever it terminates, yields* Σ_{13} *such that* $m_{12} \circ m_{23}$ *is given by* $(\sigma_1, \sigma_3, \Sigma_{13})$.

Notice that after the k-th iteration of the main loop, Σ will contain a variant of every constraint that can be deduced using at most k σ_2-resolution steps. The constraints in the proof of Theorem 5.3 fail to satisfy (3) of Theorem 5.5 and therefore FuDcompose$(\Sigma_{12}, \Sigma_{23})$ will not terminate when Σ_{12} and Σ_{23} are as in the proof of Theorem 5.3 for input. In contrast, FuDcompose$(\Sigma_{12}, \Sigma_{23})$ will terminate on Σ_{12}, Σ_{23} from the example below, which does satisfy (3) of Theorem 5.5, so recursion is not always bad.

Example 5.3. Consider the FTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

Σ_{12} is	$R(xy)$	$\rightarrow$	$S(xy)$
	$S(xy), S(yz)$	$\rightarrow$	$R(xz)$
Σ_{23} is	$S(xy)$	$\rightarrow$	$T(xy)$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Together, these constraints say that $R \subseteq S \subseteq T$, and that R and S are transitively closed (because the constraints

$$\begin{array}{rcl} S(xy), S(yz) & \rightarrow & S(xz) \\ R(xy), R(yz) & \rightarrow & R(xz) \end{array}$$

can be deduced from Σ_{12}). The constraints

$$\begin{array}{rcl} R(xy), R(yz) & \rightarrow & R(xz) \\ R(xy) & \rightarrow & T(xy) \end{array}$$

express exactly the composition $m_{12} \circ m_{23}$, and are exactly those found by FuDcompose$(\Sigma_{12}, \Sigma_{23})$. □

The coRE-hardness from Theorem 5.4 implies that algorithm FuDcompose may not terminate even when the composition is a FuD-mapping. This happens, for example, in the following case.

Example 5.4. Consider the FTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

Σ_{12} is	$R(xy)$	$\rightarrow$	$S(xy)$
	$R(xy), R(yz)$	$\rightarrow$	$R(xz)$
	$S(xy), S(yz)$	$\rightarrow$	$S(xz)$
Σ_{23} is	$S(xy)$	$\rightarrow$	$T(xy)$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$.
The constraints

$$\begin{array}{rcl} R(xy), R(yz) & \rightarrow & R(xz) \\ R(xy) & \rightarrow & T(xy) \end{array}$$

express exactly the composition $m_{12} \circ m_{23}$, but algorithm FuDcompose will never terminate since it will deduce at least the infinitely many constraints it would deduce in the proof of Theorem 5.3. This is because Σ_{12} here includes all the constraints in Σ_{12} there. □

Even if the algorithm terminates, it may produce a result which is exponential in the size of the input mappings. This is unavoidable, as the following example shows.[1]

Example 5.5. There is a FTGD-mapping m_{12} and a sequence of FTGD-mappings m_{23}^k given by Σ_{12} and Σ_{23}^k over fixed signatures $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$ where R, S, and T are binary relations such that the composition $m_{12} \circ m_{23}^k$ grows exponentially in the size of Σ_{23}^k.

The mapping m_{12} is given by:

$R(xy), R(yx)$	$\rightarrow$	$S(xy)$
$R(xy), R(xx)$	$\rightarrow$	$S(xy)$

and the family of mappings m_{23}^k is given by Σ_{23}^k which contains the single constraint

[1] This is essentially a result on query unfolding [36]. Lucian Popa first brought this to our attention through an example that required a varying schema.

$$\frac{}{S(xu_1), S(u_1u_2), ..., S(u_{k-1}y) \quad \rightarrow \quad T(xy)}$$

saying that if there is a path of length k in S then there is an edge in T. For each atom $S(uv)$ in the premise of this last constraint, we can substitute either $R(uv), R(vu)$ or $R(uv), R(uu)$ to obtain a constraint over $\sigma_1 \cup \sigma_3$ which gives 2^k constraints in the composition. □

The following conditions are sufficient for algorithm FuDcompose to terminate. On the other hand, Example 5.6 below illustrates a case where these conditions are violated. Intuitively, these conditions say that there is no "non-trivial" recursion on some atom in σ_2. It would be nice to have simpler termination conditions of wide applicability, but we are not aware of any such. Items 3 and 4 guarantee this "non-triviality." If either one fails, then the recursion can only proceed for a finite number of steps.

Theorem 5.9. *Under the hypotheses of Theorem 5.5, if no constraint of the form* $\phi(\bar{z}), S(\bar{y}) \rightarrow S(\bar{x})$ *can be deduced from* Σ_{123} *using only* σ_2*-rename-resolutions, such that*

1. $\phi(\bar{z})$ *is a conjunction of atoms over* σ_{123}*,*
2. *there is no atom* $S(\bar{w})$ *in* $\phi(\bar{z})$ *where* $\bar{w}$ *contains all the variables in* $\bar{x}$*,*
3. *there is a variable in* $\bar{x}$ *which is not in* $\bar{y}$*, and*
4. S *is a relation symbol in* σ_2

then FuDcompose$(\Sigma_{12}, \Sigma_{23})$ *terminates and therefore* $m_{12} \circ m_{23}$ *is a FuD-mapping. Furthermore, these conditions can be verified in exponential time in the size of* $\Sigma_{12} \cup \Sigma_{23}$*.*

Proof. The conditions can be checked in exponential time as follows. Run $k_{\sigma_2} - 1$ iterations of the main loop of FuDcompose$(\Sigma_{12}, \Sigma_{23})$ where k_{σ_2} is the number of relation symbols in σ_2 and check whether a constraint of the form given below appears in Σ.

For the termination claim, assume the hypotheses hold. Consider any deduction γ witnessing $\Sigma_{123} \vdash \xi$ which uses only σ_2-rename-resolutions where ξ contains a σ_2 atom in the conclusion. We may assume without loss of generality that all rename operations are performed first, followed by all resolution operations. Assume also that every rule in γ contains a single atom in its conclusion and that every rule is used in at most one resolution step.

Such a deduction can be represented as a tree T where every node is one atom. Every non-leaf node in T is the conclusion of some rule r. The children of r are the atoms in the premise of rule r. The premise of ξ consists of all the leaves of T and its conclusion is the root of T.

It is easy to check that any subtree T' of T which contains, for every node, either all its children in T or none of them, can be converted into a deduction γ' witnessing $\Sigma_{123} \vdash \xi'$ where the premise of ξ' consists of all the leaves of T' and its conclusion is the root of T'.

Since the hypothesis holds, no such subtree may contain $S(\bar{x})$ as its root and $S(\bar{y})$ as a leaf where S is a relation symbol in σ_2 and $\bar{x} \not\subseteq \bar{y}$. Therefore, any path from the leaves to the root in T containing a node $S(\bar{y})$ may only contain at most $2^r r!$ other nodes with S atoms where r is the arity of S. This means that any such path must have length bounded by $k_{\sigma_2} 2^{r_{\sigma_2}} r_{\sigma_2}!$ where k_{σ_2} is the number of relation symbols in σ_2 and r_{σ_2} is the maximum arity of a relation symbols in σ_2. As a result, up to variants there is only a finite number of conclusions of Σ_{123} obtainable through σ_2-rename-resolutions and this implies that FuDcompose$(\Sigma_{12}, \Sigma_{23})$ terminates. □

Definition 5.3. A FuD-mapping is a *good-FuD-mapping* if it is given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and no constraint of the form $\phi(\bar{z}), S(\bar{y}) \to S'(\bar{x})$ where

1. $\phi(\bar{z})$ is a conjunction of atoms over $\sigma_1 \cup \sigma_2$,
2. there is no atom $S(\bar{w})$ in $\phi(\bar{z})$ where $\bar{w}$ contains all the variables in $\bar{x}$,
3. there is a variable in $\bar{x}$ which is not in $\bar{y}$, and
4. S and S' are both relation symbols in σ_1 or both in σ_2

can be deduced from Σ_{12} using only σ_1-rename-resolutions or only σ_2-rename-resolutions. We define good-FTGD similarly.

We can check whether an FuD-mapping is a good-FuD-mapping in exponential time in the size of the constraints as in the proof of Theorem 5.9.

Theorem 5.10. *good-FuD and good-FTGD are closed under composition and inverse.*

Proof. Assume that two good-FuD-mappings are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$. Set $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$. Assume constraints ξ of the form

$$\phi(\bar{z}), S(\bar{y}) \to S'(\bar{x})$$

and ξ' of the form

$$\psi(\bar{v}), S'(\bar{x}) \to S''(\bar{u})$$

each fails at least one of the conditions 2-4 (with the appropriate subsitutions of signatures and sets of constraints for ξ') of Definition 5.3 with S' in σ_2. Then resolution on ξ

and ξ' gives ξ'':

$$\phi(\bar{z}), \psi(\bar{v}), S(\bar{y}) \rightarrow S''(\bar{u})$$

which also fails at least one of the conditions 2-4 as follows. If ξ' fails 4, then S'' is not in σ_2 and therefore ξ'' fails 4. If ξ' fails 2, then ξ'' fails 2 as well. Therefore, assume ξ' fails 3. That is, every variable in $\bar{u}$ is in $\bar{x}$. Therefore if ξ fails 2, then ξ'' also fails 2. Also, if ξ fails 3, then every variable in $\bar{x}$ is in $\bar{y}$ so ξ'' fails 3. Finally, if ξ fails 4, then S is not in σ_2 so ξ'' also fails 4.

Then it is easy to verify (by induction on the length of proofs) that no constraint ξ'' satisfying conditions 2, 3, and 4 where both S and S' are in σ_2 of Definition 5.3 can be deduced using only σ_2-rename-resolutions from the constraints specifying the two mappings. Therefore, the hypotheses of Theorem 5.9 hold so the composition exists and furthermore it satisfies the conditions of an good-FuD-mapping.

A similar proof works for good-FTGD-mappings. □

We examined many other subsets of FuD for closure under composition and inverse, but were unable to find more natural conditions of similarly wide applicability. Since source-to-target FuD-constraints are total and surjective, it is natural to wonder whether the set of all total and surjective FuD-mappings is closed under composition. The following example shows it is not.

Example 5.6. Consider the FTGD-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

Σ_{12} is	$R(xy)$	$\rightarrow$	$S(xy)$
	$R(xy), S(yz)$	$\rightarrow$	$S(xz)$
Σ_{23} is	$S(xy)$	$\rightarrow$	$T(xy)$

and where $\sigma_1 = \{R\}$, $\sigma_2 = \{S\}$, and $\sigma_3 = \{T\}$. Here m_{12} and m_{23} are total and surjective and their composition says that $\mathrm{tc}(R) \subseteq T$, which we have seen in the proof of Theorem 5.3 is not expressible even in FO. □

5.3 Second-Order Dependencies

In order to handle existential quantifiers in a ED-mapping, we will first convert the ED constraints which specify the mapping into SkED constraints (by Skolemizing) and this will give us SkED-mappings. Therefore, in this section we focus on the composition of SkED-mappings; in the next section we consider how to convert SkED-mappings back to ED-mappings. There are two cases of composition to consider. *Unrestricted* composition,

in which we are allowed to introduce additional existentially-quantified functions in order to express the composition and *restricted* composition in which we are only allowed to use function symbols from the input mappings. In this section we concentrate on restricted composition. SkED constraints require special semantics, which we examine in Section 5.6. All results in this section apply to both SkED and SkTGD-mappings (the former correspond to SO tgds which are not restricted to being source-to-target).

Theorems 5.3 and 5.4 from the previous section show that SkTGD is not closed under restricted composition and that determining whether the restricted composition of two SkTGD-mappings is a SkTGD-mapping is undecidable. This is because in restricted composition we are not allowed to add function symbols, so SkTGD does not add any power toward the restricted composition of FTGD mappings. As in the case of FuD-mappings, we give necessary and sufficient, undecidable conditions for two SkED-mappings to have restricted composition (Theorem 5.11), and we give sufficient conditions for restricted composition that can be checked efficiently.

Theorem 5.11 suggests essentially the same algorithm for composition of SkED-mappings as FuDcompose; we call it SkEDcompose. The only difference between them is that SkEDcompose operates on SkED constraints while FuDcompose operates on FuD constraints. Correctness of SkEDcompose, sufficient conditions for its termination, and good-SkED-mappings are defined for SkED just like for FuD.

Theorem 5.11. *If the SkED-mappings m_{12}, m_{23} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ with $\Sigma_{123} := \Sigma_{12} \cup \Sigma_{23}$ and $\sigma_{13} = \sigma_1 \cup \sigma_3$, then the following are equivalent*

1. *There is a finite set of constraints $\Sigma_{13} \subseteq SkED$ over the signature σ_{13} such that $m := m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_{13})$ where Σ_{13} has no function symbols or constants other than those appearing in Σ_{123}.*

2. *There is a finite set of constraints $\Sigma_{13} \subseteq SkED$ over the signature σ_{13} such that*

$$\mathrm{DC}(SkED, \Sigma_{123})|_{\sigma_{13}} = \mathrm{DC}(SkED, \Sigma_{13})$$

   where Σ_{13} has no function symbols or constants other than those appearing in Σ_{123}.

