Data Exchange, Integration, and Streams

A Volume from DEIS'10 – GI-Dagstuhl Seminar 10452

Edited by

Phokion G. Kolaitis
Maurizio Lenzerini
Nicole Schweikardt

DAGSTUHL
FOLLOW-UPS

Editors

Phokion G. Kolaitis
UC Santa Cruz & IBM Research – Almaden

Maurizio Lenzerini
Universitá di Roma La Sapienza

Nicole Schweikardt
Goethe-Universität Frankfurt am Main

ACM Classification 1998
H.2.5 Heterogeneous Databases, H.2.4 Systems, H.2.8 Database Applications, F.2.2 Nonnumerical Algorithms and Problems

ISBN 978-3-939897-61-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-61-3.

Publication date
October, 2013

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/DFU.Vol5.10452.i

DFU – Dagstuhl Follow-Ups

The series *Dagstuhl Follow-Ups* is a publication format which offers a frame for the publication of peer-reviewed papers based on Dagstuhl Seminars. DFU volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8977

www.dagstuhl.de/dfu

Contents

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt
DAGSTUHL FOLLOW-UPS Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

Preface

This volume is based on GI-Dagstuhl Seminar 10452 on “Data Exchange, Integration, and Streams” (DEIS’10) held in November 2010. Before discussing the volume itself, we present some background and an overview of the DEIS’10 event, which we co-organized.

Background

The Schloss Dagstuhl – Leibniz Center for Informatics or, simply, Dagstuhl is known as the place “where computer scientists meet”. Many computer scientists are familiar with the Dagstuhl seminars in which participants spend a week interacting with colleagues in an informal setting by sharing new results and work in progress, exchanging ideas, or embarking on new collaborations. Alongside these year-round seminars, however, Dagstuhl also hosts a different and less frequent type of event that is expressly geared towards students and postdoctoral scholars. Specifically, Dagstuhl is also the home of the GI-Dagstuhl Seminars[1], which are sponsored jointly by the German Informatics Society (GI) and the Schloss Dagstuhl – Leibniz Center for Informatics. The designated purpose of GI-Dagstuhl Seminars is to enable young researchers to learn about new developments in a particular area of research through active engagement in the seminar, which is typically organized by an international team of senior researchers. GI-Dagstuhl Seminars are typically limited to at most 20–25 participants, including the organizers.

In November of 2009, we submitted a proposal for a GI-Dagstuhl Seminar in the form of an advanced school on data exchange, data integration, and data streams. These are three different, yet inter-related, facets of information integration that have been investigated in depth by the research community in recent years.

Data exchange and data integration deal with the execution of information integration, but they adopt distinctly different approaches. Data exchange is the problem of transforming data residing in different sources into data structured under a target schema; in particular, data exchange entails the materialization of data, after the data have been extracted from the sources and re-structured into the unified format. In contrast, data integration can be described as symbolic or virtual integration: users are provided with the capability to pose queries and obtain answers via the unified format interface, while the data remain in the sources and no materialization of the restructured data is required.

In the basic data stream model, the input data consists of one or several streams of data items that can be read only sequentially, one after the other. This scenario is relevant for a large number of applications where massive amounts of data need to be processed. Typically, algorithms have to work with one or few passes over the data and a memory buffer of size significantly smaller than the input size.

Overview of the DEIS’10 Event

After our proposal was accepted, we disseminated the plan for the Advanced School on Data Exchange, Integration, and Streams (DEIS’10) via postings to a number of forums, including DBWorld, and through a dedicated web page at
`http://www.tks.cs.uni-frankfurt.de/events/deis10`

[1] `http://www.dagstuhl.de/en/program/gi-dagstuhl-seminars/`

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt
DAGSTUHL FOLLOW-UPS Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

Potential applicants were asked to submit by July 15, 2010 an application consisting of a letter of interest, a curriculum vitae, up to three representative papers or theses authored by the applicant, and a letter of recommendation from an academic supervisor or other senior colleague. We received 31 applications, out of which 22 applicants were selected to participate in DEIS'10; together with the organizers, this brought the total number of DEIS'10 participants to 25, which is the maximum that can be accommodated in a GI-Dagstuhl Seminar. The great majority of the applications received were of very high quality. In fact, we would have gladly accepted more applicants had there been more room. Of the 22 successful applicants, 18 were graduate students and 4 were postdoctoral scholars. In terms of geography, 18 were located in Europe, 3 in North America, and 1 in South America.

The participants were notified of their selection in early September 2010. Each participant was asked to study the relevant literature in a specialized topic that was assigned to him or her by the organizers of DEIS'10, based on the interests and expertise of the participants. Moreover, each participant was assigned one of the three organizers as mentor. Mentors and mentees interacted via email during September and October 2010. In particular, participants were asked to send their mentors a progress report with an outline of their presentation by the beginning of October 2010, which was followed by a semi-final draft of the slides of their presentation a week before DEIS'10 took place.

During the first day of DEIS'10, each of the three organizers gave a 90-minute tutorial on one of the three main themes of the school. Specifically, there was a tutorial on "Schema Mappings and Data Exchange" by Phokion Kolaitis, a tutorial on "Data Integration" by Maurizio Lenzerini, and a tutorial on "Data Streams" by Nicole Schweikardt. The rest of the program consisted of the presentations by the participants. Each participant was given 45 minutes to present an overview of the specialized topic assigned to her or him; the presentations were followed by or were interspersed with questions by the audience, so that a total of one hour was allotted to each specialized topic. The specialized topics covered during DEIS'10 were as follows.

Data Exchange: "The chase procedure and its applications to data exchange" by Andrian Onet; "Algorithms for computing the core of universal solutions" by Vadim Savenkov; "The inverse operator on schema mappings and its uses in data exchange" by Jorge Pérez; "Integrity constraints in data exchange" by Víctor Guttiérez-Basulto; "Semantics of query answering in data exchange and closed world reasoning" by André Hernich; "Analyzing, comparing and debugging schema mappings" by Emanuel Salinger; and "XML data exchange" by Amélie Gheerbrant.

Data Integration: "Query answering in data integration" by Piotr Wieczorek; "Data integration: consistent query answering" by Sławomir Staworko; "Data cleaning for data integration" by Ekaterini Ioannou; "Description logics for data integration" by Y. Angélica Ibáñez-Garcia; "View-based query processing" by Paolo Guagliardo; "Probabilistic data integration and probabilistic data exchange" by Livia Predoiu; "Learning and discovering queries and mappings" by Marie Jacob; "Theory of peer data management" by Sebastian Skritek; "Peer data management systems" by Armin Roth; and "XML data integration" by Lucja Kot.

Data Streams: "Basic algorithmic techniques for processing data streams" by Mariano Zelke; "Data stream management systems and query languages" by Sandra Geisler; "Querying and mining data streams" by Elena Ikonomovska; "Distributed processing of data streams and large data sets" by Marwan Hassani; and "Stream-based processing of XML documents" by Cristian Riveros.

While a small number of participants presented some of their own research work, most of the presentations were a synthesis of papers studied by the participants in the months before DEIS'10 took place. In total, well over 100 published papers were distilled and synthesized by the participants in their presentations. The slides of these presentations and the relevant bibliographical references can be found at the web page of DEIS'10.

In addition to the tutorials and the presentations of specialized topics, an after-dinner problem session was held in the second day of DEIS'10. In this session, both the organizers and the participants presented selected open problems in each of the three main themes of DEIS'10. The last time slot of DEIS'10 was a wrap-up session during which feedback about the event was solicited and tentative plans for a follow-up event were discussed.

Follow-Up

For some of the topics presented at DEIS'10, excellent survey articles already exist. Some other topics are still too nascent to justify survey articles at this point of time. For several more mature topics for which no survey articles presently exist, we felt that the time is ripe to produce such survey articles as a follow-up to DEIS'10. To this effect, we invited a number of DEIS'10 participants to contribute chapters to this volume. In several cases, we paired authors and asked them to co-author chapters that constitute a synthesis of their individual presentations at DEIS'10. Each draft chapter was peer-reviewed and subsequently revised to take into account the suggestions of the reviewers.

Overview of the Volume

The first four chapters in this volume examine several different, yet inter-related, aspects of data exchange. The underlying thread in these chapters is the systematic use of schema mappings, which are are high-level syntactic specifications that describe the relationship between two database schemas. Schema mappings have turned out to be the essential building blocks in formalizing and analyzing data inter-operability tasks, such as data exchange and data integration.

The first chapter of this volume, which is authored by Adrian Onet, gives a comprehensive overview of the properties of the chase procedure, an important algorithm that has been widely used to construct "good" solutions in data exchange and also to reason about schema mappings. The study of "good" solutions in data exchange is pursued in more depth in the second chapter, which is authored by Vadim Savenkov. Here, the focus is on the algorithmic properties of core universal solutions, which, intuitively, are the "best" solutions to materialize in data exchange. The third chapter, which is authored by Jorge Pérez, examines the various approaches that have been taken towards giving precise semantics and studying the properties of the inverse operator on schema mappings, an operator that, as the name suggests, is intended to "reverse" the action of the given schema mapping. The fourth chapter, authored by Emanuel Sallinger, presents an overview of the concepts introduced and the methods developed to reason about schema mappings with emphasis on optimality and equivalence between schema mappings.

The next four chapters explore different aspects of data integration. All chapters are centered around what is considered the main problem in this form of information integration, namely processing queries posed to the data integration system.

The first one, which is the fifth chapter of this volume, authored by Paolo Guagliardo and Piotr Wieczorek, provides an overview of the techniques for computing the answers to queries posed to the global schema of a virtual data integration system, both in the case where the global schema is expressed in the relational model, and in the case where a semi-structured data model is used instead. While most papers on query processing in data integration concentrate on positive queries, the sixth chapter of this volume, authored by André Hernich, addresses the issue of selecting the right semantics and the right algorithms for answering non-monotone queries, both in data integration and in data exchange. The seventh paper, which is authored by Armin Roth and Sebastian Skritek, deals with a sophisticated form of information integration that is receiving great attention in the last years, namely peer data integration. Unlike traditional data integration systems, peer data integration systems do not rely on a unique global schema. Instead, they allow for full autonomy of a set of data sources, with mappings between them, and no need of central coordinator. The eighth chapter, authored by Ekaterini Ioannou and Sławek Staworko, provides a discussion on techniques introduced for handling inconsistencies. Query processing in data integration is often studied under the assumption that the data integration system is logically consistent. However, this is an unrealistic assumption in many real world contexts. The chapter illustrates two main approaches to deal with inconsistencies, based on "on-line" consistent query answering, and methods for resolving the inconsistencies "off-line", respectively.