3. *There is k such that for every ξ over σ_{13} satisfying $\Sigma_{123} \vdash \xi$ there is a deduction of ξ from Σ_{123} using at most k σ_2-resolutions.*

Proof. Essentially the same as that of Theorem 5.5, using Lemma 5.12 below instead of Lemma 5.6. □

Lemma 5.12. *Under the hypotheses of Theorem 5.11, the following are equivalent:*

1. $(A, C) \models \mathrm{DC}(\mathit{SkED}, \Sigma_{123})|_{\sigma_{13}}$.
2. $\exists B\, (A, B, C) \models \Sigma_{123}$.

Proof. Assume $(A, B, C) \models \Sigma_{123}$ for some B. Then $(A, B, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})$ (by soundness) and therefore $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$ since B is not mentioned in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$.

Conversely, assume $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. In particular, this implies the existence of all Skolem functions mentioned in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. There may be additional Skolem functions mentioned in Σ_{123}, but not in $\mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. We give arbitrary values to these additional Skolem functions. We define F to be the set containing all these Skolem functions and Φ to the set of all facts in F.

We set $(A', B, C') := \mathrm{chase}((A, \emptyset, C), \Sigma_{123}, F)$.

If the chase terminates and $A = A'$ and $B = B'$, then we have $(A, B, C) \models \Sigma_{123}$ by Proposition 5.2, which implies $(A, B) \models \Sigma_{12}$ and $(B, C) \models \Sigma_{23}$, as desired.

The chase terminates because of the required safety condition (which is preserved by the deduction rules). Therefore, (A', B, C') is well-defined. Now assume, to get a contradiction, that $A \neq A'$ or $C \neq C'$. Set Δ_{AC} to the set of facts given by A and C. Then we must have

$$\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash R(\bar{c})$$

where $\bar{c}$ is a tuple of constants and R is a relation in A or C not containing $\bar{c}$ or

$$\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash c_0 = c_1$$

where c_0, c_1 are distinct constants in A or C.

We consider the former case; the latter is similar. We must have $(A, C) \not\models R(\bar{c})$. If $\Sigma_{123} \cup \Delta_{AC} \cup \Phi \vdash R(\bar{c})$ then by Proposition 5.1 there exists $\xi \in \mathrm{SkED}$ over σ_{13} such that $\Sigma_{123} \vdash \xi$ and $\Delta_{AC} \cup \Phi \cup \{\xi\} \vdash R(\bar{c})$. Since $(A, C) \models \Delta_{AC}$ and $(A, C) \not\models R(\bar{c})$, it follows that $(A, C) \not\models \xi$, contradicting $(A, C) \models \mathrm{DC}(\mathrm{SkED}, \Sigma_{123})|_{\sigma_{13}}$. □

5.4 Embedded Dependencies

Now we consider composition of ED-mappings; that is, mappings given by embedded dependencies. To compute the composition of two ED-mappings m_{12}, m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ we will proceed in three steps, as follows.

Procedure EDcompose(Σ_{12}, Σ_{23})

1. Σ'_{12} := Skolemize(Σ_{12})
   Σ'_{23} := Skolemize(Σ_{23})
2. Σ'_{13} := SkEDcompose($\Sigma'_{12}, \Sigma'_{23}$)
3. Return DeSkolemize(Σ'_{13})

The first step, Skolemize, is straightforward and the second step, SkEDcompose, has been discussed in the previous section, so here we concentrate on the third step, de-Skolemization. We provide a sound (but not complete) algorithm for de-Skolemization: DeSkolemize. Even if the second step succeeds, it may be impossible to find $\Sigma_{13} \subseteq$ ED such that $\Sigma'_{13} \equiv \Sigma_{13}$ (Example 5.8) so we identify necessary and sufficient, polynomial-time checkable conditions for our algorithm to succeed (Proposition 5.14). DeSkolemize may produce a result of size exponential in the size of its input; we show that in the general case this is unavoidable (Theorem 5.15), but we also provide polynomial-time checkable conditions for DeSkolemize to run in polynomial time in the size of its input (Proposition 5.14). The algorithm consists of 12 steps and is as follows; we provide a detailed description of each step with examples at the end of this section.

Procedure DeSkolemize(Σ)

1. Unnest
2. Check for cycles
3. Check for repeated function symbols
4. Align variables
5. Eliminate restricting atoms
6. Eliminate restricted constraints
7. Check for remaining restricted constraints
8. Check for dependencies
9. Combine dependencies
10. Remove redundant constraints
11. Replace functions with $\exists$-variables
12. Eliminate unnecessary $\exists$-variables

We prove the following correctness and efficiency results for DeSkolemize at the end of this section.

Theorem 5.13. *If* DeSkolemize(Σ) *succeeds on input* $\Sigma \subseteq$ *SkED giving* Σ'*, then*

$$\Sigma' \subseteq \textit{ED and } \Sigma' \equiv \Sigma.$$

Proposition 5.14.

1. DeSkolemize(Σ) *succeeds on input* $\Sigma \subseteq$ *SkED iff it reaches Step 9, which can be checked in polynomial time in the size of* Σ.

2. *Furthermore, for every constant* ℓ, DeSkolemize *runs in polynomial time on any set of constraints* Σ *such that by the end of Step 8 there are no more than* ℓ *constraints containing any one function symbol* f. *This can be checked in polynomial time in the size of* Σ. □

Intuitively DeSkolemize attempts to put the constraints in its input Σ into a form where they are the obvious result of Skolemization, then it reverses this Skolemization in the obvious way. Most of the work is done in bringing the constraints to such a form.

Step 11 is where function symbols are actually replaced by existentially-quantified variables. For it to work properly, constraints must be combined as is done in Step 9. These two steps are, in some sense, the main steps in the algorithm. Steps 10 and 12 are just 'clean up' steps. The remaining steps, 1 through 8 ensure that the constraints are in the proper form for Steps 9 and 11. Procedure DeSkolemize may abort at Step 2, 3, 4, 7, or 8. Except for Step 4, these steps only check that the constraints are in the desired form and abort if they are not. Before going through the steps of DeSkolemize in detail, let us look at a simple example to give some intuition for the algorithm.

Example 5.7. Assume that the input to DeSkolemize consists of the single constraint

$$R(x,y) \rightarrow S(x,f(x,y)), S(f(x,y),y)$$

which is obtained from Skolemizing the TGD-constraint

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

which says that for every edge in R there is a path of length 2 in S. Strictly speaking this is not an SkTGD constraint as defined in the preliminaries (since the function symbol f appears in the conclusion), but is close enough. Step 1 gives the following set of SkTGD constraints:

$R(x,y), u = f(x,y)$	$\rightarrow$	$S(x,u)$
$R(x,y), u = f(x,y)$	$\rightarrow$	$S(u,y)$

We call x and y base variables and u a term variable. Steps 2 and 3 succeed and, in this case, Step 4 does nothing. However if the constraints had been

$$\begin{array}{lcl} R(x,y), u = f(x,y) & \rightarrow & S(x,u) \\ R(w,z), v = f(w,z) & \rightarrow & S(w,z) \end{array}$$

then Step 4 would have replaced variables w, z, v with x, y, u in the second constraint. The goal of this step is to have every equation in which a given Skolem function appears be identical. In this case, there are no equalities that restrict the values of the Skolem functions, so Steps 5 and 6 do nothing and Step 7 succeeds. Step 8 checks that every base variable that appears in a conclusion also appears as an argument to every Skolem function in the premise (we will discuss the case of nested Skolem functions later). In this case, both x and y satisfy this condition. If we were to apply Step 11 at this point, we would obtain the constraints

$$\begin{array}{lcl} R(x,y) & \rightarrow & \exists u\, S(x,u) \\ R(x,y) & \rightarrow & \exists u\, S(u,y) \end{array}$$

which say that for every edge in R from x to y there is an outgoing S-edge from x and an incoming S edge into y. This is not quite the same as saying that there is a path of length 2 from x to y, because these edges may not meet. That is, we may have values of u witnessing that the first constraint holds which are different from those values of u which witness that the second constraint holds. This is why we first apply Step 9 (Step 4 is intended to make Step 9 possible), which gives the following constrains:

$$\begin{array}{lcl} R(x,y), u = f(x,y) & \rightarrow & S(x,u) \\ R(x,y), u = f(x,y) & \rightarrow & S(u,y) \\ R(x,y), u = f(x,y) & \rightarrow & S(x,u), S(u,y) \end{array}$$

Now Step 10 simplifies this to the single constraint:

$$\begin{array}{lcl} R(x,y), u = f(x,y) & \rightarrow & S(x,u), S(u,y) \end{array}$$

since the other two easily follow from it and Step 11 yields

$$\begin{array}{lcl} R(x,y) & \rightarrow & \exists u\, S(x,u), S(u,y) \end{array}$$

as desired. Notice that at every step we have constrains that are equivalent to those at the previous step.

Of course, this is a simple example; we will see soon that things can get more complicated. We abort at some steps of the algorithm if the constraints can not be put into the desired form for subsequent steps. By doing so, we may fail to de-Skolemize some constraints which in fact are equivalent to embedded dependencies. However, it seems

likely that deciding whether some SkED constraints are equivalent to ED constraints is undecidable, so we do not hope for a complete algorithm. The following example from [13] shows that de-Skolemization is not always possible.

Example 5.8. Consider the ED-mappings m_{12} and m_{23} given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_2, \sigma_3, \Sigma_{23})$ where

Σ_{12} is	$E(x,y)$	$\rightarrow$	$F(x,y)$
	$E(x,y)$	$\rightarrow$	$\exists u\, C(x,u)$
	$E(x,y)$	$\rightarrow$	$\exists v\, C(y,v)$
Σ_{23} is	$F(x,y), C(x,u), C(y,v)$	$\rightarrow$	$D(u,v)$

and where $\sigma_1 = \{E\}$, $\sigma_2 = \{F, C\}$, and $\sigma_3 = \{D\}$. Here, Steps 1 and 2 of EDcompose succeed, but Step 3 fails no matter what algorithm is used for it, since $m_{12} \circ m_{23}$ is not a ED-mapping as shown in [13]. □

Since DeSkolemize(Σ) may produce a result of size exponential in the size of Σ due to Step 9, EDcompose(Σ_{12}, Σ_{23}) may produce a result of size exponential in the size of $\Sigma_{12} \cup \Sigma_{23}$ due to de-Skolemization, even when the preceding composition steps yield a polynomial-size result. The following theorem shows that in the general case this is unavoidable.

Theorem 5.15. *There are two sequences of TGD-mappings m_{12}^k and m_{23}^k given by Σ_{12}^k and Σ_{23}^k such that the TGD-composition $m_{12}^k \circ m_{23}^k$ grows exponentially in the size of $\Sigma_{12}^k \cup \Sigma_{23}^k$, but the SkTGD-composition $m_{12}^k \circ m_{23}^k$ grows linearly in the size of $\Sigma_{12}^k \cup \Sigma_{23}^k$.*

This algorithm in fact can be applied to any set of SkED-constraints, but since we are interested in those SkED-constraints obtained from ED-mappings by Skolemize and SkEDcompose, we add some observations that apply to this special case. In the special case we are interested in, procedure DeSkolemize may abort only at Step 3, 7, or 8. In particular, the following result follows from these observations.

Theorem 5.16. EDcompose *generalizes view unfolding. That is, if*

1. *$\sigma_2 = \{V_1, \ldots, V_k\}$ and each V_i is a view given by the conjunctive query with equations $\exists \bar{y}_i \phi_i(\bar{x}_i, \bar{y}_i)$ over σ_1 where $\phi_i(\bar{x}_i, \bar{y}_i)$ is a conjunction of atoms (including equations) over $\bar{x}_i, \bar{y}_i$,*
2. *$\sigma_3 = \{W_1, \ldots, W_k\}$ and each W_i is a view given by the conjunctive query with equations $\exists \bar{v}_i \psi_i(\bar{u}_i, \bar{v}_i)$ over σ_2 where $\psi_i(\bar{u}_i, \bar{v}_i)$ is a conjunction of atoms (including equations) over $\bar{u}_i, \bar{y}_i$,*

3. Σ_{12} *is a functional mapping from* σ_1 *to* σ_2 *given by*

$$\begin{array}{lrcl} (\alpha_i) & \phi_i(\bar{x}_i, \bar{y}_i) & \rightarrow & V_i(\bar{x}_i) \\ (\beta_i) & V_i(\bar{x}_i) & \rightarrow & \exists \bar{y}_i \phi_i(\bar{x}_i, \bar{y}_i), \end{array}$$

and

4. Σ_{23} *is a functional mapping from* σ_2 *to* σ_3 *given by*

$$\begin{array}{lrcl} (\gamma_i) & \psi_i(\bar{u}_i, \bar{v}_i) & \rightarrow & W_i(\bar{u}_i) \\ (\delta_i) & W_i(\bar{u}_i) & \rightarrow & \exists \bar{v}_i \psi_i(\bar{u}_i, \bar{v}_i), \end{array}$$

then EDcompose *correctly computes their composition*

$$\begin{array}{rcl} \psi_i^{\bar{\phi}}(\bar{u}_i, \bar{v}_i) & \rightarrow & W_i(\bar{u}_i) \\ W_i(\bar{u}_i) & \rightarrow & \exists \bar{v}_i \psi_i^{\bar{\phi}}(\bar{u}_i, \bar{v}_i), \end{array}$$

where $\psi_i^{\bar{\phi}}$ *denotes the result of substituting in* ψ_i *the conjunctions* $\phi_1, \ldots, \phi_k$ *for the ocurrences of* $V_1, \ldots, V_k$.

Proof. (Sketch) Since SkEDcompose does resolution only through σ_2, the following resolution patterns are possible:

1. α_i with β_i,
2. one or more constraints from $\{\alpha_1, \ldots, \alpha_k\}$ with γ_i,
3. δ_i with β_j, and
4. δ_i with γ_i.

In particular, SkEDcompose terminates. It is easy (but tedious) to verify that (a) the constraints obtained by resolution of δ_i with γ_i can be deduced from the others and that (b) the remaining constraints can be deskolemized by DeSkolemize. □

The algorithm DeSkolemize, depends on $\vdash^*$, which is used in Steps 5 and 10. $\vdash^*$ is *some* sound polynomial-time approximation of $\models$. That is, if $\Sigma \vdash^* \phi$, then $\Sigma \models \phi$. Of course, the converse may not hold, but we require that if $\phi \in \Sigma$, then $\Sigma \vdash^* \phi$. Its use in Step 10 is non-essential; there we simply take advantage of the fact that $\vdash^*$ is available. In Step 5 we do use it essentially; however, even if $\Sigma \vdash^* \phi$ is only true when $\phi \in \Sigma$, DeSkolemize will succeed on a large class of inputs.

There are known cases in which there exists such $\vdash^*$ which is also complete, but has complexity **NP**. For example, when Σ has *stratified witnesses* (see [9] and [11]), then $\vdash^*$ is complete and can be computed in **NP**, by first 'chasing' the premise of ϕ, then looking for a homomorphism of the conclusion of ϕ into the result of this chase. (The chase needed in this case is a straightforward adaptation to the case where existential quantification is replaced by Skolem functions; the functions are treated as uninterpreted symbols.) The fact that Σ has *stratified witnesses* ensures that the chase terminates and that the result of chasing ϕ is polynomial in the size of ϕ. The **NP** complexity comes from looking for a homomorphism. A more detailed discussion for the options in the implementation of $\vdash^*$ would take us too far away from our main concerns here. For the purposes of the algorithm below, $\vdash^*$ can be treated as a black box. DeSkolemize works for any $\vdash^*$ which is sound; the more complete $\vdash^*$ is, the larger the set of inputs on which DeSkolemize succeeds.

We proceed to discuss every step in more detail. For each step, we provide a brief explanation and, where appropriate, an example.

1. **Unnest:**

   The goal of this step is to bring the constraints into a normal form which will make it easier for subsequent steps to operate on them.

   Set $\Lambda_1 := \{\psi : \phi \in \Sigma\}$ where ψ is equivalent to and obtained from ϕ by "unnesting" terms and eliminating non-variable terms from relational atoms and from the conclusion so that in ψ:

   (a) Function symbols occur only in equalities in the premise.
   (b) Every term $f(\bar{x})$ occurs in only one atom.
   (c) Every equation is of the form $y = z$ or of the form $u = f(\bar{x})$, where $u, \bar{x}, y$, and z are variables and f is a function symbol. We call the later a *defining* equation for u. Furthermore, u (which we call a *term* variable for f) does not appear in any relational atom or on the left-hand side of any other defining equation. We call variables which are not term variables *base* variables.
   (d) The conclusion contains at most one atom.

2. **Check for cycles:**

   The goal of this step is to abort the computation for constraints which contain cyclic dependencies among Skolem terms. Such constraints can not be de-Skolemized.

For every $\phi \in \Lambda_1$, construct the graph G_ϕ where the edges are variables in ϕ and where there is an edge (v, u) iff there is an equation of the form $v = f(\ldots u \ldots)$. We say that variable v depends on u, if there is a path in G_ϕ from v to u. If G_ϕ has a cycle, abort. Otherwise, set $\Lambda_2 := \Lambda_1$.

In the general case, a term variable may depend on other term variables or even on itself. This step is intended to rule out the latter case. For example, this happens in the following constraint:

$$R(x, y), u = f(x, v), v = g(y, u) \rightarrow S(u, v).$$

In the special case of constraints arising in EDcompose, Lemma 5.17 below allows us to assume that term variables depend only on base variables. That is, there are no equalities of the form $v = f(\ldots u \ldots)$ where u is a term variable. In particular, this guarantees that G_ϕ will have no cycles.

Notice that (5.20) can be obtained by resolution from (5.18) and

$$\phi(j\bar{y}), S(j\bar{y}), \bar{v} = \bar{g}(j\bar{y}) \rightarrow \tau(j\bar{y}, \bar{v}). \tag{5.17}$$

(5.17) is obtained from (5.19) by the substitution $\bar{y} \mapsto j\bar{y}$.

3. **Check for repeated function symbols:**

   The goal of this step is to abort the computation if any constraints contain two atoms with the same function symbol. While in some special cases it is possible to exploit some symmetries in order to de-Skolemize such constraints, we take the easy way out and give up. These are not constraints obtained directly from Skolemizing first order constraints since in such constraints every appearance of a Skolem function would have exactly the same arguments.

   For every $\phi \in \Lambda_2$ check that ϕ does not contain two atoms with the same function symbol. If it does, abort. Otherwise, set $\Lambda_3 := \Lambda_2$. This is the step in which DeSkolemize fails on the mappings in Example 5.8. The Skolemized constraints from Example 5.8 are:

Σ_{12} is	$E(x, y)$	$\rightarrow$	$F(x, y)$
	$E(x, y), u = f(x, y)$	$\rightarrow$	$C(x, u)$
	$E(x, y), v = g(x, y)$	$\rightarrow$	$C(y, v)$
Σ_{23} is	$F(x, y), C(x, u), C(y, v)$	$\rightarrow$	$D(u, v)$

SkEDcompose gives the following four constraints:

$E(x,y), E(x,w), E(y,z), u = f(x,w), v = f(y,z)$	$\rightarrow$	$D(u,v)$
$E(x,y), E(x,w), E(z,y), u = f(x,y), v = g(z,y)$	$\rightarrow$	$D(u,v)$
$E(x,y), E(w,x), E(y,z), u = g(w,x), v = f(y,z)$	$\rightarrow$	$D(u,v)$
$E(x,y), E(w,x), E(z,y), u = g(w,x), v = g(z,y)$	$\rightarrow$	$D(u,v)$

4. **Align variables:**

The goal of this step is to get all ocurrences of a Skolem term to be the same. When we Skolemize constraints, this must be the case.

Rename the variables in Λ_3 to obtain Λ_4 satisfying:

(a) For every function symbol f and any two equalities of the form $u = f(\bar{x})$ and $v = f(\bar{y})$ in Λ_4, u is the same variable as v and $\bar{x}$ is the same sequence of variables as $\bar{y}$.

(b) For every two different function symbols f and g and any two equalities of the form $u = f(\bar{x})$ and $v = g(\bar{y})$ in Λ_4, u and v are different variables.

If this is not possible, abort. After this step, there is a unique term variable v_f associated to the function symbol f.

This steps fails, for example, on the following constraints:

$R(x,y), u = f(x,y), v = g(x,y)$	$\rightarrow$	$S(u,v)$
$R(x,y), u = f(x,y), v = g(y,x)$	$\rightarrow$	$T(u,v)$

In the special case of constraints arising in EDcompose this step will always succeed. This follows from Lemma 5.17 below and the following considerations. After the Skolemize steps, every function symbol appears in exactly one constraint if we allow multiple atoms in the conclusion or, equivalently, in constraints with identical premises. Clearly, we can always align variables on such constraints. Also, we can clearly align variables after an expand/rename step, so let us consider a resolution step. Assume we have a set of constraints Σ in which the variables are aligned and that to this set Σ we add the constraint

$$\phi(\bar{x}), \psi(\bar{z}) \rightarrow \rho(\bar{x}, \bar{z})$$

obtained by resolution from $\phi(\bar{x}) \rightarrow S(\bar{y})$ and $S(\bar{y}), \psi(\bar{z}) \rightarrow \rho(\bar{y}, \bar{z})$, both in Σ, where the variables in $\bar{y}$ are also in $\bar{x}$ and where $\phi(\bar{x})$ and $\psi(\bar{z})$ are conjunctions of atoms with variables from $\bar{x}$ and $\bar{z}$ respectively. We need to consider Skolem

functions in $\phi(\bar{x})$ and in $\psi(\bar{z})$. Lemma 5.17 shows that those in $\psi(\bar{z})$ can be replaced with new Skolem functions which depend only on the base variables. On the other hand, those in $\phi(\bar{x})$ have not changed and are already aligned. Therefore, the variables in Σ' consisting of Σ and the new constraint are also aligned. Since variables in the input constraints to SkEDcompose can be aligned (because they have been obtained by Skolemization), it follows that the variables in the output constraints of SkEDcompose can also be aligned.

5. **Eliminate restricting atoms:**

   If u is a term variable for f and u appears in any other atom, we call that atom an *f-restriction*. If ϕ has an f-restriction in the premise, we say that f restricts ϕ. If ϕ has an f-restriction in the conclusion, we say that ϕ restricts f. If there is any function f which restricts ϕ, we say that ϕ is restricted. This step and the next two deal with restrictions.

   Set $\Lambda_5 := \{\phi' : \phi \in \Lambda_4\}$ where ϕ' is ϕ with the maximal set of restrictions removed from the premise which gives $\Lambda_4 \vdash^* \phi'$. It is easy to verify that such a maximal set always exists and is unique.

   Consider, for example, the constraints

ϕ_1	$R(x)$	$\rightarrow$	$\exists y\, S(x,y)$
ϕ_2	$S(x,y)$	$\rightarrow$	$U(x,y)$
ϕ_3	$S(x,x)$	$\rightarrow$	$T(x)$

   where $\sigma_2 = \{S\}$. Skolemization, basic composition, and the first few steps of DeSkolemize give the following constraints Λ_4:

ψ_1	$R(x), y = f(x)$	$\rightarrow$	$U(x,y)$
ψ_2	$R(x), y = f(x), x = y$	$\rightarrow$	$T(x)$

   In this case, $\Lambda_5 = \Lambda_4$. These constraints can be de-Skolemized to yield the following constraints

ψ_1'	$R(x)$	$\rightarrow$	$\exists y\, U(x,y)$
ψ_2'	$\forall y(U(x,y) \rightarrow x = y)$	$\rightarrow$	$(U(x,x) \rightarrow T(x))$

   which, however, are not ED-constraints. On the other hand, if we add the constraints

ϕ_4	$S(x,y)$	$\rightarrow$	$T'(x)$
ϕ_5	$T'(x)$	$\rightarrow$	$T(x)$

then Λ_5 is

ψ_1	$R(x), y = f(x)$	$\rightarrow$	$U(x, y)$
ψ_3	$R(x), y = f(x)$	$\rightarrow$	$T(x)$
ψ_4	$R(x), y = f(x)$	$\rightarrow$	$T'(x)$
ψ_5	$T'(x)$	$\rightarrow$	$T(x)$

Notice that the restriction on ψ_2 has been eliminated to give ψ_3, since ψ_4 and ψ_5 imply ψ_3.

The basic intuition is that we do not know how to de-Skolemize a constraint of the form

$$\phi(\bar{x}), u = f(\bar{x}), v = g(\bar{x}), u = v \rightarrow S(\bar{y})$$

because we do not know how to express the restriction $u = v$ on the Skolem functions once these are replaced by existentially-quantified variables ($\exists$-variables for short). Part of the problem is that the restriction is in the premise, but after the replacement, the $\exists$-variables corresponding to the Skolem functions appear only in the conclusion. A similar problem occurs with a constraint of the form

$$\phi(\bar{x}), u = f(\bar{x}), v = x_i \rightarrow S(\bar{y}).$$

6. **Eliminate restricted constraints:**

   In this step, we eliminate some constraints as follows. We first classify all function symbols as either free or restricted. Free function symbols will be those for which we are free to pick any values. We recursively define the restricted function symbols as follows. f is restricted if there is a constraint ϕ which restricts f and such that all functions which restrict ϕ are restricted. In particular, f is restricted if a constraint ϕ restricts f and no function restricts ϕ. All other functions are free. Now we set Λ_6 to be the set of constraints $\phi \in \Lambda_5$ such that no free function restricts ϕ. The rationale for this step is that if a free function f restricts ϕ, then we can choose the values of f such that a restricting equation for v_f in the premise never holds.

   For example, in the following constraints

ϕ_1	$R(x, y), u = f(x, y), v = g(x, y), u = y$	$\rightarrow$	$T(x, v)$
ϕ_2	$R(x, y), u = f(x, y), v = g(x, y), v = y$	$\rightarrow$	$T(x, u)$

   f restricts ϕ_1 which restricts g and g restricts ϕ_2 which restricts f. Therefore, both f and g are free and we can eliminate both constraints. It is clear that this is sound since we can set f and g such that the range of f and g are disjoint from

each other and from the values appearing in R and T. Then the premises of ϕ_1 and ϕ_2 will never hold and both constraints will always be satisfied regardless of the choice of R and T.