The last two chapters deal with data streams. Both chapters are centered around the question of how to efficiently process massive amounts of data in a real-time manner, using memory buffers that are significantly smaller than the input data.

The first one, which is the ninth chapter of this volume, authored by Elena Ikonomovska and Mariano Zelke, gives an overview of algorithmic techniques for data stream processing. It presents abstract models for data stream processing and contains a tutorial on fundamental techniques for sampling and sketching data, as well as a survey of algorithmic approaches for similarity mining, group testing, clustering, and summarizing data streams. The tenth chapter, authored by Sandra Geisler, gives an overview of data stream management systems (DSMS), i.e., database management systems specifically designed for processing data streams. It gives details on the architecture of DSMS, surveys existing systems and query languages, and discusses methods for monitoring the data quality of DSMS.

Acknowledgments

We are grateful to the Schloss Dagstuhl – Leibniz Center for Informatics and to the German Informatics Society (GI) for giving us the opportunity to organize DEIS'10. We also wish to acknowledge the German Research Foundation (DFG) and the ACM Special Interest Group on Management Of Data (SIGMOD) for their generous financial support, which made it possible to provide travel support to participants of DEIS'10. We are particularly grateful to Dr. Marc Herbstritt, Member of the Scientific Staff of the Schloss Dagstuhl – Leibniz Center for Informatics, for his help and encouragement throughout the process of organizing DEIS'10 and preparing this volume. Finally, we wish to thank the participants of DEIS'10, the authors of the chapters in this volume, and the anonymous reviewers of the chapters. This volume would not have been possible without the contributions of all these colleagues.

June 2013 *Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt*

The Chase Procedure and its Applications in Data Exchange

Adrian Onet

Concordia University
Montreal, Canada
adrian_onet@yahoo.com

Abstract

The initial and basic role of the chase procedure was to test logical implication between sets of dependencies in order to determine equivalence of database instances known to satisfy a given set of dependencies and to determine query equivalence under database constrains. Recently the chase procedure has experienced a revival due to its application in data exchange. In this chapter we review the chase algorithm and its properties as well as its application in data exchange.

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

Keywords and phrases chase, chase termination, data exchange, incomplete information

Digital Object Identifier 10.4230/DFU.Vol5.10452.1

1 Introduction

The main focus of this chapter is an introduction to the chase procedure and its importance in data exchange, as it is already announced in the title. A retrospective look at the evolution of the chase procedure proves with no doubt its importance and effectiveness in solving several data related problems. Originally, the chase was developed for testing logical implication between sets of embedded dependencies [36]. In fact, the logical implication problem tests whether all databases satisfying a set of dependencies must also satisfy another given dependency. Later, the chase was reformulated for other types of dependencies such as functional, join and multivalued dependencies [37, 49]. Beeri and Vardi [10] proposed a unified treatment for the implication problem by introducing the chase for tuple-generating and equality-generating dependencies, classes of dependencies large enough to express all the previous classes. Moreover, the chase procedure was also shown to be useful for determining if two database instances (that may contain nulls) represent the same set of possible instances under a set of dependencies [43]. Finally, the chase was also used for testing query equivalence and containment under database constraints [3, 31].

More recently, the chase procedure has gained a lot of attention due to its usefulness in: data integration [33, 11], ontologies [14, 13], inconsistent databases and data repairs [5, 2, 23], data exchange [19], query optimization [17, 42], peer data exchange [9], and data correspondence [23]. In this chapter we will focus on the advantages of using the chase procedure in data exchange. We will show that the chase can be used to compute representative target solutions in data exchange. Intuitively, the data exchange problem consists of transforming a source database into a target one according to a set of source to target dependencies describing the mapping between the source and the target. The set of dependencies may also include target dependencies, that is constraints over the target database. It is important to mention that the source and the target schemata are considered to be distinct. To be more precise: given a source instance I and a set Σ of source-to-target and target dependencies, an instance J over the target schema is said to be a data-exchange solution (or simply solution) for I and Σ, if $I \cup J$ satisfies all dependencies in Σ. One of the most important representation for this (usually infinite) set of solutions was introduced by Fagin et al. [19]. They considered the finite instance obtained by chasing the initial source instance with the set of dependencies. Such an instance, if it exists, was called a *universal solution.* In their paper, Fagin et al. showed that the universal solution is a good candidate to be materialized on the target. In particular, the universal solution can be used to compute certain answers to (unions of) conjunctive queries over the target instance.

Even though not exhaustive, this chapter investigates the chase procedure when applied against a set of tuple-generating and equality generating dependencies. We also investigate different chase variations proposed for data exchange and review their properties. Section 3 is devoted to the chase procedure and to some of its variations when the constraints are specified by sets of tuple-generating and equality generating dependencies. This section it is not only focused on mapping constraints (i.e. source-to-target and target dependencies) but also general constraints giving us a full picture for the chase termination problem. In the same Section 3, we will also see that each of the chase variation presented computes instances that are homomorphically equivalent and thus any of this chase variations can be used in computing universal solutions. Even more, to the best of our knowledge, there exists only one chase variation which is complete in finding universal solutions. It is known that the chase procedure might not terminate for some input instances; even more, it was shown that it is undecidable to test if a chase procedure terminates for a given set of dependencies and a given input instance. Given this, there was tremendous work in finding classes of dependencies that ensure the chase termination for all input instances. Section 4 presents some of the main such classes and reviews the subset relationship and complexity of testing the membership problem for these classes. In Section 5 we review the role played by the chase procedure in data exchange. Finally, Section 6 presents an extension of the chase process that deals with larger classes of dependencies including inequalities, disjunctions and negations. All the proofs presented in this chapter are only sketches, the complete proofs can be found in the mentioned literature.

2 Preliminaries

For basic definitions and concepts we refer to [1]. We will consider the complexity classes PTIME, NP, coNP, DP, RE, coRE, and first few levels of the polynomial hierarchy. For definitions of these classes we refer to [45].

Let us start with some preliminary notions. A *schema* $\mathbf{R}$ is a finite set $\{R_1, \ldots, R_n\}$ of relation names, each R_i having a fixed arity, $arity(R_i)$. Let Const be a countably infinite set of constants, Null be a countably infinite set of labeled nulls and Var be a countable infinite set of variables, such that the sets are pairwise disjoint. From the domain $\mathsf{Dom} = \mathsf{Const} \cup \mathsf{Null}$ and the finite set $\mathbf{R}$ we build up a Herbrand structure consisting of all expressions of the form $R(a_1, a_2, \ldots, a_k)$, where R is a k-ary relation name from $\mathbf{R}$ and a_i's are values in Dom. Such an expression is called a tuple. A database instance I is then simply a finite set of tuples. We denote the set of values occurring in an instance I by $dom(I)$. An instance I, such that $dom(I) \subseteq \mathsf{Const}$, is called a *ground instance.*

Let then I and J be instances over a schema $\mathbf{R}$. A *homomorphism* h from I to J is a function on $\mathsf{Const} \cup \mathsf{Null}$, that is identity on Const, extended to tuples, relations and instances in the natural way, such that $h(I) \subseteq J$. We write $I \rightarrow J$ in case there exists a homomorphism from I to J. By $I \leftrightarrow J$ we denote the fact that $I \rightarrow J$ and $J \rightarrow I$. A homomorphism from I

to J is said to be *full* if $h(I) = J$. A full injective homomorphism is called *embedding*. A homomorphism h from I to J is said to be a *retraction* if h is identity on $dom(J)$. In this case J is called a *retract* of I. An instance J is said to be a proper *retract* of instance I, if J is a retract of I and $J \subset I$. An instance I is said to be a core if it does not have any proper retract. An instance J is said to be a core of I if J is a retract of I and it is also a core. The cores of an instance I are unique up to isomorphism and therefore we can talk about *the core* of an instance I and denote it $core(I)$.

A relational atom is an expression of the form $R(\bar{x})$, where R is a relational symbol from a schema $\mathbf{R}$ and $\bar{x} \in (\mathsf{Const} \cup \mathsf{Var})^{arity(R)}$. For an easier representation, by $\bar{x}$ we also represent the sets of elements in $\bar{x}$ and we denote by $|\bar{x}|$ the cardinality of such set. A conjunctive query $\varphi(\bar{x})$ over a schema $\mathbf{R}$ is a conjunction of relational atoms from $\mathbf{R}$, where $\bar{x}$ denotes the variables of the atoms in φ. CQ identifies the class of conjunctive queries and UCQ the class of unions of conjunctive queries. The class $\mathsf{CQ}^{\neq}$ denotes all conjunctive queries that also allow the inequality atom (similarly is defined the class $\mathsf{UCQ}^{\neq}$). The extension of previous classes by allowing negation gives $\mathsf{CQ}^{\neg}$,$\mathsf{UCQ}^{\neg}$, $\mathsf{CQ}^{\neg,\neq}$ and $\mathsf{UCQ}^{\neg,\neq}$. Given a formula $\alpha : R_1(\bar{x}_1) \wedge R_2(\bar{x}_2) \wedge \ldots \wedge R_n(\bar{x}_n)$ and a mapping $h : (\mathsf{Const} \cup \mathsf{Var}) \to (\mathsf{Const} \cup \mathsf{Null})$, identity on Const, by $h(\alpha)$ we denote the instance $\{R_1(h(\bar{x}_1)), R_2(h(\bar{x}_2)), \ldots, R_n(h(\bar{x}_n))\}$.

Finally, a *tuple generating dependency* (tgd) is a first order sentence ξ of the form: $\forall \bar{x}, \bar{y}\ \big(\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})\big)$, where α (the body) and β (the head) are conjunctive queries, $\bar{x}$ and $\bar{y}$ denote the universally quantified variables, and $\bar{z}$ the existentially quantified ones. We denote by $body(\xi)$ the set of all atoms in the body and by $head(\xi)$ the set of all atoms in the head. An *equality generating dependency* (egd) is a first order sentence ξ of the form $\forall \bar{x}\ \big(\alpha(\bar{x}) \to x_1 = x_2\big)$. An egd is like a tgd, except that the consequent is an equality between the variables x_1 and x_2 that also are part of $\bar{x}$. For simplicity for these types of formulae, we will omit the universal quantifiers; also the conjunction between atoms will be denoted by comma. Thus, the tgd $\forall x, y\ \big(R(x, y) \wedge R(y, x) \to \exists z\ T(x, z) \wedge S(z)\big)$ will be simply denoted as $R(x, y), R(y, x) \to \exists z\ T(x, z), S(z)$. A *full tgd* is a tgd that has no existentially quantified variables. A LAV tgd is a tgd with only one atom in the body. Let ξ be a tgd of the form $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, and $\bar{a} \in (\mathsf{Const})^{|\bar{x}|+|\bar{y}|}$. We denote by $\xi(\bar{a})$ the first order sentence obtained from ξ by replacing the universal quantified variables with the corresponding constants in $\bar{a}$. An instance I is said to *satisfy* a set of dependencies, denoted $I \models \Sigma$, if I satisfies Σ in the theoretic standard model sense.