7. **Check for remaining restricted constraints:**
   If there are any restricted constraints in Λ_6, abort. Otherwise, set $\Lambda_7 := \Lambda_6$.

8. **Check for dependencies:**
   For every $\phi \in \Lambda_7$ and every term variable v in ϕ, define $D_{\phi,v}$ to be the set of base variables on which v depends. Set V_ϕ to the set of base variables which appear in the conclusion of ϕ. Now for every term variable v in the conclusion of ϕ, check that $V_\phi \subseteq D_{\phi,v}$. If this fails, abort. Otherwise, set $\Lambda_8 := \Lambda_7$.

   In this step and the next one, we make sure that once we replace Skolem functions with $\exists$-variables (which will happen in Step 11) we get equivalent constraints. One direction of this equivalence is straightforward: we can set the $\exists$-variables to the values of the corresponding Skolem functions. The difficulty is in the other direction. We must make sure that given values for the $\exists$-variables witnessing that the ED-constraints hold, we can set the Skolem functions to also witness that the corresponding SkED-constraints hold.

   To understand why we must check dependencies, consider the constraints

Σ_{12} is	$A(x), y = f(x)$	$\rightarrow$	$F(x,y)$
	$B(u), v = g(u)$	$\rightarrow$	$G(u,v)$
Σ_{23} is	$F(x,y), G(u,v)$	$\rightarrow$	$T(x,y,u,v)$

   which when composed yield the single constraint ϕ:

$$A(x), B(u), y = f(x), v = g(u) \rightarrow T(x,y,u,v).$$

   In this case, $D_{\phi,y} := x$, $D_{\phi,v} := u$, $V_\phi := \{x, u\}$ and the check fails. The problem with the obvious replacement of Skolem functions with $\exists$-variables is that in ϕ'

$$A(x), B(u) \rightarrow \exists y, v\, T(x,y,u,v)$$

   y depends on both x and u, instead of only on x as desired and v also depends on both x and u, instead of only on u as desired. In fact ϕ and ϕ' are not equivalent, as witnessed by the relations $A := \{1, 2\}$, $B := \{3\}$, and $T := \{\langle 1537\rangle, \langle 2538\rangle\}$ for which ϕ' holds, but not ϕ. The problem here is that $x = 1$ forces $g(3) := 7$,

yet $x = 2$ forces $g(3) = 8$ and these choices are incompatible. Interestingly, the following set of two first-order sentences Φ is equivalent to ϕ:

$$A(x) \to \exists y \forall u (B(u) \to T(x, y, u, v))$$

$$B(u) \to \exists v \forall x (A(x) \to T(x, y, u, v)).$$

However, these are not ED-constraints, which is what our algorithm tries to produce.

9. **Combine dependencies:**

Set $\Lambda_9 := \{\psi_\Phi : \emptyset \neq \Phi \subseteq \Lambda_8\}$ where ψ_Φ is defined as follows. If there is a function f which appears in every $\phi \in \Phi$, then the premise of ψ_Φ consists of the atoms in all the premises in Φ and the conclusion of ψ_Φ consists of the atoms in all the conclusions of Φ (remove duplicate atoms). Otherwise, ψ_Φ is some constraint in Φ. Notice that $\Lambda_9 \supseteq \Lambda_8$ since $\psi_{\{\phi\}} = \phi$.

We have already seen the need for this step in Example 5.7. There Λ_8 is

ϕ_1 is	$R(x,y), u = f(x,y)$	$\to$	$S(x,u)$
ϕ_2 is	$R(x,y), u = f(x,y)$	$\to$	$S(u,y)$

and Λ_9 is

ψ_{ϕ_1}	$R(x,y), u = f(x,y)$	$\to$	$S(x,u)$
ψ_{ϕ_2}	$R(x,y), u = f(x,y)$	$\to$	$S(u,y)$
ψ_{ϕ_1,ϕ_2}	$R(x,y), u = f(x,y)$	$\to$	$S(x,u), S(u,y)$

This step is not always as trivial as it looks in Example 5.7. Consider, for example, the constraints from the proof of Theorem 5.15 below where $1 \leq i \leq k$:

Σ_{12}^k is	$R_0(x)$	$\to$	$\exists y\, S_0(x,y)$
	$R_i(x)$	$\to$	$S_i(x)$
Σ_{23}^k is	$S_0(xy), S_i(x)$	$\to$	$T_i(y)$

In this case Λ_8 consists of k constraints of the form

$$R_0(x), y = f(x), R_i(x) \to T_i(y)$$

and Λ_9 consists of $2^k - 1$ constraints of the form

$$R_0(x), y = f(x), R_Z(x) \to T_Z(y)$$

where Z is a non-empty subset of $\{1, \ldots, k\}$ and where $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$ and similarly for T_Z.

10. **Remove redundant constraints:**
    Pick some set $\Lambda_{10} \subseteq \Lambda_9$ such that $\Lambda_{10} \vdash^* \phi$ for every $\phi \in \Lambda_9$, and such that this does not hold for any proper subset of Λ_{10}.

    We have seen above that in Example 5.7 Λ_9 is

ψ_{ϕ_1}	$R(x,y), u = f(x,y)$	$\rightarrow$	$S(x,u)$
ψ_{ϕ_2}	$R(x,y), u = f(x,y)$	$\rightarrow$	$S(u,y)$
ψ_{ϕ_1,ϕ_2}	$R(x,y), u = f(x,y)$	$\rightarrow$	$S(x,u), S(u,y)$

    In this case, $\Lambda_{10} := \{\psi_{\phi_1,\phi_2}\}$ since both ψ_{ϕ_1} and ψ_{ϕ_2} follow from ψ_{ϕ_1,ϕ_2}. This happens because the premises of ψ_{ϕ_1} and ψ_{ϕ_2} are the same, but this is not always the case. In particular, $\Lambda_{10} = \Lambda_9$ in the case of the constraints from the proof of Theorem 5.15 discussed above.

11. **Replace functions with $\exists$-variables:**
    Set $\Lambda_{11} := \{\phi' : \phi \in \Lambda_{10}\}$ where the premise of ϕ' is the premise of ϕ with all equalities removed and where the conclusion of ϕ' is the conclusion of ϕ, with all variables appearing on the left of equalities in ϕ existentially quantified.

    This step is where the elimination of Skolem functions actually takes place, but since most of the preparatory work has already been done, it is very simple. For example, it converts

    $$R(x,y), u = f(x,y) \rightarrow S(x,u), S(u,y)$$

    to

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

12. **Eliminate unnecessary $\exists$-variables:**
    Set $\Lambda_{12} := \{\phi' : \phi \in \Lambda_{11}\}$ and return Λ_{12} where ϕ' is like ϕ, but where existentially quantified variables which do not appear in the conclusion atom have been removed (with their corresponding existential quantifier).

    If we apply Step 11 to the following constraint

    $$R(x,y), u = f(x,y), v = g(x,y) \rightarrow S(x,u), S(u,y)$$

we would obtain

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

Clearly v is not needed, so in this Step we replace the constraint above with

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

Example 5.9. Consider three runs of the algorithm DeSkolemize(Σ^i_{13}), for $i \in \{1, 2, 3\}$. Let $\Sigma^i_{13} = \{\gamma_1, \ldots, \gamma_i\}$ be a set of the following (unnested) SkTGD constraints:

γ_1	$R_1(y), R_2(x), y = f(x)$	$\rightarrow$	$T_1(x)$
γ_2	$R_2(x), y = f(x)$	$\rightarrow$	$T_2(y)$
γ_3	$R_2(x), y = f(x)$	$\rightarrow$	$R_1(y)$

For completeness, we note that each Σ^i_{13} is obtained by first de-Skolemizing ED-mappings given by Σ^i_{12} and Σ^i_{23}, which are shown below, and then invoking SkEDcompose:

i	Σ^i_{12}	Σ^i_{23}	Σ^i_{13}
1.	$\{\alpha_1, \alpha_2\}$	$\{\beta_2\}$	$\{\gamma_1\}$
2.	$\{\alpha_1, \alpha_2\}$	$\{\beta_1, \beta_2\}$	$\{\gamma_1, \gamma_2\}$
3.	$\{\alpha_1, \alpha_2, \alpha_3\}$	$\{\beta_1, \beta_2\}$	$\{\gamma_1, \gamma_2, \gamma_3\}$

Dependencies $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2$ are specified as:

α_1	$R_1(x)$	$\rightarrow$	$S_1(x)$
α_2	$R_2(y)$	$\rightarrow$	$\exists z(S_2(zy))$
α_3	$S_2(zy)$	$\rightarrow$	$R_1(z)$
β_1	$S_2(zy)$	$\rightarrow$	$T_2(z)$
β_2	$S_1(x), S_2(xy)$	$\rightarrow$	$T_1(y)$

In all three runs of DeSkolemize(Σ^i_{13}), Steps 2 and 3 pass since each of γ_1, γ_2, γ_3 is cycle-free and has no multiple atoms with the same function symbol. Step 4 has no effect, since the variable names of the dependencies are already aligned. The remaining steps are explained below.

In the run DeSkolemize($\{\gamma_1\}$), Step 5 has no effect, because $\{\gamma_1\}$ is a singleton set. Its only member γ_1 gets eliminated in Step 6, since there are no rules in $\{\gamma_1\}$ with f-restricting atoms in conclusions. Intuitively, γ_1 is a tautology because we can always construct f whose range is disjoint with R_1. Hence, $\{\gamma_1\}$ is equivalent to the empty set of constraints, which is trivially in TGD.

In the run DeSkolemize($\{\gamma_1, \gamma_2\}$), γ_2 contains an f-restricting atom T_2 in its conclusion. Hence, we cannot eliminate the restricted constraint γ_1 in Step 6, and so de-Skolemization aborts in Step 7.

In the run DeSkolemize($\{\gamma_1, \gamma_2, \gamma_3\}$), we are able to de-Skolemize despite γ_2. In Step 5, $\Delta_0 = \{\gamma_1, \gamma_2, \gamma_3\}$. By considering the only function symbol f, we get $\Delta_1 = \{\psi, \gamma_2, \gamma_3\} \equiv \Delta_0$ where ψ is obtained by eliminating the restricting atom $R_1(y)$ from the premise of γ_1 as

$$\psi \quad := \quad R_2(x), y = f(x) \rightarrow T_1(x)$$

Clearly, $\Delta_0 \vdash^* \psi$, since $\Delta_0 \supset \{\gamma_1, \gamma_3\} \vdash^* \psi$. Δ_1 has no restricting constraints, so Step 6 has no effect and Step 7 passes. Step 8 succeeds with $\Lambda_8 = \Lambda_7 = \{\psi, \gamma_2, \gamma_3\}$, since every dependency in Δ_1 has at most one term variable y in its conclusion. Taking γ_3 as an example, we get $V_{\gamma_3} = \{x, y\}$, $D_{\gamma_3,x} = D_{\gamma_3,y} = \bigcup_{u \in V_{\gamma_3}} D_{\gamma_3,u} = \{x\}$.
In Step 9, combining the dependencies for $\Phi = \Lambda_8$ yields

$$\gamma_4 \quad := \quad R_1(y), R_2(x), y = f(x) \rightarrow T_1(x), T_2(y), R_1(y)$$

(Combinations resulting from proper subsets of Λ_8 are not shown for brevity). In Step 10, we remove the redundant constraints which include ψ, γ_2, γ_3, because they share the premise with γ_4 and their conclusion is subsumed by that of γ_4; we obtain $\Lambda_{10} = \{\gamma_4\}$. Finally, replacing function f by an existential variable in γ_4 yields

$$\Lambda_{12} = \{R_2(x) \rightarrow \exists y(T_1(x) \wedge T_2(y) \wedge R_1(y))\}$$

Thus, DeSkolemize($\{\gamma_1, \gamma_2, \gamma_3\}$) $\subseteq$ TGD. □

The following technical lemma shows that in the case of constraints that arise from Skolemizing ED constraints and running SkEDcompose on them, there is no need to consider nested Skolem functions. The main point is that a constraint of the form 5.20 below can be replaced by a constraint of the form 5.23 below. The technical conditions below reflect the case that arises from applying a resolution step, which is the only step in SkEDcompose which may give rise to nested Skolem functions. Constraint 5.20 below is obtained by resolution of constraints 5.18 and 5.19.