3 The chase procedure

The importance of the chase procedure in data exchange was first brought to the forefront by Fagin, Kolaitis, Miller and Popa in their seminal paper [19], where the chase was used as a tool for constructing a "general" solution to the data-exchange problem. In their approach the chase procedure applies at each iteration a chase step, that either adds a new tuple or changes the instance to model some equality generating dependency, or fails when the instance could not be changed to satisfy an equality generating dependency. Based on this chase procedure, several variations were proposed [12, 16, 38, 24]. To differentiate them, we will call the chase procedure presented by Fagin et al. [19] *the standard chase*. Since most of the practical database constraints (such as key and inclusion dependencies) can be represented as sets of tuple generating (tgd) and equality generating (egd) dependencies, in this section we will present the chase procedure applied on such dependencies. Later on, more precisely in Section 6, we will introduce a variation of the chase procedure that also deals with dependencies containing inequalities and disjunctions. For an ease of notation, through this section if not mentioned otherwise, we will use the notation I to represent an arbitrary instance over a given schema and Σ to refer to an arbitrary set of tgds and egds over a schema explicitly mentioned if it does not follow directly from the context.

3.1 The chase step

The chase procedure is a repetitive application of a chase step. Each chase step "applies" a tgd or egd, on a subset of the instance.

The tgd chase step. Let I be an instance and ξ be the tgd $\alpha(\bar{x},\bar{y}) \to \exists\bar{z}\ \beta(\bar{x},\bar{z})$ both over a schema $\mathbf{R}$. A pair (ξ,h) is said to be a *trigger* for I, if h is a homomorphism such that $h(\alpha(\bar{x},\bar{y})) \subseteq I$. In case we also have that there is no extension $\tilde{h}$ of h such that $\tilde{h}(\beta(\bar{x},\bar{z})) \subseteq I$, then (ξ,h) is said to be an *active trigger* for I. The tgd ξ is said to be *applicable* to I with homomorphism h if (ξ,h) is a trigger (active or not) for I.

To *fire* the trigger (ξ,h) means to transform I into the instance $J = I \cup \tilde{h}(\beta(\bar{x},\bar{z}))$, where $\tilde{h}$ is a *distinct extension* of h, i.e. an extension of h that assigns new fresh nulls to the existential variables in β. By "new fresh" we mean the next unused element in some fixed enumeration of the nulls. We call this transformation as an *oblivious-chase step* and denote it $I \xrightarrow{*,(\xi,h)} J$. In case (ξ,h) is an active trigger for I, the transformation is called *standard-chase step* and is denoted $I \xrightarrow{(\xi,h)} J$. Clearly any standard-chase step is also an oblivious-chase step but the converse does not always hold.

▶ **Example 1.** Let us consider instance $I = \{R(a,b), R(b,a), S(b,c)\}$ and tgd ξ:

$$R(x,y), R(y,x) \to \exists z\ S(x,z).$$

Homomorphism $h = \{x/a, y/b\}$ maps the body of ξ to I and there is no extension of h that maps the head of ξ into I. That is, the trigger (ξ,h) is active for I and $I \xrightarrow{(\xi,h)} J$, where $J = I \cup \{S(a,X)\}$ and $\tilde{h}(z) = X$. On the other hand, for the homomorphism $h' = \{x/b, y/a\}$ the pair (ξ,h') is a trigger for I, but it is not an active trigger. In this case $I \xrightarrow{*,(\xi,h')} J'$, where $J' = I \cup \{S(b,X)\}$.

The complexity of testing if there exists a trigger (active trigger) for a given instance I and a fixed (or given) tgd ξ is given by the following theorem:

▶ **Theorem 2.** [26] *Let ξ be a tgd and I an instance. Then*

1. *for a fixed ξ, testing whether there exists a trigger or an active trigger on a given I is polynomial;*
2. *testing whether there exists a trigger for a given ξ on a given I is* NP*-complete;*
3. *testing whether there exists an active trigger for a given ξ and a given I is Σ_2^p-complete.*

Proof. Let us consider ξ to be a tgd of the form $\alpha(\bar{x},\bar{y}) \to \exists\bar{z}\ \beta(\bar{z})$. The polynomial cases can be verified by checking all homomorphisms from the body of the dependency into the instance. We also need to consider for the active trigger problem if it has for each such homomorphism an extension that maps the head of the dependency into the instance. These tasks can be carried out in $O(n^{|\alpha|})$ and $O(n^{|\alpha|+|\beta|})$ time, respectively.

It is easy to see that the trigger existence problem is NP-complete in combined complexity, as the problem is equivalent to testing whether there exists a homomorphism between two instances (in our case α and I); a problem known to be NP-complete.

Regarding the combined complexity of the active trigger existence problem, we observe that it is in Σ_2^P, since one may guess a homomorphism h from α into I, and then use an NP oracle to verify that there is no extension h' of h, such that $h'(\beta) \subseteq I$. In the case of the lower bound, we will reduce the following problem to the active trigger existence problem. Let $\phi(\bar{x}, \bar{y})$ be a Boolean formula in 3CNF over the variables in $\bar{x}$ and $\bar{y}$. *Is the formula* $\exists \bar{x}\, \neg\big(\exists \bar{y}\, \phi(\bar{x}, \bar{y})\big)$ *true?* The problem is a variation of the standard $\exists\forall$-QBF problem [48] and known to be Σ_2^P-complete [46].

For the reduction, let ϕ be given. We construct an instance I_ϕ and a tgd ξ_ϕ. The instance I_ϕ is:

F		
1	0	0
0	1	0
0	0	1
1	1	0
1	0	1
0	1	1
1	1	1

N	
0	1
1	0

The tgd $\xi_\phi = \alpha \to \beta$ is constructed as follows: for each variable $x \in \bar{x}$ in $\phi(\bar{x}, \bar{y})$, the body α will contain the atom $N(x, x')$ (x' is used to represent $\neg x$). The head β is existentially quantified over that set $\bigcup_{y \in \bar{y}} \{y, y'\}$ of variables. For each conjunct C of ϕ, we place an atom $F(x, y, z)$ in β, where x, y and z are the variables in C, with the convention that if the variable x is negated in C, then x' is used in the atom. Finally, for each $y \in \bar{y}$, we place in β the atom $N(y, y')$, denoting that y and y' should not have the same truth assignment. It is easy to note that $\exists \bar{x}\, \neg\big(\exists \bar{y}\, \phi(\bar{x}, \bar{y})\big)$ is true, if and only if there exists an active trigger (ξ_ϕ, h) for I_ϕ. ◀

The egd chase step. Let I be an instance and ξ the egd $\alpha(\bar{x}) \to x_i = x_j$, where $x_i, x_j \in \bar{x}$. We say that ξ is *applicable* to I with the homomorphism h, if the following holds:

1. h maps the atoms of $\alpha(\bar{x})$ to tuples of I,
2. $h(x_i) \neq h(x_j)$.

The pair (ξ, h) is called an egd active trigger for I, or simply a trigger. Let (ξ, h) be an egd trigger for I. In case h maps both variables x_i and x_j to constants, then we say that the egd chase step *fails*, and represent this by $I \xrightarrow{(\xi,h)} \perp$. Otherwise, we say that the egd chase step *does not fail* and denote this by $I \xrightarrow{(\xi,h)} J$, where instance J is computed as follows:

1. if both $h(x_i)$ and $h(x_j)$ are labeled nulls, then J is obtained from I by replacing all occurrences of $h(x_i)$ with $h(x_j)$, considering that there is an enumeration of the variables such that $i < j$.
2. if either $h(x_i)$ or $h(x_j)$ is a constant and the other is a labeled null, then J is obtained from I by replacing all occurrences of the labeled null with the constant.

▶ **Example 3.** Consider instance $I = \{R(a, b), R(c, X), R(X, Y)\}$ and ξ: $R(x, y) \to x = y$. There are three distinct homomorphisms that map the body of ξ into I: $h_1 = \{x/a, y/b\}$, $h_2 = \{x/c, y/X\}$ and $h_3 = \{x/X, y/Y\}$. As $h_1(x)$ and $h_1(y)$ are distinct constants, it follows that $I \xrightarrow{(\xi,h_1)} \perp$. On the other hand, $h_2(x)$ is a constant and $h_2(y)$ is a null. Thus, we have $I \xrightarrow{(\xi,h_2)} J$, where $J = \{R(a, b), R(c, c), R(c, Y)\}$ is obtained by replacing all occurrences of null X with constant c in I. Finally, h_3 maps both variables x and y to distinct nulls, making the egd ξ applicable on I with the homomorphism h_3. Hence $I \xrightarrow{(\xi,h_3)} J'$, where $J' = \{R(a,b), R(c,Y), R(Y,Y)\}$ is obtained from I by replacing X with Y, considering that y follows x in the variable enumeration.

Similarly to the tgd chase step, we have the following complexity results for the egd trigger-existence problem. Note that for egds the trigger and active trigger notions coincide.

▶ **Theorem 4.** *Let ξ be an egd and I an instance. Then*
1. *for a fixed ξ, testing whether there exists a trigger on a given I is polynomial, and*
2. *testing whether there exists a trigger for a given ξ on a given I is* NP*-complete.*

Proof. Similar with the proof of Theorem 2. ◀

3.2 The chase algorithm

Using the previously introduced chase steps, we are now ready to present the standard-chase algorithm. This algorithm can be described as an iterative application of the standard-chase steps. In case one of the egd chase steps fails, then the algorithm will fail. If the algorithm does not fail, it nondeterministically chooses another active trigger, tgd or egd, and proceeds with the corresponding standard-chase step. The algorithm terminates when either one of the egd chase step fails or when there are no other active triggers. More formally, the standard-chase algorithm can be described as follows:

```
STANDARD-CHASE(I,Σ)
1 $I_0 := I$; $i := 0$;
2 if exists active trigger $(\xi, h)$ for $I_i$
3     then
4         if $I_i \xrightarrow{(\xi,h)} \perp$
5           then return FAIL
6           else $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i := i + 1$
7     else return $I_i$
8 goto 2
```

Note that the previous algorithm introduces a nondeterministic step at line 2, induced by the trigger choice. This makes the chase process to be viewed as a tree, where level i in the tree represents the i-th step in the chase algorithm, and where to each node a new edge is added for each of the applicable active trigger. Each path from the root of the tree to a leaf node represents an *execution branch*, or simply a branch, similarly to the nondeterministic finite automata. Thus the algorithm may return different instances depending on the considered branch. There are cases when, for some branches, the algorithm fails while it does not fail for other branches, as it is shown in Example 6. This happens by exhaustively choosing the same dependencies in the nondeterministic step.

Moreover, the standard-chase algorithm stops if it either fails, due to an egd trigger at step 4, or there are no other active triggers to be applied. As the tgds are adding new tuples to the instance, it may be that the chase algorithm never terminates as in Example 5.

Fagin et al. [19] showed that in case the standard-chase algorithm fails on one execution branch, then it will fail on all finite branches.

For the branches for which the algorithm does not fail, a *standard-chase sequence* is a finite or infinite sequence $(I_0, I_1, I_2, \ldots, I_n, \ldots)$ such that $I_0 = I$ and $I_i \xrightarrow{(\xi,h)} I_{i+1}$, for some $i \geq 0$ and some active trigger (ξ, h). If for some branch the algorithm terminates in the finite, then there exists a positive integer n such that for the standard-chase sequence $(I_0, I_1, I_2, \dots, I_n)$ there is no active trigger for I_n.

As shown in the following example, a standard chase sequence may be finite or infinite, for the same set of tgds and the for same input instance.

▶ **Example 5.** Consider instance $I = \{R(a, b)\}$ and tgds:

$$\begin{aligned} \xi_1 &= R(x, y) \rightarrow R(y, x), \text{ and} \\ \xi_2 &= R(x, y) \rightarrow \exists z\ R(y, z). \end{aligned}$$

If first we chose the tgd trigger $(\xi_1, \{x/a, y/b\})$, the tuple $R(b, a)$ is added to the instance I forming an instance $I' = I \cup \{R(b, a)\}$. It can be easily noticed that there is no other active trigger on I' involving either ξ_1 or ξ_2. From this it follows that the sequence (I, I') is a finite standard-chase sequence. On the other hand, if in the standard-chase algorithm we chose first the active trigger $(\xi_2, \{x/a, y/b\})$, and from there on only chose active triggers over ξ_2, we get the following infinite chase sequence:

$$I_0 = I \xrightarrow{(\xi_2,h_1)} I_1 \xrightarrow{(\xi_2,h_2)} \dots \xrightarrow{(\xi_2,h_n)} I_n \xrightarrow{(\xi_2,h_{n+1})} \dots$$

$I_0 = I$:

R	
a	b

I_1:

R	
a	b
b	X_1

I_n:

R	
a	b
b	X_1
X_1	X_2
$\dots$	
X_{n-1}	X_n

The next example shows a case when the standard-chase algorithm fails on some branches and does not terminate (implicitly does not fail) on others.

▶ **Example 6.** Let us now consider a slightly changed set of dependencies from the previous example:

$$\begin{aligned} \xi_1 &= R(x, y) \rightarrow T(y, x); \\ \xi_2 &= T(x, y) \rightarrow x = y; \text{ and} \\ \xi_3 &= R(x, y) \rightarrow \exists z\ R(y, z). \end{aligned}$$

Consider the instance $I = \{R(a, b)\}$. If applying an active trigger $(\xi_1, \{x/a, y/b\})$, it will add the tuple $T(a, b)$ to I. Next, when applying the active trigger $(\xi_2, \{x/a, y/b\})$ the standard-chase algorithm will fail. However, if the chosen branch uses only the triggers over ξ_3, the standard-chase algorithm will not terminate, as previously shown.

In the previous example the standard-chase algorithm did not fail because we exhaustively applied active triggers over the same dependency. To avoid such cases, the chase algorithm is required to be *fair*, defined as follows:

▶ **Definition 7.** Let I_0 be an instance and Σ a set of tgds and egds. A standard-chase sequence $(I_0, I_1, \dots)$ is said to be *fair* if for all i and for all active triggers (ξ, h) for I_i, where $\xi \in \Sigma$, there exists j such that either $I_j \xrightarrow{(\xi,h)} I_{j+1}$ or the trigger (ξ, h) is not active for I_j. A standard-chase algorithm is said to be *fair*, if it only produces fair chase sequences.

In the rest of this chapter we will consider, if not mentioned otherwise, all the chase algorithms to be fair.

Let us now turn our attention to standard-chase algorithms that terminate in a finite number of steps. The following proposition shows the relationship between the finite instances returned by the algorithm.

▶ Proposition 8. [19] If K and J are two finite instances returned by the standard-chase algorithm on two distinct execution branches, with input I and Σ, then K and J are homomorphically equivalent, that is $K \leftrightarrow J$.

Based on the homomorphic equivalence class, if there exists an execution branch for which the standard-chase algorithm with input I and Σ terminates in the finite and does not fail, then we denote by $chase^{\textsf{std}}{}_{\Sigma}(I)$ one representative of the equivalence class for the resulting finite instances. If the standard chase fails or if it does not terminate in the finite on all branches, then we set $chase^{\textsf{std}}{}_{\Sigma}(I) = \perp$. With the instance I and the dependencies Σ defined in Example 5, we have $chase^{\textsf{std}}{}_{\Sigma}(I) = \{R(a, b), R(b, a)\}$. On the other hand, with the input instance and the dependencies defined in Example 6 we have $chase^{\textsf{std}}{}_{\Sigma}(I) = \perp$. The following theorem, developed by Fagin et. al, states one of the main properties of the standard-chase algorithm. As seen later in this section, this property also holds for the other chase variations.

▶ **Theorem 9.** [19] *If $chase^{\textsf{std}}{}_{\Sigma}(I) \neq \perp$, then $chase^{\textsf{std}}{}_{\Sigma}(I) \models \Sigma$ and $I \to chase^{\textsf{std}}{}_{\Sigma}(I)$.*

Let us now turn our attention to the problem defined as the termination of standard-chase algorithm. It is easy to see that the cause of non-termination lies in the existentially quantified variables in the head of tgds. Thus, for simplicity, for the following classes we omitted egds.

▶ **Definition 10.** Given an instance I, by $\mathsf{CT}^{\textsf{std}}_{I,\forall}$ we denote the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\textsf{std}}_{I,\forall}$ iff all standard-chase sequences for I and Σ are finite. We denote by $\mathsf{CT}^{\textsf{std}}_{I,\exists}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\textsf{std}}_{I,\exists}$ iff there exist some standard-chase sequences for I and Σ that are finite.

The previous notations are extended to classes of tgd sets for which the standard chase terminates on all input instances as follows:

▶ **Definition 11.** We denote by $\mathsf{CT}^{\textsf{std}}_{\forall\forall}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\textsf{std}}_{\forall\forall}$ iff for all instances I all standard-chase sequences of I with Σ are finite. We denote by $\mathsf{CT}^{\textsf{std}}_{\forall\exists}$ the class of tgd sets such that $\Sigma \in \mathsf{CT}^{\textsf{std}}_{\forall\exists}$ iff for all instances I there exists at least one standard-chase sequence of I and Σ that is finite.

From the termination classes definition it is clear that $\mathsf{CT}^{\textsf{std}}_{\forall\forall} \subseteq \mathsf{CT}^{\textsf{std}}_{\forall\exists}$. Also from the set of dependencies presented in Example 5, it follows that the inclusion is strict.

Deutsch, Nash and Remmel in [16] showed that, given I and Σ, the problems of testing whether $\Sigma \in \mathsf{CT}^{\textsf{std}}_{I,\forall}$ or $\Sigma \in \mathsf{CT}^{\textsf{std}}_{I,\exists}$ are undecidable in general. More recently, Grahne and O. [26] extended this undecidability result to the $\mathsf{CT}^{\textsf{std}}_{\forall\exists}$ class too. That is for a given Σ, the problem of testing if $\Sigma \in \mathsf{CT}^{\textsf{std}}_{\forall\exists}$ is coRE-complete. In case we allow a single *denial constraint*, then the class $\mathsf{CT}^{\textsf{std}}_{\forall\forall}$ is coRE-complete as well. Where a denial constraint is a tgd of the form $\alpha(\bar{x}) \to \perp$, which is satisfied by an instance I only if there is no homomorphism h such that $h(\alpha(\bar{a})) \subseteq I$,

Given the previous result, the next best hope is to find large decidable classes of dependencies included in $\mathsf{CT}^{\textsf{std}}_{\forall\forall}$. One such class is the one of full tgds, that is tgds without existential quantifiers. In Section 4 we review other decidable classes of dependency sets that are known to be in $\mathsf{CT}^{\textsf{std}}_{\forall\forall}$.

Before ending this section we need to reiterate that the termination classes are defined over tgds. Even if the egds do not introduce new nulls, they still may play an important role in the standard-chase termination problem, as shown in the next example.

▶ **Example 12.** Let $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned} \xi_1 &= R(x,y) \to \exists z\ S(y,z); \\ \xi_2 &= S(x,x) \to \exists z\ R(x,z);\ \text{and} \\ \xi_3 &= S(x,y) \to x = y. \end{aligned}$$

Let $I = \{R(a,b)\}$. It is easy to see that the standard-chase algorithm converges to the infinite instance $J = \{R(a,b), R(b,X_1), \ldots, R(X_{n-1},X_n), \ldots, S(b,b), S(X_1,X_1), \ldots, S(X_n,X_n), \ldots\}$. On the other hand, $\Sigma' = \{\xi_1, \xi_2\} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, that is without the egd ξ_3 the standard chase algorithm terminates on all execution branches with any input instance.

3.3 Chase variations

After the standard chase was presented as a method of computing "general" solutions in data exchange [19], many variations of the standard-chase algorithm were proposed in the literature [12, 16, 38, 24] . In the remaining part of this section, dedicated to the chase algorithm, we try to differentiate between the main chase variations by highlighting their termination properties.

3.3.1 The oblivious chase

This focuses on one of the simplest variations of the standard chase named the oblivious chase (also known as naïve chase). This procedure is based on the relaxation of the chase step. The oblivious chase presented here differs from the one described by Cali et al. [12] by not relying on any order. As we will see, this does not affect the finite instance returned by the chase algorithm.