Lemma 5.17. *If $\Sigma \subseteq$ SkCQ$^=$ consists of the following three constraints:*

$$\psi(\bar{x}), \bar{u} = \bar{f}(\bar{x}) \rightarrow S(j\bar{y}), \rho(\bar{x}, \bar{u}) \tag{5.18}$$

$$\phi(\bar{y}), S(\bar{y}), \bar{v} = \bar{g}(\bar{y}) \rightarrow \tau(\bar{y}, \bar{v}) \tag{5.19}$$

$$\psi(\bar{x}), \phi(j\bar{y}), \bar{u} = \bar{f}(\bar{x}), \bar{v} = \bar{g}(j\bar{y}) \to \tau(j\bar{y}, \bar{v}) \tag{5.20}$$

where

- $\bar{x}, \bar{y}, \bar{u}$, *and* $\bar{v}$ *are disjoint tuples of variables,*
- S *is a relational symbol,*
- $\psi(\bar{z}), \phi(\bar{z}), \rho(\bar{z})$, *and* $\tau(\bar{z})$ *are conjunctions of atoms with variables from* $\bar{z}$*,*
- $j : \bar{y} \to \bar{x}\bar{u}$ *is a function mapping variables to variables (so* $j\bar{y}$ *is a tuple of variables from* $\bar{x}\bar{u}$*, possibly with repetitions)*
- $\bar{f}\bar{x}$ *is a tuple of functions* $f_1, \ldots, f_k$ *whose arguments are variables from* $\bar{x}$*, and*
- $\bar{g}\bar{y}$ *is a tuple of functions* $g_1, \ldots, g_\ell$ *disjoint from* $\bar{f}$ *whose arguments are variables from* $\bar{y}$*.*

then Σ *is equivalent to* $\Sigma' \subseteq \mathrm{SkCQ}^=$ *which consists of*

$$\psi(\bar{x}), \bar{u} = \bar{f}'(\bar{x}) \to S(j\bar{y}), \rho(\bar{x}, \bar{u}) \tag{5.21}$$

$$\phi(\bar{y}), S(\bar{y}), \bar{v} = \bar{g}'(\bar{y}) \to \tau(\bar{y}, \bar{v}) \tag{5.22}$$

$$\psi(\bar{x}), \phi(j\bar{y}), \bar{u} = \bar{f}(\bar{x}), \bar{v} = \bar{h}(\bar{x}) \to \tau(j\bar{y}, \bar{v}) \tag{5.23}$$

where $\bar{f}', \bar{g}'$, *and* $\bar{h}$ *are disjoint tuples of functions which do not appear in* Σ*.*

Moreover, (5.20) is equivalent to (5.23).

Proof. If Σ holds, then we have functions $\bar{f}$ and $\bar{g}$ witnessing this. Set $\bar{f}' := \bar{f}$, $\bar{g}' := \bar{g}$ and define functions $\bar{h}$ by $\bar{h}(\bar{x}) := \bar{g}(j\bar{y})$ where $\bar{u} := \bar{f}(\bar{x})$ (so $j\bar{y}$ is uniquely determined by $\bar{x}$). Then (5.18'), (5.19'), and (5.23) hold. That is, $\Sigma \models \Sigma'$. A similar argument shows that (5.20) implies (5.23).

If, on the other hand, Σ holds, then we have functions $\bar{f}', \bar{g}'$ witnessing that (5.18') and (5.19') hold. Then $\bar{f} := \bar{f}'$ and $\bar{g} := \bar{g}'$ witness that (5.18) and (5.19) hold, and (5.20) follows from these.

Now assume that (5.23) holds. We have $\bar{f}$ and $\bar{h}$ witnessing this. We want to show that (5.20) holds. Define G as follows:

$$G(\bar{a}) := \{\bar{h}(\bar{x}) : \psi(\bar{x}), \phi(j\bar{y}), \bar{u} = \bar{f}(\bar{x}), j\bar{y} = \bar{a}\}$$

(remember that $j\bar{y}$ is a tuple of variables in $\bar{x}, \bar{u}$). Set $\bar{g}(\bar{a}) := \bar{b}$ for some tuple $\bar{b} \in G(\bar{a})$ (it does not matter which one) if $G(\bar{a}) \neq \emptyset$. Otherwise, set $\bar{g}(\bar{a}) := \bar{c}$ for some arbitrary tuple $\bar{c}$.

We must show that $\bar{f}$ and $\bar{g}$ as defined witness that (5.19) and (5.20) hold. Assume the premise of (5.20) holds for some values of $\bar{x}$. Set $\bar{u} := f(\bar{x})$ and $\bar{v} := \bar{g}(j\bar{y})$. We need to show that $\tau(j\bar{y}, \bar{v})$ holds.

Since the premise of (5.20) holds, $h(\bar{x}) \in G(j\bar{y})$ and therefore $G(j\bar{y}) \neq \emptyset$. This implies that $\bar{g}(j\bar{y}) \in G(j\bar{y})$. That is, $\bar{g}(j\bar{y}) = \bar{h}(\bar{z})$ for some $\bar{z}$ such that $\psi(\bar{z}), \phi(j\bar{y})$ and $\bar{x}, \bar{f}(\bar{x})$ and $\bar{z}, \bar{f}(\bar{z})$ coincide on the range of j. Since $\bar{g}(j\bar{y}) = \bar{h}(\bar{z})$, the premise of

$$\psi(\bar{z}), \phi(j\bar{y}), \bar{v} = \bar{h}(\bar{z}) \rightarrow \tau(j\bar{y}, \bar{v}) \tag{5.24}$$

holds. (5.24) is obtained from (5.23) by the substitution $\bar{x} \mapsto \bar{z}$. Since (5.23) holds, (5.24) holds. Therefore $\tau(j\bar{y}, \bar{v})$ holds as desired. □

Proof. (**Theorem 5.13**) At the beginning and end of every step of the DeSkolemize we have constraints that are equivalent to each other. This is obvious for some steps which do nothing other than verify that some condition holds and is easy to verify for all other steps except the step which replaces functions with ∃-variables. We will call the constraints with functions just before this step Λ and those with ∃-variables just after this step Λ'. We need to show that $\Lambda \equiv \Lambda'$. The direction $\Lambda \models \Lambda'$ is easy; all we need to do is set the ∃-variables to the values given by the corresponding functions. The direction $\Lambda' \models \Lambda$ is harder. We have done some of the previous steps, in particular the combining of dependencies, to ensure this holds. So suppose $D \models \Lambda'$ and that v is the ∃-variable that corresponds to the function f_v. We set $f_v(\bar{c})$ to a value of v which witnesses that a constraint ψ_Φ holds for $\bar{c}$ where ψ_Φ is as defined in the step "combine dependencies" and where the premise of ψ_Φ holds for $\bar{c}$, yet the premise of no constraint $\psi_{\Phi'}$ with $\Phi \subset \Phi'$ holds for $\bar{c}$. Clearly, there is a unique such set Φ, since if both ψ_{Φ_1} and ψ_{Φ_2} hold for $\bar{c}$, then $\psi_{\Phi_1 \cup \Phi_2}$ also holds for $\bar{c}$. Now assume that the premise of some constraint $\phi \in \Lambda$ in which f appears holds for a tuple $\bar{c}$. Then we must have $\phi \in \Phi$ and since ψ_Φ holds and its premise holds for $\bar{c}$, its conclusion must also hold for $\bar{c}$. Since we have set $f_v(\bar{c})$ to a value that witnesses this, the conclusion of ϕ must also hold for $\bar{c}$. □

Proof. (**Theorem refpr-deskol-termin**) Procedure DeSkolemize may only abort at Step 2, 3, 4, 7, or 8 and it has no loops that may not terminate. Therefore, if Step 9 is reached, DeSkolemize will terminate. Furthermore, all steps can be carried out in polynomial time, except for Step 9, which may give an exponential increase in the size of the constraints. However, if the hypothese of part 2 hold, then no such exponential increase can occur. □

Proof. (**Theorem 5.15**) Set $[k] := \{1, \ldots, k\}$. Consider the TGD-mappings m_{12}^k and m_{23}^k given by $(\sigma_1^k, \sigma_2^k, \Sigma_{12}^k)$ and $(\sigma_2^k, \sigma_3^k, \Sigma_{23}^k)$ where

Σ_{12}^k is	$R_0(x)$	$\rightarrow$	$\exists y\, S_0(x, y)$
	$R_i(x)$	$\rightarrow$	$S_i(x)$
Σ_{23}^k is	$S_0(xy), S_i(x)$	$\rightarrow$	$T_i(y)$

for $i \in [k]$ and where $\sigma_1^k = \{R_i : i \in \{0, \ldots, k\}\}$, $\sigma_2^k = \{S_i : i \in \{0, \ldots, k\}\}$, and $\sigma_3^k = \{T_i : i \in [k]\}$. The SkTGD-composition $m_{13}^k := m_{12}^k \circ m_{13}^k$ is given by the set Σ_{13}^k of constraints

$$R_0(x), y = f(x), R_i(x) \rightarrow T_i(y)$$

for $i \in [k]$, which grows linearly in the size of $\Sigma_{12}^k \cup \Sigma_{23}^k$.

The TGD-composition $m_{13}^k := m_{12}^k \circ m_{13}^k$ can be obtained by de-Skolemizing Σ_{13}^k. It is given by the set $\Sigma_{13}'^k$ of $2^k - 1$ constraints

$$R_0(x), R_Z(x) \rightarrow \exists y\, T_Z(y)$$

such that $\emptyset \neq Z \subseteq [k]$ where $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$ and $T_Z(x) := \bigwedge_{i \in Z} T_i(x)$. On the other hand, m_{13}^k cannot be expressed by any $(\sigma_1, \sigma_3, \Sigma)$, $\Sigma \subseteq$ TGD) where Σ has fewer than 2^{k-1} constraints. This inexpressibility result is given at the end of Section 5.5, in which we introduce a mechanism that enables us to complete this proof. □

5.5 Inexpressibility tool for embedded dependencies

In this section we develop a formal vehicle for proving inexpressibility results for ED, TGD, FuD, and FTGD mappings. We use it to show the inexpressibility claim of Theorem 5.15. However, the tools presented in this section may be of independent value and could be used for obtaining inexpressibility results for other problems in database theory.

To illustrate the intuition behind this mechanism, consider a set Σ of logical sentences over σ, and some structure A_0 over σ. The truth value of Σ in the structure A_0 is $A_0 \models \Sigma$. Now suppose we add one or more tuples to some relation in A_0 and obtain the structure A_1. The truth value of Σ in A_1 may remain the same, or may flip from 'true' to 'false', or from 'false' to 'true'. As we keep adding tuples, we obtain a chain of successively larger structures $A_0, \ldots, A_n, \ldots$ with the corresponding truth values of Σ for each structure in the chain. The truth values of Σ form contiguous segments within which Σ remains 'true' (positive segments) or 'false' (negative segments). For example, the truth

values ('true','true','false','false','true') for a chain of structures $(A_0, A_1, A_2, A_3, A_4)$ partition the chain into three segments: a positive segment (A_0, A_1) followed by a negative segment (A_2, A_3) followed by a positive segment (A_4). To characterize Σ, we count the maximal number n of negative segments for *any* such chain of structures over Σ. If this number is finite, we call Σ n-monotonic, and non-monotonic otherwise. To characterize a class of constraints, we study the monotonicity properties of its constituent sentences.

Next we give formal definitions of chains, segments, and n-monotonic sentences.

Definition 5.4. Let $\mathcal{K}$ be a set of structures over σ and Σ be a set of sentences over σ. Then, for each pair of structures $A, B \in \mathcal{K}$

1. $A \subseteq B$, if for all relation symbols R over σ, $R^A \subseteq R^B$
2. $A \subset B$, if $A \subseteq B$ and $A \neq B$
3. $A \simeq_\Sigma B$, if $(A \models \Sigma$ iff $B \models \Sigma)$ and $\forall C \in \mathcal{K}(A \subseteq C \subseteq B \vee B \subseteq C \subseteq A \rightarrow A \models \Sigma$ iff $C \models \Sigma)$

Definition 5.5. A set $\mathcal{K}$ of structures over signature σ is a *chain* if $(\mathcal{K}, \subset)$ is a total order.

Definition 5.6. A *segment* is an equivalence class of $(\mathcal{K}, \simeq_\Sigma)$. Segment $\mathcal{S}$ is *positive* for Σ if $A \models \Sigma$ for all $A \in \mathcal{S}$, and *negative* otherwise.

Definition 5.7. Let Σ be a set of sentences.

1. Σ is *n-monotonic* if every chain for Σ has at most n negative segments.
2. Σ is *strictly n-monotonic* if Σ is n-monotonic and there exists a chain for Σ with exactly n negative segments.
3. Σ is *monotonic* if it is n-monotonic for some $n \in \mathbb{N}$.

We proceed to study the monotonicity properties of the mapping languages that we focus on in this article. First, we illustrate some 0-monotonic, 1-monotonic, and non-monotonic sentences, to familiarize the reader with the concept.

Example 5.10. $\Sigma = \{R(x) \rightarrow R(x)\}$ is 0-monotonic. More generally, Σ is 0-monotonic if and only if Σ is a tautology. In fact, if Σ is a tautology, then $A \models \Sigma$ for all A over the signature of Σ, i.e., each chain of structures for Σ contains a single positive segment. Conversely, if no chain contains a negative segment, so $A \models \Sigma$ for all A and hence Σ is a tautology.

Example 5.11. $\Sigma = \{R(x) \rightarrow \exists y S(y)\}$ is 1-monotonic. The sentence in Σ is equivalent to $R \neq \emptyset \rightarrow S \neq \emptyset$. Hence, Σ partitions each chain $\mathcal{K}$ into at most three segments containing structures $(\emptyset, \emptyset), (R, \emptyset), (R', S)$, respectively, for some non-empty R, R', S, $R \subseteq R'$. The structure $(R, \emptyset)$ belongs to the only negative segment in such a chain.