The oblivious-chase algorithm is an iterative application of the oblivious-chase step, that is at each iteration all triggers are considered, and not only the active ones as in the standard-chase algorithm. Recall that for the trigger (ξ, h) and the instance I the oblivious-chase step is denoted as $I \xrightarrow{*,(\xi,h)} J$, where instance J is constructed the same way as in the standard-chase step. Note that if (ξ, h) is a trigger for I, then (ξ, h) will also be a trigger for J, where $I \xrightarrow{*,(\xi,h)} J$. To avoid such infinite loops, the oblivious-chase algorithm applies each trigger only once.

▶ **Example 13.** Consider the instance $I = \{R(a,b), R(b,a), S(b,c)\}$ and the tgd ξ defined as $R(x,y), R(y,x) \to \exists z\ S(x,z)$ as in Example 1. The homomorphism $h = \{x/a, y/b\}$ maps the body of ξ to I, and there is no extension of h that maps the head of ξ into I. This makes (ξ, h) both a standard and an oblivious-chase trigger. However, the homomorphism $h_1 = \{x/b, y/a\}$ also maps the body of ξ to I, but there exists the extension $\tilde{h}_1 = \{x/b, y/a, z/c\}$ of h_1, such that $\tilde{h}_1$ maps the head of ξ into I. Hence (ξ, h_1) is a trigger but not an active trigger for I. The instance J is obtained by applying the oblivious-chase step $I \xrightarrow{*,(\xi,h_1)} J$, where $J = I \cup \{S(b,Y)\}$ and Y is a new labeled null.

Because of the nondeterministic way the oblivious-chase algorithm selects the triggers at each iteration, we may have different execution branches. Similarly to the termination classes defined for the standard chase, we introduce corresponding termination classes of tgd sets for the oblivious chase: $\mathsf{CT}^{\mathsf{obl}}_{I,\exists}$, $\mathsf{CT}^{\mathsf{obl}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and $\mathsf{CT}^{\mathsf{obl}}_{\forall\exists}$.

The following proposition shows that the termination classes are not affected by the nondeterministic nature of the algorithm.

▶ Proposition 14. Let I be an instance. Then $\mathsf{CT}^{\mathsf{obl}}_{I,\forall} = \mathsf{CT}^{\mathsf{obl}}_{I,\exists}$ and $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} = \mathsf{CT}^{\mathsf{obl}}_{\forall\exists}$.

Proof. The proof follows from the observation that for the oblivious-chase algorithm, when the input set of dependencies are only tgds, the set of triggers applied on each branch is the same, up to isomorphism. Thus, if the oblivious chase terminates on one execution branch, then it will terminate on all branches. ◀

From the previous proof it also follows that in case the oblivious chase terminates for instance I and set of tgds Σ, then the returned on all execution branches are isomorphically equivalent. As we will see in the following example, if we allow egds, the instances returned are not guaranteed to be isomorphically equivalent. Still, using the same proof techniques as the one used to prove Theorem 9, it can be shown that in this case, if the chase terminates and does not fail, the instances returned are homomorphically equivalent. Thus, if the oblivious chase terminates with input I and Σ (containing both tgds and egds), then we denote by $chase^{\mathbf{obl}}{}_{\Sigma}(I)$ one representative instance of the homomorphic equivalence class. If the oblivious chase fails or if it does not terminate, we set $chase^{\mathbf{obl}}{}_{\Sigma}(I) = \perp$.

▶ **Example 15.** Let $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned} \xi_1 &= S(x) \to \exists y\ R(x,y); \\ \xi_2 &= R(x,y) \to \exists z\ T(y,z);\ \text{and} \\ \xi_3 &= R(x,y) \to x = y. \end{aligned}$$

Let us now consider the instance $I = \{S(a)\}$. If we apply the dependencies in the order ξ_1, ξ_2, ξ_3 and then ξ_2 again, we get the following instance $J_0 = \{S(a), R(a,a), T(a,X_1), T(a,X_2)\}$. But, if we apply the dependencies in the order ξ_1, ξ_3 and finally ξ_2, the instance returned by the oblivious-chase algorithm is $J_1 = \{S(a), R(a,a), T(a,Y_1)\}$. Clearly J_0 and J_1 are not isomorphically equivalent but they are homomorphically equivalent.

From the observation that all active triggers are also triggers, it follows that:

▶ Proposition 16. $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

Proof. The inclusion follows directly from the definition of the trigger and the active trigger. For the strict inclusion part consider dependency set from Example 17. ◀

▶ **Example 17.** Consider $\Sigma = \{R(x,y) \to \exists z\ R(x,z)\}$. Clearly there is no active trigger on Σ for any instance I. On the other hand, for $I = \{R(a,b)\}$ the oblivious-chase algorithm will create the following infinite chase sequence:

$$I_0 = I \xrightarrow{*,(\xi,h_1)} I_1 \xrightarrow{*,(\xi,h_2)} \dots \xrightarrow{*,(\xi,h_n)} I_n \xrightarrow{*,(\xi,h_{n+1})} \dots$$

$$\begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ a & X_1 \end{array} \qquad \begin{array}{cc} \multicolumn{2}{c}{R} \\ \hline a & b \\ a & X_1 \\ a & X_2 \\ \multicolumn{2}{c}{\dots} \\ a & X_n \end{array}$$

In order to relate termination of the standard and the oblivious algorithms, we introduce a transformation called *enrichment* that takes a tgd $\xi = \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$ over a schema $\bar{R}$, and converts it into the tgd $\hat{\xi} = \alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z}), H(\bar{x},\bar{y})$, where H is a new relational symbol that does not appear in $\mathbf{R}$. For a set Σ of tgds defined on schema $\mathbf{R}$, the transformed set is $\widehat{\Sigma} = \{\hat{\xi} : \xi \in \Sigma\}$. Using the enrichment notion, we can present the relation between the standard and oblivious-chase terminations.

▶ **Theorem 18.** [25] *Let Σ be a set of tgds and I an instance. Then we have:*
1. $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{I,\forall}$ *if and only if* $\widehat{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$, *and*
2. $\Sigma \in \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ *if and only if* $\widehat{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

Proof. It follows from the observation that for any instance I if (ξ, h) is a trigger for I, then $(\widehat{\xi}, h)$ is also an active trigger for I. ◀

Cali et al. showed in [12] that it is undecidable if the oblivious chase terminates for a given input I and given set of tgds Σ. The same result applies for the class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ if we allow at least one denial constraint [26]. It remains an open problem if the class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ remains undecidable when considering only sets of tgds.

We close the presentation of the oblivious-chase algorithm by linking together the finite instances resulting from both chase algorithms.

▶ **Theorem 19.** [12] *Let I be an instance and let Σ be a set of tgds and egds, such that $chase^{\mathbf{obl}}{}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{obl}}{}_{\Sigma}(I)$ and $chase^{\mathbf{obl}}{}_{\Sigma}(I) \models \Sigma$.*

3.3.2 The semi-oblivious chase

The semi-oblivious-chase method was first introduced by Marnette in [38]. For this, let ξ be a tgd $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$; then the triggers (ξ, h) and (ξ, g) are considered equivalent if $h(\bar{x}) = g(\bar{x})$. The semi-oblivious chase works as the oblivious one, except that exactly one trigger from each such equivalence class is fired in a branch.

For a better differentiation between the chase algorithms presented so far consider the following example:

▶ **Example 20.** Let $\Sigma = \{\xi\}$ contain the tgd $\forall x, y\ R(x, y) \to \exists z\ T(x, z)$, and consider the instance $I = \{R(a, b), R(a, c), R(d, e), T(a, a)\}$. In this case there exist only three triggers $\tau_1 = (\xi, \{x/a, y/b\})$, $\tau_2 = (\xi, \{x/a, y/c\})$ and $\tau_3 = (\xi, \{x/d, y/e\})$. From these triggers only τ_3 is an active trigger for I. Thus, there is only one step to be executed for the standard chase: $I \xrightarrow{\tau_3} J^{\mathbf{std}}$, where $J^{\mathbf{std}} = I \cup \{T(d, X_1)\}$. Because the homomorphism from the trigger τ_1 maps x to value a as the homomorphism from the trigger τ_2, it follows that τ_1 is equivalent with τ_2. That is in the semi-oblivious chase only two triggers are applied: the active trigger τ_3 and the representative of the equivalence class for τ_1 and τ_2. Hence, the resulted instance is $J^{\mathbf{sobl}} = I \cup \{T(d, X_2)\} \cup \{T(a, X_3)\}$. Finally, the oblivious chase will apply all triggers returning instance $J^{\mathbf{obl}}$. Bellow are the tabular representations of the instances resulted by applying the standard, semi-oblivious and oblivious-chase algorithms. The instances are restricted to relation T:

$J^{\mathbf{std}}$

T	
a	a
d	X_1

$J^{\mathbf{sobl}}$

T	
a	a
d	X_2
a	X_3

$J^{\mathbf{obl}}$

T	
a	a
d	X_4
a	X_5
a	X_6

Similarly to the previous chase algorithms, we define termination classes over sets of tgds: $\mathsf{CT}^{\mathsf{sobl}}_{I,\exists}$,$\mathsf{CT}^{\mathsf{sobl}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{sobl}}_{\forall\exists}$ and $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ for the semi-oblivious algorithm. The following proposition shows that the nondeterministic behavior of the semi-oblivious-chase algorithm does not influence the termination for different execution branches.

▶ Proposition 21. Let I be an instance. Then $\mathsf{CT}^{\mathsf{sobl}}_{I,\forall} = \mathsf{CT}^{\mathsf{sobl}}_{I,\exists}$ and $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} = \mathsf{CT}^{\mathsf{sobl}}_{\forall\exists}$.

Proof. It follows from the observation that the set of representative trigger for each equivalence classes is the same for all execution branches. ◀

Similarly to the oblivious chase case, it can be shown that in case the semi-oblivious-chase algorithm terminates and not fails with the input instance I and the set of tgds Σ, then the instances returned by each execution branch are isomorphically equivalent. In case we allow egds in Σ, then the returned instances will be homomorphically equivalent. In this case we denote by $\mathit{chase}^{\mathbf{sobl}}{}_{\Sigma}(I)$ one representative instance of the homomorphic equivalence class for the instances computed on each execution branch. In case the semi-oblivious-chase algorithm fails or it is non-terminating, we set $\mathit{chase}^{\mathbf{sobl}}{}_{\Sigma}(I) = \bot$.