Example 5.12. $\Sigma = \{R(x) \rightarrow S(x)\}$ is non-monotonic. Let $A_k = (\{c_0, \ldots, c_k\}, \{c_0, \ldots, c_{k-1}\})$, $B_k = (\{c_0, \ldots, c_k\}, \{c_0, \ldots, c_k\})$ be structures over $\sigma = \{R, S\}$ where all c_i constants are distinct. Clearly, $A_k \not\models \Sigma$, $B_k \models \Sigma$, $A_k \subset B_k \subset A_{k+1}$. That is, there exists a chain A_0, B_0, A_1, B_1, ..., A_k, B_k, ... of singleton segments that witnesses non-monotonicity of Σ.

We generalize the above examples for several classes of sentences. First, we consider tuple-generating dependencies that may only contain dependent (see definition below) existential variables. We show that such sentences are 0-monotonic or non-monotonic (Lemma 5.18). We will see that this class of sentences, which subsumes full tuple-generating dependencies, is the only source of non-monotonic dependencies that make up the constraints in our mapping languages.

Second, we examine tuple-generating dependencies that express inclusions of boolean conjunctive queries, and prove that these are 1-monotonic (Lemma 5.21). Third, we prove the same monotonicity property for equality-generating dependencies (Lemma 5.22). After examining the above classes of constraints, we prove Lemma 5.24 that establishes a monotonicity bound for a set of sentences based on the monotonicity of its members. These lemmas lead to the main result of this section, which states the monotonicity properties for the (first-order) mapping languages that we consider.

We call a variable $y \in \{\bar{y}\}$ *dependent* in a tgd $\varphi = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y} Q(\bar{x}, \bar{y}))$ if there exist variables $u_1, \ldots, u_n$ such that u_i and u_{i+1} appear in the same atom of Q, $1 \leq i < n$, and $u_1 = y$, $u_n \in \{\bar{x}\}$. Otherwise, y is called independent. We start with tuple-generating dependencies φ that may only contain dependent existential variables.

Lemma 5.18. *Let $\varphi = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y} Q(\bar{x}, \bar{y}))$ be a tuple-generating dependency such that each $y \in \{\bar{y}\}$ is dependent. Then, φ is 0-monotonic or non-monotonic.*

Proof. Assume φ is not 0-monotonic, i.e., it is not a tautology. The proof of non-monotonicity is based on the following two observations:

- If φ is false in some structure A, then there exists a larger structure $B \supset A$ that makes φ true (Lemma 5.19), and

- If φ is true in some structure B, then there exists a larger structure $A \supset B$ that makes φ false (Lemma 5.20).

Together, the above observations assert the existence of an infinite chain of alternating structures that witness non-monotonicity of φ. □

We prove the subordinate Lemmas 5.19 and 5.20. Notice that Lemma 5.19 applies to arbitrary tuple-generating dependencies.

Lemma 5.19. *If a tuple-generating dependency φ is violated in some structure A, then there exists a larger structure $B \supset A$ that satisfies the dependency. That is, $\forall A : A \not\models \varphi \rightarrow \exists B(B \models \varphi \wedge A \subset B)$.*

Proof. Let A be a structure such that $A \not\models \varphi$. Construct a complete structure B in which every relation R^B is a cross-product over the domain of A. Clearly, $B \models \varphi$, $A \subseteq B$. Since $A \not\models \varphi$, so $A \neq B$ and we obtain $A \subset B$. □

Lemma 5.20. *Let $\varphi = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y} Q(\bar{x}, \bar{y}))$ be a tuple-generating dependency such that each $y \in \{\bar{y}\}$ is dependent, and φ is not a tautology. Then, if φ is true in some structure B, then there exists a larger structure $A \supset B$ that violates the dependency. That is, $\forall B : B \models \varphi \rightarrow \exists A(A \not\models \varphi \wedge B \subset A)$.*

Proof. Let B be a structure such that $B \models \varphi$. Since φ is not a tautology, there exists a structure F_0 such that $F_0 \not\models \varphi$. F_0 is non-empty (since the empty structure makes φ true). Let F be an isomorphic copy of F_0 that does not have any constants in common with B. Set $A := B \cup F$. Clearly, $A \supset B$. We show that $A \not\models \varphi$.

Since $F_0 \not\models \varphi$, so $F \not\models \varphi$. Let $\bar{t}$ be a tuple such that $F \models P(\bar{t})$, $F \not\models \exists \bar{y} Q(\bar{t}, \bar{y})$. Such a tuple exists since F is non-empty. Given that $F \models P(\bar{t})$, we obtain $A = B \cup F \models P(\bar{t})$ because P is a conjunctive query.

There remains to show that $A = B \cup F \not\models \exists \bar{y} Q(\bar{t}, \bar{y})$. Suppose the contrary, i.e., $A \models \exists \bar{y} Q(\bar{t}, \bar{y})$. Then, there is an atom $R(\bar{z})$ in Q where $\bar{z} \subseteq \bar{x} \cup \bar{y}$, and a tuple $\bar{r}$ such that $F \not\models R(\bar{r})$ and $B \cup F \models R(\bar{r})$. Since B and F do not share constants, $R(\bar{r})$ must contain only constants from B. In other words, $\bar{z} \subseteq \bar{y}$, i.e., R contains only $\bar{y}$ variables, and $\bar{z} \neq \emptyset$. By the premise of the lemma, each $\bar{y}$ variable is dependent. That is, there exists an atom in Q that contains both $\bar{x}$ and $\bar{y}$ variables. However, this atom is satisfied in A only if $B \cap F \neq \emptyset$, a contradiction. □ □

To illustrate Lemma 5.20, consider the dependency $\forall xy(R(xy) \rightarrow \exists z(R(xz), S(z))$. Since z is connected, the dependency satisfies the premise of the lemma and is non-monotonic.

Next we consider sentences φ that express inclusions of boolean conjunctive queries. We show that such sentences are 1-monotonic.

Lemma 5.21. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \rightarrow \exists\bar{y}Q(\bar{y}))$. Then, φ is 1-monotonic.*

Proof. Observe that φ is equivalent to the sentence $\psi \rightarrow \theta$, where $\psi = \exists\bar{x}P(\bar{x})$ and $\theta = \exists\bar{y}Q(\bar{y})$ are boolean conjunctive queries.

We need to show that each chain for φ contains at most one negative segment. Assume the opposite, i.e., there exists a chain that contains structures $A \subset B \subset C$ such that $A \not\models \varphi$, $B \models \varphi$, $C \not\models \varphi$ (where A and C belong to two distinct negative segments). Then, the following formula must be true:

$$(A \models \psi \wedge A \not\models \theta) \wedge (B \not\models \psi \vee B \models \theta) \wedge (C \models \psi \wedge C \not\models \theta)$$

Since $A \subset B$, so $A \models \psi$ implies $B \models \psi$. That is, $B \models \theta$ must hold to make the disjunction true. But since $B \subset C$, $B \models \theta$ implies $C \models \theta$. This contradicts $C \not\models \theta$. Hence, our assumption was false, and φ is 1-monotonic. □

As a last building block, we consider a generalized form of equality-generating dependencies, where multiple equality atoms may appear in the conclusion, and show that these are 1-monotonic.

Lemma 5.22. *Let $\varphi = \forall\bar{x}(P(\bar{x}) \rightarrow \psi(\bar{x}))$, where $P(\bar{x})$ is a conjunctive query and $\psi(\bar{x})$ is a set of equalities between variables in $\bar{x}$. Then, φ is 1-monotonic.*

Proof. We need to show that each chain for φ contains at most one negative segment. Consider a chain for φ that contains a structure A such that $A \not\models \varphi$. Then, there exists a tuple t that satisfies the premise, $A \models P(t)$, but violates the equality conditions, $A \not\models \psi(t)$. Once the equality conditions are violated, $\psi(t)$ remains false in every larger structure. That is, for every structure $B \supset A$, $B \models P(t)$ and $B \not\models \psi(t)$, and hence $B \not\models \varphi$. Therefore, each chain over φ contains at most one negative segment. Hence, φ is 1-monotonic. □

We examined the building blocks of our mapping languages. The following three lemmas explain how to put these building blocks together.

Lemma 5.23. *A set Σ of embedded dependencies is equivalent to a set Σ' of embedded dependencies such that each $\psi \in \Sigma'$ satisfies the premise of one of the Lemmas 5.18, 5.21, or 5.22.*

Proof. Let $\varphi \in \Sigma$, $\varphi = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}\psi(\bar{x}, \bar{y}))$. Construct φ_1 by eliminating all equality atoms from φ that involve variables from $\bar{y}$ as follows: if $y \in \bar{y}$ appears in an equality atom $y = z$ in ψ, replace all occurrences of y in ψ by z and remove $y = z$. Clearly, $\varphi \equiv \varphi_1$. If φ_1 has no equality atoms left, it is a tuple-generating dependency; set $\Sigma_\varphi = \{\varphi_1\}$. Otherwise, since each equality atom in φ_1 mentions only variables in $\bar{x}$, so φ_1 is equivalent to $\{\varphi_2, \varphi_3\}$ where φ_2 is a tuple-generating dependency and φ_3 is a generalized equality-generating dependency satisfying the premise of Lemma 5.22; set $\Sigma_\varphi = \{\varphi_2, \varphi_3\}$.

Let $\varphi' = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}Q(\bar{x}, \bar{y}))$ be the tuple-generating dependency in Σ_φ. If each $y \in \{\bar{y}\}$ is dependent (or φ' is a full tgd), then φ' satisfies the premise of Lemma 5.18. Otherwise, $\varphi' = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}_1, \bar{y}_2(Q_1(\bar{x}, \bar{y}_1), Q_2(\bar{y}_2))$ where $\bar{y}_1$ contains only dependent variables and $\bar{y}_2$ contains only independent variables. Consequently, $\varphi' \equiv \{\varphi_a, \varphi_b\}$ where $\varphi_a = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}_1 Q_1(\bar{x}, \bar{y}_1))$ satisfies the premise of Lemma 5.18 and $\varphi_b = \forall \bar{x}(P(\bar{x}) \rightarrow \exists \bar{y}_2 Q_2(\bar{y}_2))$ satisfies the premise of Lemma 5.21. □ □

Lemma 5.24 establishes an upper bound on the combined monotonicity for a set of arbitrary monotonic sentences.

Lemma 5.24. *If every sentence $\varphi \in \Sigma$ is n_φ-monotonic, then Σ is n-monotonic where $n = \sum_{\varphi \in \Sigma} n_\varphi$.*

Proof. Let $\mathcal{K}$ be a chain for Σ. Σ partitions $\mathcal{K}$ into a set of disjoint segments $(\mathcal{K}, \simeq_\Sigma)$. Each sentence $\varphi_i \in \Sigma$ partitions $\mathcal{K}$ into a set of disjoint segments $(\mathcal{K}, \simeq_{\varphi_i})$. Let $\mathcal{S}$ be a negative segment of $(\mathcal{K}, \simeq_\Sigma)$, and let $A \in \mathcal{S}$, $A \not\models \Sigma$. Hence, there exists some φ_i such that $A \not\models \varphi_i$. Therefore, $\mathcal{S}$ overlaps with some negative segment $\mathcal{S}_i \in (\mathcal{K}, \simeq_{\varphi_i})$ that contains A. Notice that for each $B \in \mathcal{S}_i$, $B \not\models \varphi_i$ implies $B \not\models \Sigma$. Therefore, $\mathcal{S}_i \subseteq \mathcal{S}$. That is, each negative segment of $(\mathcal{K}, \simeq_\Sigma)$ fully contains a negative segment of $(\mathcal{K}, \simeq_\varphi)$ for some $\varphi \in \Sigma$. Since all segments of $(\mathcal{K}, \simeq_\Sigma)$ are disjoint, there are at most as many negative segments in $(\mathcal{K}, \simeq_\Sigma)$ as the cumulative number n of negative segments in $(\mathcal{K}, \simeq_\varphi)$ for all $\varphi \in \Sigma$. Each $\varphi \in \Sigma$ is n_φ-monotonic, so $(\mathcal{K}, \simeq_\varphi)$ contains at most n_φ negative segments. Therefore, $n \leq \sum_{\varphi \in \Sigma} n_\varphi$. □

We would like to make an equally general statement for the case where Σ contains a non-monotonic dependency. However, it seems difficult to do so for an arbitrary class of dependencies. Although a non-monotonic dependency has chains with an unbounded number of negative segments, it is possible that the monotonic sentences in Σ

'wipe out' all but a finite number of negative segments in each such chain. Therefore, in Lemma 5.25 we focus specifically on embedded dependencies.

Lemma 5.25. *Let Σ be a set of embedded dependencies in which each dependency satisfies the premise of one of the Lemmas 5.18, 5.21, or 5.22. Then Σ is non-monotonic or n-monotonic where $n = \sum n_\varphi$ for all monotonic $\varphi \in \Sigma$.*

Proof. If Σ has an implied dependency φ, the monotonicity of Σ is identical with that of $\Sigma - \{\varphi\}$. So, without loss of generality we assume that Σ does not contain any implied dependencies.