The same as for the oblivious chase case, we can find a rewriting of the dependencies such that we can relate the termination of the semi-oblivious-chase algorithm to the termination of the standard-chase algorithm. We introduce a transformation, called *semi-enrichment*, that takes a tgd $\xi = \alpha(\bar{x}, \bar{y}) \to \exists\bar{z}\ \beta(\bar{x}, \bar{z})$ over a schema $\mathbf{R}$, and converts it into the tgd $\tilde{\xi} = \alpha(\bar{x}, \bar{y}) \to \exists\bar{z}\ \beta(\bar{x}, \bar{z}), H(\bar{x})$, where H is a new relational symbol that does not appear in $\mathbf{R}$. For a set Σ of tgds defined on schema $\mathbf{R}$, the transformed set is $\tilde{\Sigma} = \{\tilde{\xi} : \xi \in \Sigma\}$. Using the semi-enrichment notion, the relation between the standard and semi-oblivious-chase terminations can be presented as follows.

▶ **Theorem 22.** [26] *Let Σ be a set of tgds and I an instance. Then we have:*
1. $\Sigma \in \mathsf{CT}^{\mathsf{sobl}}_{I,\forall}$ *if and only if* $\tilde{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{I,\forall}$, *and*
2. $\Sigma \in \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ *if and only if* $\tilde{\Sigma} \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.

The following theorem relates the instances returned by the semi-oblivious chase and the instances returned by the standard-chase algorithm.

▶ **Theorem 23.** [38] *Let I be and instance and let Σ be a set of tgds and egds, such that $\mathit{chase}^{\mathbf{sobl}}{}_{\Sigma}(I) \neq \bot$. Then $\mathit{chase}^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow \mathit{chase}^{\mathbf{sobl}}{}_{\Sigma}(I)$ and $\mathit{chase}^{\mathbf{sobl}}{}_{\Sigma}(I) \models \Sigma$.*

Similarly to the standard chase algorithm, the problem of testing if the semi-oblivious chase is terminating for a given instance and a given set of tgds is undecidable [38]. The same result applies for the class $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$, if we allow at least one denial constraint [26]. It remains an open problem if the class $\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$ remains undecidable when considering only sets of tgds.

The proposition below shows the set relationship between termination classes for the chase variations presented so far.

▶ Proposition 24. [26] $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$.

Proof. The inclusions are clear from the definition of the corresponding chase steps. For the first strict inclusion consider $\Sigma_1 = \{R(x, y) \to \exists z\ R(x, z)\}$. From Example 17, we know that $\Sigma_1 \notin \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$. On the other hand, it is easy to see that for any instance I only a maximum of $|I|$ triggers will be applied on each execution branch by the semi-oblivious chase. For the second strict inclusion consider $\Sigma_2 = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x) \to \exists z\ S(z), T(z, x), \text{ and} \\ \xi_2 &= S(x) \to \exists z\ R(z), T(x, z). \end{aligned}$$

It is easy to check that the standard chase terminates for Σ_2 with any input instance I. Additionally, the semi-oblivious chase does not terminate for Σ_2 and instance $I = \{R(a)\}$. ◀

3.3.3 The core chase

The class of chase algorithms is enriched by the core chase algorithm introduced by Deutsch et al. in [16]. We need to clarify from the very beginning that the core chase differs from the other variations by executing in parallel all applicable standard tgd chase steps and also computing the core of the unified instance. Note that we may only apply the standard tgd chase steps in parallel but not the egd chase steps, as the latter may modify the given instance by equating existing labeled nulls to constants or to other labeled nulls.

For a better understanding, we slightly changed the algorithm from [16] by applying all the egd triggers before applying in parallel all the active tgd triggers. This modification does not change however the result or the complexity of the given algorithm.

```
CORE-CHASE(I,Σ)
1  $I_0 := I$; $i := 0$;
2  if exists a standard egd trigger $(\xi, h)$ for $I_i$
3      then
4          if $I_i \xrightarrow{(\xi,h)} \perp$
5             then return FAIL
6             else  $I_i \xrightarrow{(\xi,h)} I_{i+1}$; $i := i+1$ goto 2
7  if exists a standard tgd trigger for $I_i$
8      then
9          For all active trigger $(\xi, h)$ for $I_i$, compute in parallel $I_i \xrightarrow{\xi,h} J_j$
10         $I_{i+1} := Core(\bigcup_j J_j)$; $i := i+1$
11     else
12         return $I_i$
13 goto 2
```

By applying all the triggers in parallel, the core-chase algorithm eliminates the non-deterministic part introduced by the standard-chase algorithm. In case the core-chase algorithm terminates in the finite and does not fail for input I and Σ, we denote the returned instance by $chase^{\mathbf{core}}{}_{\Sigma}(I)$. In case the core chase fails or it is non-terminating, we set $chase^{\mathbf{core}}{}_{\Sigma}(I) = \perp$.

Similarly to the other chase variations, for the core chase we introduce classes of tgd sets $\mathsf{CT}^{\mathsf{core}}_{I,\forall}$, $\mathsf{CT}^{\mathsf{core}}_{I,\exists}$, $\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$ and $\mathsf{CT}^{\mathsf{core}}_{\forall\exists}$. Because the core chase is deterministic, it follows that $\mathsf{CT}^{\mathsf{core}}_{I,\forall} = \mathsf{CT}^{\mathsf{core}}_{I,\exists}$ for any instance I and that $\mathsf{CT}^{\mathsf{core}}_{\forall\forall} = \mathsf{CT}^{\mathsf{core}}_{\forall\exists}$.

▶ **Theorem 25.** [16] *Let I be an instance. Then* $\mathsf{CT}^{\mathsf{std}}_{I,\exists} \subset \mathsf{CT}^{\mathsf{core}}_{I,\forall}$ *and* $\mathsf{CT}^{\mathsf{std}}_{\forall\exists} \subset \mathsf{CT}^{\mathsf{core}}_{\forall\forall}$.

Proof. For the second strict inclusion consider $\Sigma = \{R(x) \to \exists y\, R(y), S(x)\}$. Clearly the standard chase does not terminate on any branch with $I = \{R(a)\}$ and Σ, that is $\Sigma \notin \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$. On the other hand, for any instance I, the core chase will terminate in maximum $|I_R|$ steps, where I_R is instance I restricted to tuple over relation R. ◀

In [16] it is shown that the membership problem for the class $\mathsf{CT}^{\mathsf{core}}_{I,\forall}$ for a given instance I is RE-complete. To this, it was shown most recently [26] that the membership problem for the class $\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$ is coRE-complete.

It may be noted that at line 10 the core-chase algorithm does not simply compute the union between all the instances computed at line 9, but it also computes its core. This gives the following link between the core chase and standard-chase algorithms:

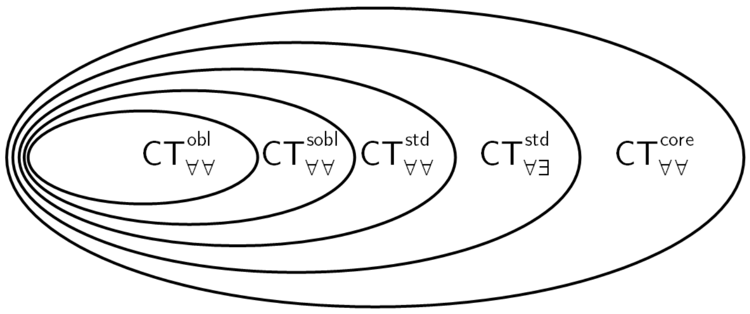

$\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{std}}_{\forall\forall}$
$\mathsf{CT}^{\mathsf{std}}_{\forall\exists}$
$\mathsf{CT}^{\mathsf{core}}_{\forall\forall}$

Figure 1 Termination classes for different chase variations.

▶ **Theorem 26.** *Let I be and instance and let Σ be a set of tgds and egds, such that $chase^{\mathbf{std}}{}_{\Sigma}(I) \neq \bot$. Then $chase^{\mathbf{std}}{}_{\Sigma}(I) \leftrightarrow chase^{\mathbf{core}}{}_{\Sigma}(I)$ and $chase^{\mathbf{core}}{}_{\Sigma}(I) \models \Sigma$, even more, $Core(chase^{\mathbf{std}}{}_{\Sigma}(I)) = chase^{\mathbf{core}}{}_{\Sigma}(I)$.*

Before ending this section about chase variations, let us summarize the differences between the presented chase algorithms. First we saw that in case the algorithm terminates and does not fail with input I and Σ, then the instances returned by all of the presented chase variations are homomorphically equivalent. We also saw that the complexity of testing the existence of a trigger is slightly easier than testing the existence of an active trigger. From this it follows that the oblivious and semi-oblivious-chase steps are less expensive than the standard-chase step that is also less expensive than the core-chase step. On the other hand, a set of dependencies is more likely to terminate for the core chase than any of the other chase variations. Figure 1 shows the set inclusion relationship between different termination classes.

4 Sufficient conditions for the chase termination

In the previous section we saw that it is undecidable to test for all the chase variations if the chase will terminate for a given instance and a given set of dependencies. This motivated the research community to find large classes of tgd sets that ensure termination of the standard-chase algorithm for all instances. In this section some of these classes of tgd sets will be presented for which it is known that the standard chase terminates on all execution branches for all input instances. We also investigate here if these classes are sufficient to guarantee the chase termination for other chase variations beside the standard chase. For this, we will say that a class of sets of tgds $\mathcal{C}$ is closed under enrichment if $\Sigma \in \mathcal{C}$ implies $\hat{\Sigma} \in \mathcal{C}$. Similarly, the class $\mathcal{C}$ is closed under semi-enrichment if $\Sigma \in \mathcal{C}$ implies $\tilde{\Sigma} \in \mathcal{C}$. From Theorems 18 and 22 it directly follows:

▶ **Corollary 27.** *Let $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$.*

1. *if $\mathcal{C}$ is closed under enrichment, then $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$.*
2. *if $\mathcal{C}$ is closed under semi-enrichment, then $\mathcal{C} \subseteq \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.*

We will use the previous corollary to show that the termination classes presented here not only guarantee the termination for the standard-chase algorithm but also for the semi-oblivious-chase algorithm.