If Σ has monotonic dependencies only, the statement of the lemma follows from Lemma 5.24. Otherwise, the only source of non-monotonicity are the dependencies φ satisfying the premise of Lemma 5.18. Let $\varphi \in \Sigma$ be such a non-monotonic dependency. Further, let $\Sigma_r = \Sigma - \{\varphi\}$. Since Σ does not contain implied dependencies, so $\Sigma_r \not\models \varphi$.

To show non-monotonicity of Σ, we construct an unbounded chain of structures similarly to how this is done in Lemma 5.18, but using a modified mechanism that preserves the truth value of Σ_r as an invariant in each structure of the chain.

Let A be a structure such that $A \not\models \varphi$, $A \models \Sigma_r$. We construct a structure B such that $B \models \Sigma$, $A \subset B$ by chasing the constraints in Σ, such that the values for existentially quantified variables are drawn from the existing constants in A. Since A is finite, the process terminates yielding a finite $B \models \Sigma$. The chase adds at least one new tuple to some relation in B to make φ true, hence $A \subset B$.

Now, let B be a structure such that $B \models \Sigma$. Since $\Sigma_r \not\models \varphi$, there exists a structure F_0 such that $F_0 \not\models \varphi$ and $F_0 \models \Sigma_r$. We construct $A \not\models \varphi$ as in Lemma 5.20 for each B using this fixed F_0. Since $F_0 \models \Sigma_r$, so $A \models \Sigma_r$.

Together, these constructions witness non-monotonicity of Σ. □

Now we are ready to state the main result of this section. The case analysis in the following theorem is based on Lemma 5.23.

Theorem 5.26.

1. *Each FTGD dependency is 0-monotonic or non-monotonic.*
2. *Each FuD dependency is 1-monotonic or non-monotonic.*
3. *Each TGD dependency is 1-monotonic or non-monotonic.*
4. *Each ED dependency is 2-monotonic or non-monotonic.*

Proof.

1. Follows immediately from Lemma 5.18.

2. Let $\varphi \in$ FuD. Suppose $\varphi \notin$ FTGD. Then, φ is equivalent to a set $\Sigma = \{\psi_1, \psi_2\}$ of constraints where $\psi_1 \in$ FTGD and ψ_2 is a generalized equality-generating dependency (possibly with multiple equality atoms in the conclusion). ψ_1 is 0-monotonic or non-monotonic. By Lemma 5.22, ψ_2 is 1-monotonic. Hence, by Lemmas 5.24 and 5.25, φ is 1-monotonic or non-monotonic.

3. Let $\varphi \in$ TGD. Then, three cases are possible: (a) $\varphi \in$ FTGD, (b) φ is an inclusion of boolean conjunctive queries, or (c) φ is equivalent to $\{\psi_1, \psi_2\}$ where $\psi_1 \in$ FTGD and ψ_2 is an inclusion of boolean conjunctive queries. In case (a), φ is 0-monotonic or non-monotonic. In case (b), φ is 1-monotonic by Lemma 5.21. In case (c), φ is 1-monotonic or non-monotonic by Lemmas 5.22, 5.24, and 5.25. Hence, φ is 1-monotonic or non-monotonic.

4. Let $\varphi \in$ ED. Suppose $\varphi \notin$ TGD. Then, φ is equivalent to a set $\Sigma = \{\psi_1, \psi_2\}$ of constraints where $\psi_1 \in$ TGD and ψ_2 is a generalized equality-generating dependency. ψ_1 is 1-monotonic or non-monotonic. ψ_2 is 1-monotonic. Hence, by Lemmas 5.24 and 5.25, φ is 2-monotonic or non-monotonic. □

□

Example 5.13. To illustrate a 2-monotonic ED-dependency, consider $\varphi = \forall x, y(R(x, y) \rightarrow \exists z(S(z) \wedge x = y))$. The chain of structures $A_0 = (\{(a, a)\}, \emptyset)$, $A_1 = (\{(a, a)\}, \{b\})$, $A_2 = (\{(a, a), (a, b)\}, \{b\})$ contains two negative segments.

As an application of the inexpressibility tools presented in this section, we complete the proof of Theorem 5.15 from Section 5.4.

Proof. (Inexpressibility claim of **Theorem 5.15**) Let $[k] = \{1, \ldots, k\}$ and let φ_Z be a constraint of the form

$$R_0(x), R_Z(x) \rightarrow \exists y\, T_Z(y)$$

where $Z \subseteq [k]$, $R_Z(x) := \bigwedge_{i \in Z} R_i(x)$, and $T_Z(x) := \bigwedge_{i \in Z} T_i(x)$.

Let $\Sigma = \{\varphi_Z : \emptyset \neq Z \subseteq [k]\}$.

Each dependency $\varphi_Z \in \Sigma$ specifies inclusion of boolean conjunctive queries and is hence 1-monotonic by Lemma 5.21. Σ contains a total of $2^k - 1$ dependencies. Therefore, by Lemma 5.24, Σ is at most $(2^k - 1)$-monotonic.

We show that Σ is strictly (2^k-1)-monotonic. Let Z_c denote the subset of $[k]$ where index c is the binary encoding of Z, i.e., $c = \sum_{j \in Z} 2^j$. Then, $Z_0, \ldots, Z_{2^k}$ is an enumeration of subsets of $[k]$. The enumeration has the property that $Z_{c_2} \not\subseteq Z_{c_1}$ for any $c_1 < c_2$. We construct a chain of structures for Σ that contains $2^k - 1$ negative segments by induction.

- Let A_0 be the empty structure for Σ, $A_0 \models \Sigma$.
- Let $A_c \models \Sigma$ be a structure obtained in induction step c. Construct B_{c+1} by copying all relations from A_c and adding the constant c to relation R_0 and all relations R_i such that $i \in Z_{c+1}$. The added constant c violates exactly one dependency $\varphi_{Z_{c+1}} \in \Sigma$. No other dependency gets violated since $Z_{c_{i+1}} \not\subseteq Z_j$ for any $j \leq c$. We have $B_{c+1} \not\models \Sigma$, $B_{c+1} \supset A_c$.
- Let $B_c \not\models \Sigma$ be a structure obtained in induction step c, $c \geq 1$. Construct A_c by copying all relations from B_c and setting $T_i := R_i$ for all $i \in [k]$. This construction makes all dependencies of Σ satisfied in A_c. Hence, $A_c \models \Sigma$, $A_c \supset B_c$.

The constructed chain $B_1, A_1, \ldots, A_{2^k-1}, B_{2^k}$ witnesses that Σ is strictly $(2^k - 1)$-monotonic. By Theorem 5.26, each TGD-dependency is 2-monotonic or non-monotonic. Therefore, by Lemmas 5.24 and 5.25, at least $\lceil (2^k-1)/2 \rceil = 2^{k-1}$ embedded dependencies are needed to express Σ. By the same theorem, Σ is not expressible by any set of full dependencies. □

5.6 Semantics of SkED Constraints

In this section we examine the semantics of SkED constraints. The semantics of SkED are somewhat special, but seem to be needed to obtain domain independence (Example 5.14).

We introduce another fragment of SO, ∃SOED which has the standard second-order semantics. We show that source-to-target SkED-mappings are also source-to-target ∃SOED-mappings (Theorem 5.27).[2] However, this translation may incur and exponential increase in size.

We first discuss the semantics of SkED constraints. The main question is, what is the universe from which the functions can take values? That is, what is their allowed range? Intuitively, the problem is with the universe of the existentially quantified intermediate database.

[2] when restricted to structures with at least two elements in the active domain

Example 5.14. Consider the FTGD-mappings m_{12}^k and m_{23}^k given by $(\sigma_1, \sigma_2^k, \Sigma_{12}^k)$ and $(\sigma_2^k, \sigma_3, \Sigma_{23}^k)$ where

Σ_{12}^k is	$R(x)$	$\rightarrow$	$\exists y S_i(y)$
Σ_{23}^k is	$S_i(x), S_j(x)$	$\rightarrow$	$T(x)$

for $1 \leq i, j \leq k$ and $i \neq j$, where $\sigma_1 = \{R\}$, $\sigma_2^k = \{S_1, \ldots, S_k\}$, and $\sigma_3 = \{T\}$.

Consider the case where $R = \{1, \ldots, k-1\}$ in A and T is empty in C. Firstly, notice that $(A, C) \in m_{12}^k \circ m_{23}^k$ as witnessed by the database B where $S_i = \{i\}$.

Skolemizing and composing the constraints above we obtain Σ_{13}^k given by the set of constraints

$$\{R(x), R(y), f_i(x) = z, f_j(y) = z \rightarrow T(z) : 1 \leq i, j \leq k, i \neq j\}$$

where f_i is the Skolem function corresponding to $\exists y S_i(y)$. If we restrict the range of every f_i to fall within the domain of A and C, then one of the constraints above must fail, as follows. Consider the case where $x = 1$ and $y = 1$. Then the set $\{f_i(1) : 1 \leq i \leq k\}$ must be a subset of $\{1, \ldots, k-1\}$. By the pigeonhole principle, there must be i and j such that $i \neq j$ and $f_i(1) = f_j(1)$. Then the constraint corresponding to such i, j fails for (A, C), since T is empty in C. Therefore, $(A, C) \not\models \Sigma_{13}^k$. On the other hand, if we keep the same relations R and T, but allow the domain to have at least k values, then we have $(A, C) \models \Sigma_{13}^k$ witnessed by setting $f_i(x) = i$ for all i, x.

This shows that if we restrict the range of the Skolem functions to be domain of the input structures, then we may have domain-dependent formulas, even though they satisfy the safety conditions. Certainly, no such domain-dependent formulas can express the composition, since whether (R, T) belong to the composition or not does not depend on their domains. □

Therefore we require all databases to be finite (i.e., all relations are finite), but to have an implicit countably infinite universe. Notice that no finite domain would work for all constraints since the example above gives a family of sets of constraints for which the meaning changes depending on whether the domain has size less than k or not. We allow the functions to take any values from this implicit universe.

Since the semantics of SkED are special, it is natural to ask whether the constraints in SkED can be expressed in some fragment of ∃SO under the usual second-order semantics. We show that this is possible for source-to-target SkED.

Theorem 5.27. *Every finite set of source-to-target SkED constraints (under the semantics described above) is equivalent to a finite set of source-to-target ∃SOED constraints*

(under the usual second-order semantics) when restricted to instances with at least two elements.

Proof. (Outline) In the case of source-to-target constraints, we know that we do not have "recursive" Skolem terms. That is, there are no Skolem terms of the form $f(\dots f \dots)$ (directly, or indirectly through equalities). Therefore, there is a finite number of values we can refer to by building Skolem terms on top of the elements of the domain. Intuitively, these are all the elements that the intermediate database needs to have and the worst case is when they are all different. If the domain has n elements and we have p Skolem functions of arity q, then an easy upper bound on the number of elements we can refer to is $\leq n^{(p+q)^p}$ whenever $n \geq 2$. (This is shown by induction depth of the Skolem terms; at each step we go from $m \geq n$ possible values to

$$m + pm^q \leq (p+1)m^q \leq 2^p m^q \leq n^p m^q \leq m^{(p+q)}$$

possible values.) Therefore, we can encode all these values with tuples of arity $r = (p+q)^p$. We encode every value c from the original domain as the tuple $(c, \dots, c)$; that is, c repeated r times.

Given a finite set Σ of source-to-target SkED (which we assume w.l.o.g. to be in unnested form), we first compute r, then transform each constraint $\phi \in \Sigma$ by replacing every occurrence of an equation of the form $f(\bar{x}) = y$ with $F(\bar{x}, \bar{y})$ where $\bar{y}$ is a tuple of arity r. We also replace y with $\bar{y}$ everywhere except in relational atoms. To every relational atom, we add a set of equalities of the form $y = y_1, \dots, y = y_k$ which we abbreviate $y = \bar{y}$. Finally, we add constraints of the form

$$\dots \rightarrow \exists \bar{y} F(\bar{x}, \bar{y})$$

$$F(\bar{x}, \bar{y}), F(\bar{x}, \bar{y}') \rightarrow \bar{y} = \bar{y}'$$

where $\dots$ are obtained from any premises which mention f. Notice that we need both equalities and FO existential quantifiers in the conclusions and that we may incur an exponential increase in size since we need r to be exponential in p. □

Notice that the proof only requires that we do not have "recursive" Skolem terms. Source-to-target is a strong condition that ensures this, but weaker conditions on the set of constraints $\Sigma_{12} \cup \Sigma_{23}$ suffice. For example, it is enough to require that $\Sigma_{12} \cup \Sigma_{23}$ have *stratified witnesses* (see [9] and [11]). When such conditions hold, we can compose

∃SOED-mappings using a technique similar to that of the proof of Theorem 5.27. In this case, we do not Skolemize, but replace every relation S of arity s from σ_2 with an existentially quantified relation R of arity rs (where r is as in the proof of Theorem 5.27). Then we replace every occurrence of a universally quantified variable v in S with $\bar{v}$ and add the equation $v = \bar{v}$ and replace every occurrence of an existentially quantified variable u with $\bar{u}$. This gives an algorithm for composition of source-to-target ∃SOED-mappings.

5.7 Other Basic Operators

In addition to composition, we are interested in several other basic operators including domain, range, intersection, cross-product, and inverse. These operators take for input mappings and models and give as output mappings or models (a *model* is a set of instances). The following table summarizes the definitions of these basic operators:

$\mathrm{dom}(m) :=$	$\{A : \exists B\, \langle A, B\rangle \in m\}$.
$\mathrm{rng}(m) :=$	$\{B : \exists A\, \langle A, B\rangle \in m\}$.
$\mathcal{A} \cap \mathcal{B} :=$	$\{A : A \in \mathcal{A}, A \in \mathcal{B}\}$.
$\mathrm{id}(\mathcal{A}) :=$	$\{\langle A, A\rangle : A \in \mathcal{A}\}$.
$\mathcal{A} \times \mathcal{B} :=$	$\{\langle A, B\rangle : A \in \mathcal{A}, B \in \mathcal{B}\}$.
$m_1 \cap m_2 :=$	$\{\langle A, B\rangle : \langle A, B\rangle \in m_1, \langle A, B\rangle \in m_2\}$.
$m^{-1} :=$	$\{\langle B, A\rangle : \langle A, B\rangle \in m\}$.

As in the case of mappings, we say that a model $\mathcal{A}$ is given by (σ_1, Σ_1) if it consists exactly of those databases over the signature σ_1 which satisfy the constraints Σ_1. If, furthermore, Σ_1 is finite subset of $\mathcal{L}$ we say that $\mathcal{A}$ is an $\mathcal{L}$-model. As in the case of composition, we say that $\mathcal{L}$ is *closed* under one of these operators if it produces an $\mathcal{L}$-model or $\mathcal{L}$-mapping whenever the inputs are compatible $\mathcal{L}$-models or $\mathcal{L}$-mappings.

Proposition 5.28. *Every $\mathcal{L} \supseteq$ FTGD is closed under identity, cross product and intersection.*

Proof. If m_{12} and m_{34} are given by $(\sigma_1, \sigma_2, \Sigma_{12})$ and $(\sigma_3, \sigma_4, \Sigma_{34})$ and $\mathcal{A}$ and $\mathcal{B}$ are given by (σ_1, Σ_1) and (σ_2, Σ_2), then

- $\mathcal{A} \times \mathcal{B}$ is given by $(\sigma_1, \sigma_2, \Sigma_1 \cup \Sigma_2)$.
- $\mathcal{A} \cap \mathcal{B}$ is given by $(\sigma_1, \Sigma_1 \cup \Sigma_2)$ (here $\sigma_1 = \sigma_2$).
- $m_{12} \cap m_{34}$ is given by $(\sigma_1, \sigma_2, \Sigma_{12} \cup \Sigma_{34})$ (here $\sigma_1 = \sigma_3$ and $\sigma_2 = \sigma_4$).

To express identity we need to refer to the third auxiliary signature σ_2' (which we normally ignore) which contains, for every relation symbol R in σ_2, a relation symbol R' of the same arity. In this case, $\sigma_1 = \sigma_2$ so $\sigma_2' = \sigma_1'$.

- $\mathrm{id}(\mathcal{A})$ is given by $(\sigma_1, \sigma_1, \sigma_1', \Sigma_1 \cup \Sigma)$ where Σ consists of two constraints of the form

$$\begin{aligned} \forall(\bar{x})(R(\bar{x}) &\rightarrow R'(\bar{x})) \\ \forall(\bar{x})(R'(\bar{x}) &\rightarrow R(\bar{x})) \end{aligned}$$

  for every R in σ_1. □

□

Proposition 5.29. *Each one of the operators composition, range, and domain can be reduced to any one of the others.*

Proof. If

- m_{12} is given by $(\sigma_1, \sigma_2, \Sigma_{12})$,
- m_{23} is given by $(\sigma_2, \sigma_3, \Sigma_{23})$,
- m_1 is given by $(\sigma_1 \cup \sigma_3, \sigma_2, \Sigma_{12} \cup \Sigma_{23})$,
- m_2 is given by $(\sigma_2, \sigma_1 \cup \sigma_3, \Sigma_{12} \cup \Sigma_{23})$,
- $\mathrm{dom}(m_1)$ is given $(\sigma_1 \cup \sigma_3, \Sigma_1)$, and
- $\mathrm{rng}(m_2)$ is given $(\sigma_1 \cup \sigma_3, \Sigma_2)$,

then $m_{12} \circ m_{23}$ is given by $(\sigma_1, \sigma_3, \Sigma_1)$ and $(\sigma_1, \sigma_3, \Sigma_2)$.
Conversely, if

- m_{12} is given by $(\sigma_1, \sigma_2, \Sigma_{12})$,
- m_{21} is given by $(\sigma_2, \sigma_1, \emptyset)$,
- $m_{12} \circ m_{21}$ is given by $(\sigma_1, \sigma_1, \Sigma_1)$, and
- $m_{21} \circ m_{12}$ is given by $(\sigma_2, \sigma_2, \Sigma_2)$,

then $\mathrm{dom}(m_{12})$ and $\mathrm{rng}(m_{12})$ are given respectively by (σ_1, Σ_1) and (σ_2, Σ_2). □

Proposition 5.29 and Theorem 5.4 give the following.

Corollary 5.30. *Checking whether the domain or range of a FTGD-mapping is a FTGD-model is undecidable.*

All the languages we consider satisfy the premises of Proposition 5.28. Therefore, Proposition 5.29 indicates that we can concentrate our attention on closure under composition and inverse. Notice that if an $\mathcal{L}$-mapping m is given by $(\sigma_1, \sigma_2, \Sigma_{12})$, then its inverse is given by $(\sigma_2, \sigma_1, \Sigma_{12})$, which is, of course, easy to compute. However, the restrictions on $\mathcal{L}$ may be such that the second expression no longer gives an $\mathcal{L}$-mapping. For example, this happens with source-to-target constraints. This is why we seek restrictions on $\mathcal{L}$ which are symmetric with respect to the input and output signatures and which guarantee closure under composition.

This chapter is based on "Composition of Mappings Given by Embedded Dependencies" by Alan Nash and Phil Berstein and Sergey Melnik [32] (journal version [33]). I was responsible for developing all the concepts in this paper, except as follows. The section on de-Skolemization is joint work with Sergey Melnik. The main ideas and the presentation for the section on inexpressibility are by Sergey Melnik. Sergey Melnik and Phil Bernstein participated in many discussion where the concepts discussed were clarified.

6

Conclusion

In Chapter 3 we have introduced universal set solutions and their more general counterparts, templates, and we have shown how to use them to compute certain answers. We have shown that any chase-like algorithm produces strong templates and we have presented a new chase, the unordered minimizing chase, which will always find a strong template if there is one. We did not explore the complexity of this new chase. Finding the core of a general structure is NP-complete. However, we hope that the techniques for computing cores of chase results in Chapter 4 can be applied to give improvements in efficiency.

We have also introduced new conditions for chase termination, wider than those previously known. We have shown how to use templates to check containment and containment under constraints. Finally, we have shown that we can relax the notions of universal set solutions and templates to k-universal set solutions and k-templates and have shown that these more relaxed notions are still sufficient to compute certain answers to queries with at most k variables and for testing containment of queries with at most k variables.

We have also shown that the finite and unrestricted versions of templates differ. We have an interesting situation: since we only care about finite instances, we really need weak templates, but all chase-like algorithms produce only strong templates. It is natural to ask whether there are algorithms that can compute weak templates. This remains a hard open problem, connected with the fundamental differences between finite model theory and unrestricted model theory. We would like to close with an intriguing connection.

Given a set of instances K, set $\bar{K} := \{B : \exists A \in K, A \to B\}$. Whether we consider only finite instances or unrestricted instances should be clear from context.

The following result follows from the infinite and finite versions of the "preservation under homomorphisms" theorems, the latter recently proved by Benjamin Rossman [35].

Theorem 6.1.

1. *Σ has a strong template iff $\bar{K}$ is axiomatizable by a first-order sentence, where K is the set of unrestricted models of Σ.*

2. *Σ has a weak template iff $\bar{K}$ is axiomatizable by a first-order sentence where K is the set of finite models of Σ.*

In Chapter 4 we have solved a central problem of the theory of data exchange by proving that cores of data exchange problems can be computed in polynomial time in case the target constraints consist of EGDs and a weakly acyclic set of TGDs. We developed and combined several new ideas for achieving this tractability result. We hope that our result will foster the use of cores in data exchange and will find its way to implementations.

In Chapter 5 we have explored mapping composition. Mapping composition is one of the key operators that are used for manipulating schemas and mappings between schemas. We studied composition of mappings given by embedded dependencies, which are expressive enough for many data management applications. We addressed challenges that were not considered in prior work, in particular the ones due to recursion and de-Skolemization.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.

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

[4] P. A. Bernstein. Applying Model Management to Classical Meta-Data Problems. In *Conference on Innovative Data Systems Research (CIDR)*, pages 209–220, 2003.

[5] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A Vision of Management of Complex Models. *SIGMOD Record*, 29(4):55–63, 2000.

[6] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 77–90, New York, NY, USA, 1977. ACM Press.

[7] Alin Deutsch, Bertram Ludaescher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. In *ICDT*, 2005.

[8] Alin Deutsch and Val Tannen. Mars: A system for publishing xml from mixed and redundant storage. In *VLDB*, pages 201–212, 2003.

[9] Alin Deutsch and Val Tannen. Reformulation of XML Queries and Constraints. In *ICDT*, 2003.

[10] Heinz-Dieter Ebbinhaus and Jörg Flum. *Finite Model Theory*. Springer, 2nd edition, 1999.

[11] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. In *ICDT*, 2003. full version in: Theor. Comput. Sci. 336(1): 89-124 (2005).

[12] R. Fagin, P. G. Kolaitis, and L. Popa. Data Exchange: Getting to the Core. In *ACM PODS*, pages 90–101, 2003. Full version in ACM TODS, 30(1):147-210(2005).

[13] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *ACM PODS*, pages 83–94, 2004.

[14] Ronald Fagin. Extending the core greedy algorithm to allow target tgds with singleton left-hand sides. Unpublished Manuscript, 2005.

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

[16] Ariel Fuxman, Phokion G. Kolaitis, Renée J. Miller, and Wang Chiew Tan. Peer data exchange. In *PODS*, 2005.

[17] G. Gottlob and A. Nash. Data Exchange: Computing Cores in Polynomial Time. In *ACM PODS*, 2006. To appear.

[18] G. Gottlob and A. Nash. Efficient Core Computation in Data Exchange. *Journal of the ACM*, 2006. Submitted.

[19] Georg Gottlob. Computing cores for data exchange: New algoritms and practical solutions. In *PODS*, 2005. Extended version of the present paper. Currently available at: www.dbai.tuwien.ac.at/staff/gottlob/extcore.pdf.

[20] Georg Gottlob and Christian G. Fermüller. Removing redundancy from a clause. *Artif. Intell.*, 61(2):263–289, 1993.

[21] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *JCSS*, 64(3):579–627, 2002.

[22] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. ICDE 2003.

[23] P. Hell and J. Nešetřil. The core of a graph. *Discr. Math.*, 109(1-3):117–126, 1992.

[24] Christoph Koch. Query rewriting with symmetric constraints. In *Foundations of Information and Knowledge Systems (FoIKS)*, 2002.

[25] Phokion Kolaitis. Schema mappings, data exchange and metadata management. In *PODS*, 2005.

[26] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In *International Conference on Very Large Data Bases (VLDB)*, pages 572–583, 2003.

[27] Maier, Sagiv, and Yannakakis. On the complexity of testing implication of functional and join dependencies. *Journal of the ACM*, 1981.

[28] David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

[29] S. Melnik, P. A. Bernstein, A. Y. Halevy, and E. Rahm. Supporting Executable Mappings in Model Management. In *SIGMOD*, pages 167–178, 2005.

[30] S. Melnik, E. Rahm, and P. A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *SIGMOD*, pages 193–204, 2003.

[31] Sergey Melnik. *Generic Model Management: Concepts and Algorithms.* Ph.D. Thesis, Univ. of Leipzig, Springer LNCS 2967, 2004.

[32] Alan Nash, Phil Berstein, and Sergey Melnik. Composition of mappings given by embedded dependencies. In *ACM PODS*, 2005.

[33] Alan Nash, Phil Berstein, and Sergey Melnik. Composition of mappings given by embedded dependencies. *ACM TODS*, 2006. Submitted.

[34] Alan Nash, Alin Deutsch, and Jeff Remmel. Data exchange, data integration, and the chase. 2006. Manuscript, Submitted for publication, 2006.

[35] Benjamin Rossman. Existential positive types and preservation under homomorphisisms. In *LICS*, pages 467–476, 2005.

[36] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *Journal of the ACM*, 27(4):633–655, 1980.

[37] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 2nd edition, 1997.

[38] Moshe Vardi. Inferring multivalued dependencies from functional and join dependencies. *Acta Informatica*, 1983.

[39] Cong Yu and Lucian Popa. Constraint-Based XML Query Rewriting For Data Integration. In *SIGMOD*, pages 371–382, 2004.