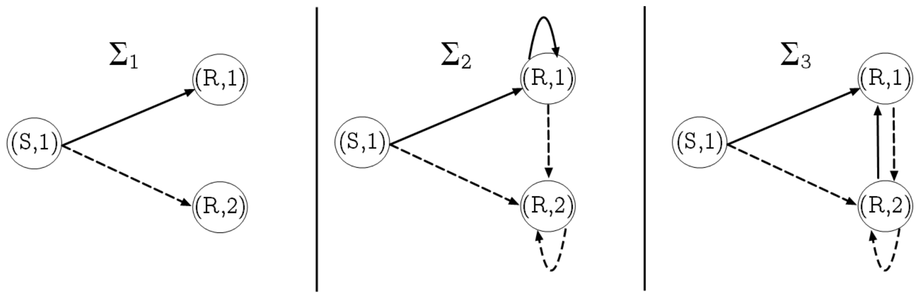


Figure 2 Extended dependency graphs associated with dependencies from Example 30.

4.1 Rich acyclicity

The class of *richly acyclic* set of dependencies was introduced by Hernich and Schweikardt in [30] in a different context, and it was shown in [25] that this class guarantees termination for the oblivious chase on any input instance.

▶ **Definition 28.** [19, 12] For a given database schema $\mathbf{R}$ define a *position* in $\mathbf{R}$ to be a pair (R, k), where R is a relation symbol in $\mathbf{R}$ and k a natural number with $1 \leq k \leq arity(R)$, such that k identifies the k-th element in R.

The notion of extended-dependency graph is defined as follows:

▶ **Definition 29.** [30] Let Σ be a set of tgds over schema $\mathbf{R}$. The *extended-dependency graph* associated with Σ is a directed edge-labeled graph $G^E_\Sigma = (V, E)$, such that each vertex represents a position in $\mathbf{R}$ and $((R, i), (S, j)) \in E$, if there exists a tgd $\xi \in \Sigma$ of the form $\alpha(\bar{x}, \bar{y}) \to \exists \bar{z}\ \beta(\bar{x}, \bar{z})$, and if one of the following holds:

1. $x \in \bar{x}$ and x occurs in α on position (R, i) and in β on position (S, j). In this case the edge is labeled as universal;
2. $x \in \bar{x} \cup \bar{y}$ and x occurs in α on position (R, i) and variable $z \in \bar{z}$ that occurs in β on position (S, j). In this case the edge is labeled as existential.

The following example illustrates the previous definitions:

▶ **Example 30.** Consider database schema $\mathbf{R} = \{S, R\}$, with $arity(S) = 1$ and $arity(R) = 2$. The set $\{(S, 1), (R, 1), (R, 2)\}$ represents all positions in $\mathbf{R}$. Let Σ_1 contain the following dependency over $\mathbf{R}$:

$$\xi_{11} \quad = \quad S(x) \to \exists y\ R(x, y)$$

let Σ_2 contain the following dependencies:

$$\xi_{21} \quad = \quad S(x) \to \exists y\ R(x, y), \text{ and}$$
$$\xi_{22} \quad = \quad R(x, y) \to \exists z\ R(x, z)$$

and, finally, let Σ_3 be a slight modification of Σ_2:

$$\xi_{31} \quad = \quad S(x) \to \exists y\ R(x, y), \text{ and}$$
$$\xi_{32} \quad = \quad R(x, y) \to \exists z\ R(y, z).$$

Figure 2 captures the extended dependency graphs associated with Σ_1, Σ_2 and Σ_3 (note that the existential edges are represented as dotted lines).

▶ **Definition 31.** [30] A set of tgds Σ is said to be *richly acyclic* if its extended dependency graph does not contain a cycle going through an existential edge. We denote by RA the class of all richly acyclic tgd sets.

Note that the problem of testing if $\Sigma \in \mathsf{RA}$ is polynomial in size of Σ. Returning to Example 30, Σ_1 is richly acyclic because it does not contain any cycles. On the other hand, neither Σ_2 or Σ_3 are richly acyclic. As we will see in the following subsection, the RA ensures termination for the standard-chase algorithm on any input instance, that is $\mathsf{RA} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. The next theorem follows directly from Corollary 27 and the observation that RA is closed under enrichment.

▶ **Theorem 32.** [25] *Let $\Sigma \in \mathsf{RA}$ and let I be an instance. Then there exists a polynomial in size of I that bounds the length of every oblivious-chase sequence of I and Σ.*

Mainly, the previous result states that $\mathsf{RA} \subseteq \mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and based on the termination hierarchy represented in Figure 1, it follows that any set of tgds from RA ensures termination of any of the chase variation previously presented on any input instances.

4.2 Weak acyclicity

Fagin et al. [19] introduced the class of *weakly acyclic* dependencies as a class of sets of tgds that ensures standard-chase termination on all execution branches for all input instances. Intuitively weak acyclicity checks if the set of tgds does not have a cyclic condition such that another new null value forces the adding of a new null value.

▶ **Definition 33.** [19] Let Σ be a set of tgds over schema $\mathbf{R}$. The *dependency graph* associated with Σ is a directed edge-labeled graph $G_\Sigma = (V, E)$, such that the set of vertexes V represents the positions in $\mathbf{R}$. There is an edge $((R,i),(S,j)) \in E$, if there exists a dependency $\xi \in \Sigma$ of the form $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$. There exists $x \in \bar{x}$ such that x occurs in position (R,i) in α and if one of the following holds:
1. x occurs in β in position (S,j). In this case the edge is labeled as universal;
2. there exists variable $z \in \bar{z}$ which occurs in position (S,j) in β. In this case the edge is labeled as existential.

▶ **Definition 34.** [19] A set of tgds Σ is said to be *weakly acyclic* if the corresponding dependency graph does not have any cycle going through an existential edge. By WA is denoted the class of all weakly acyclic sets of tgds.

Note that the problem of testing if $\Sigma \in \mathsf{WA}$ is polynomial in size of Σ. Figure 3 illustrates the dependency graphs associated with the dependencies from Example 30. Based on the previous definition, it follows that : Σ_1 is weakly acyclic as the dependency graph does not contain any cycles; Σ_2 is weakly acyclic as its dependency graph has a cycle going only through universal edges; Σ_3 is not weakly acyclic as it has a cycle going through an existential edge. From the definitions of the RA and WA classes, it follows that $\mathsf{RA} \subseteq \mathsf{WA}$. Also because $\Sigma_2 \in \mathsf{WA}$ and $\Sigma_2 \notin \mathsf{RA}$, it follows that the inclusion is strict, that is $\mathsf{RA} \subset \mathsf{WA}$.

▶ **Theorem 35.** [19] *Let $\Sigma \in \mathsf{WA}$ and let I be an instance. Then there exists a polynomial in size of I that bounds the length of every standard-chase sequence of I and Σ.*

From the chase termination hierarchy it follows that if $\Sigma \in \mathsf{WA}$, then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\exists}$ and $\Sigma \in \mathsf{CT}^{\mathsf{core}}_{\forall\forall}$. Besides, even if $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ the classes $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$ and WA are incomparable. For this consider the tgd set Σ from Example 17. It is easy to see that $\Sigma \in (\mathsf{WA} \setminus \mathsf{CT}^{\mathsf{obl}}_{\forall\forall})$. For the other direction consider $\Sigma' = \{S(y), R(x,y) \to \exists z\ R(y,z)\}$, clearly $\Sigma' \in (\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \setminus \mathsf{WA})$.

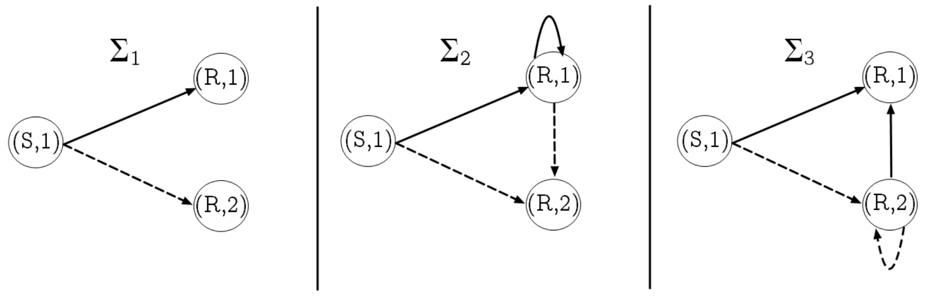


Figure 3 Dependency graphs associated with dependencies from Example 30.

From the observation that the class WA is closed under semi-enrichment, Theorem 35 and Corollary 27, it follows:

▶ **Theorem 36.** $\mathsf{WA} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.

Proof. For the strict inclusion part of this theorem consider the same set of dependencies $\Sigma' = \{S(y), R(x,y) \to \exists z\ R(y,z)\}$. ◀

4.3 Safe dependencies

Meier, Schmidt and Lausen [41] observed that the weak acyclicity condition takes into account nulls that may not create infinite standard-chase sequences. For example, consider the set $\Sigma = \{\xi\}$ [41] where:

$$\xi \ = \ R(x,y,z), S(y) \to \exists w\ R(y,w,x).$$

Figure 4a) represents the corresponding dependency graph with cycles going through existential edges involving position $(R,2)$. Thus, the dependency is not weakly acyclic. On the other hand, the newly created null in position $(R,2)$ may create new null values only if the same null also appears in position $(S,1)$. Based on the given dependency, new nulls cannot be generated in position the $(S,1)$. Hence, this dependency can not cyclically create new nulls.

In order to introduce the notion of safe dependencies, we first need to define the following concept.

▶ **Definition 37.** [12] The *affected positions* associated with a set of tgds Σ is the set $\mathit{aff}(\Sigma)$ defined as follows. For all positions (R,i) that occur in the head of some tgd $\xi \in \Sigma$, then

1. if an existential variable appears in position (R,i) in ξ, then $(R,i) \in \mathit{aff}(\Sigma)$;
2. if universally quantified variable x appears in position (R,i) in the head and x appears only in affected positions in the body, then $(R,i) \in \mathit{aff}(\Sigma)$.

Intuitively, the affected positions are those where new null values can occur during the chase process. For example, the set of affected positions associated with the set of dependencies $\Sigma = \{R(x,y,z), S(y) \to \exists w\ R(y,w,x)\}$ is $\mathit{aff}(\Sigma) = \{(R,2)\}$.

▶ **Definition 38.** [41] The *propagation graph* for a set of tgds Σ is a directed edge labeled graph $P_\Sigma = (\mathit{aff}(\Sigma), E)$. Where $((R,i),(S,j)) \in E$ if there exists a dependency $\xi \in \Sigma$ of the form $\alpha(\bar{x},\bar{y}) \to \exists \bar{z}\ \beta(\bar{x},\bar{z})$, there exists a variable x that occurs in α in position (R,i), x occurs only in affected positions in α and one of the following holds:

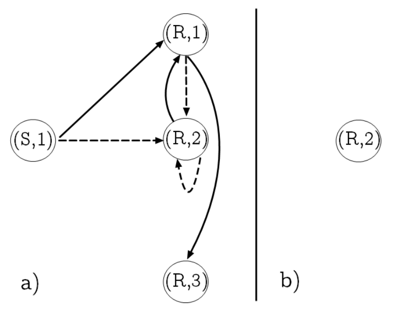


Figure 4 a) Dependency graph, b) Propagation graph for $\{R(x, y, z), S(y) \to \exists w\ R(y, w, x)\}$.

1. x appears in β in affected position (S, j). In this case the edge is labeled as universal;
2. there exists variable $z \in \bar{z}$ which occurs in position (S, j) in β. In this case the edge is labeled existential.

Considering the same dependency set $\Sigma = \{R(x, y, z), S(y) \to \exists w\ R(y, w, x)\}$, Figure 4a) represents the corresponding dependency graph and Figure 4b) the corresponding propagation graph. Note that the propagation graph contains only one node, corresponding to the affected position $(R, 2)$. Because y appears in the head in a non-affected position $(S, 1)$, it follows that there are no edges in the propagation graph.

▶ **Definition 39.** [41] A set of tgds Σ is called *safe* if its propagation graph P_Σ does not have a cycle going through an existential edge. By SD is denoted the class of all safe sets of tgds.

Note that the problem of testing if $\Sigma \in \mathsf{SD}$ is polynomial in size of Σ. In our example, the dependency graph did not contain any edges (see Figure 4b)), hence $\Sigma \in \mathsf{SD}$.

▶ **Theorem 40.** [41] *Let $\Sigma \in \mathsf{SD}$. Then there exists a polynomial in size of I that bounds the length of every standard-chase sequence of I and Σ.*

From our running example in this subsection and from the definition of the SD class, it follows that $\mathsf{WA} \subset \mathsf{SD}$. Also, similarly to the weakly acyclic class it can be shown that SD is closed under semi-enrichment. Thus, because the previous theorem states that $\mathsf{SD} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, it follows that actually $\mathsf{SD} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall} \subset \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. This means that given a set of tgds which is safe, one may use the semi-oblivious chase to compute an instance which is (see previous section) homomorphically equivalent to any instance returned by the standard chase with the same input. Finally, it can be easily noted that, even if SD is bigger than WA, it does not contain the termination class $\mathsf{CT}^{\mathsf{obl}}_{\forall\forall}$. For this consider $\Sigma = \{R(x, x) \to \exists z\ R(x, y)\}$, clearly $\Sigma \in (\mathsf{CT}^{\mathsf{obl}}_{\forall\forall} \setminus \mathsf{SD})$.

4.4 Super weak acyclicity

The following class of sets of tgds properly extends the class of safe sets of tgds and consequently the class of weakly acyclic and richly acyclic sets of tgds. The new class of dependencies, introduced by Marnette [38], beside omitting the nulls that can't generate infinite chase sequences, as in the case of safe dependencies, also takes into account the repeating variables. For a more uniform presentation of the sufficient classes, we will slightly change the notations used in [38].

In this subsection we assume that the set of dependencies Σ has distinct variable names in each tgd. We also assume that there exists a total order between the atoms in each dependency. With this, we can now define the *atom position* to be a triple (ξ, R, i), where ξ is a dependency in Σ, R is a relation name that occurs in ξ, i a positive integer $i \leq n$, where n is the maximum number of occurrences of R in ξ, given by the total order between the atoms in the tgd. Clearly each atom position uniquely identifies an atom in Σ. Similarly to the notion of position, a *place* can be defined to be a pair $((\xi, R, i), k)$, where (ξ, R, i) is an atom position and $1 \leq k \leq arity(R)$. Intuitively, the place identifies the variable that appears in the k-th attribute in the atom represented by (ξ, R, i).

Let $Var(\xi)$ denote the set of variables that occurs in dependency ξ. As mentioned, for any two distinct dependencies ξ_1 and ξ_2 from Σ, we have $Var(\xi_1) \cap Var(\xi_2) = \emptyset$. The mapping Var is extended in the natural way to a set of dependencies Σ, $Var(\Sigma) = \cup_{\xi \in \Sigma} Var(\xi)$. Similarly, we define the mappings $Var^{\exists}$ and $Var^{\forall}$ that map each dependency ξ to the set of existentially quantified variables in ξ and to the universally quantified variables in ξ, respectively. Clearly for each dependency ξ, $Var^{\exists}(\xi)$ and $Var^{\forall}(\xi)$ represent a partition of $Var(\xi)$.

Given a tgd ξ and $y \in Var^{\exists}(\xi)$, $\mathsf{Out}(\xi, y)$ is defined to be the set of places in the head of ξ where y occurs. Given a set a tgd ξ and $x \in Var^{\forall}(\xi)$, $\mathsf{In}(\xi, x)$ is defined to be the set of places in the body of ξ where x occurs. Intuitively, $\mathsf{Out}(\xi, y)$ represents the places where variable y is "exported" when applying tgd ξ. Similarly, $\mathsf{In}(\xi, x)$ represents the places that need to be "filled" for variable x in order for ξ to be applied.

Given an atom position (ξ, R, i), a substitution θ is a function that maps each variable x that occurs in the atom (ξ, R, i) to a constant if $x \in Var^{\forall}(\xi)$ and to a fresh new constant, if $x \in Var^{\exists}(\xi)$, where by "new fresh" we mean the next unused element in some fixed enumeration of the constants. The atom resulted by replacing each variable in the atom given by (ξ, R, i) with the substitution θ is denoted by $\theta(\xi, R, i)$. Two atoms (ξ_1, R, i_1) and (ξ_2, R, i_2) are said to be *unifiable* if there exist substitutions θ_1 and θ_2 such that $\theta_1(\xi_1, R, i_1) = \theta_2(\xi_2, R, i_2)$. Two places $p_1 = ((\xi_1, R, i_1), k_1)$ and $p_2 = ((\xi_2, R, i_2), k_2)$ are said to be *unifiable* if $k_1 = k_2$ and (ξ_1, R, i_1) is unifiable with (ξ_2, R, i_2). By $p_1 \sim p_2$ it is denoted that p_1 and p_2 are unifiable. Let us define Γ_Σ to be a function that maps each variable x to the set of places where x occurs in Σ. Γ^H_Σ represents the function that maps each variable x to the set of places from the head of some dependency where x occurs. Similarly, the function Γ^B_Σ maps each variable x to the set of places from the body of some dependency where x occurs.

For a better understanding of the previous notions, let us consider the example:

▶ **Example 41.** [47] Let $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x) \rightarrow \exists y, z\, S(x, y, z), \text{ and} \\ \xi_2 &= S(v, w, w) \rightarrow R(w). \end{aligned}$$

The atom positions for Σ are: $(\xi_1, R, 1)$, corresponding with the atom $R(x)$; $(\xi_1, S, 1)$, corresponding with the atom $S(x, y, z)$; $(\xi_2, S, 1)$, corresponding with the atom $S(v, w, w)$; and $(\xi_2, R, 1)$, corresponding with the atom $R(w)$. We have $Var^{\forall}(\Sigma) = \{x, v, w\}$ and $Var^{\exists}(\Sigma) = \{y, z\}$. Clearly $(\xi_1, R, 1)$ is unifiable with $(\xi_2, R, 1)$, consider for example unifiers $\theta_1 = \{x/a\}$ and $\theta_2 = \{w/a\}$ where both variables x and w are in $Var^{\forall}(\Sigma)$. On the other hand, the atom positions $(\xi_1, S, 1)$ and $(\xi_2, S, 1)$ are not unifiable because y and z are existentially quantified variables and thus any unifier will map y and z to distinct constants (recall that each existential variable is mapped to a new fresh constants). Hence, we only have

$((\xi_1, R, 1), 1) \sim ((\xi_2, R, 1), 1)$. For variable x the set $\Gamma_\Sigma(x) = \{((\xi_1, R, 1), 1), ((\xi_1, S, 1), 1)\}$, $\Gamma^B_\Sigma(x) = \{((\xi_1, R, 1), 1)\}$ and $\Gamma^H_\Sigma(x) = \{((\xi_1, S, 1), 1)\}$.

Given two sets of places P and Q, we denote $P \sqsubseteq Q$ if for all $p \in P$ there exists $q \in Q$ such $p \sim q$. Let us now define mapping $\mathsf{Move}(\Sigma, Q)$ that gives the smallest set of places P such that $Q \subseteq P$, and for all variables x that occurs in a body of some dependency $\xi \in \Sigma$ if $\Gamma^B_\Sigma(x) \sqsubseteq P$ then $\Gamma^H_\Sigma(x) \subseteq P$. Intuitively, the $\mathsf{Move}(\Sigma, Q)$ returns the smallest set of places such that new atoms may be generated in those positions by chasing some atoms given by the places in Q.

▶ **Definition 42.** [38] Given Σ a set of tgds and $\xi_1, \xi_2 \in \Sigma$, we say ξ_1 triggers ξ_2 in Σ, and it is denoted by $\xi_1 \rightsquigarrow_\Sigma \xi_2$, iff there exist a variable $y \in Var^\exists(\xi_1)$ and a variable $x \in Var^\forall(\xi_2)$ occurring in both the body and the head of ξ_2 such that:

$$\mathsf{In}(\xi_2, x) \sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y)).$$

▶ **Definition 43.** [38] A set of tgds Σ is said to be *super-weakly acyclic* iff the trigger relation $\rightsquigarrow_\Sigma$ is acyclic. We denote by SwA the set off all super-weakly acyclic tgd sets.

▶ **Example 44.** Let us consider the same set of dependencies $\Sigma = \{\xi_1, \xi_2\}$ from Example 41. The place $((\xi_1, S, 1), 1)$ is not unifiable with $((\xi_2, S, 1), 1)$, thus $\mathsf{In}(\xi_2, \mathsf{w}) \not\sqsubseteq \mathsf{Move}(\Sigma, \mathsf{Out}(\xi_1, y))$, that is $\xi_1 \not\rightsquigarrow_\Sigma \xi_2$. Similarly, ξ_2 does not contain any existential variables and so it follows that $\xi_2 \not\rightsquigarrow_\Sigma \xi_1$. As both dependencies do not share common relation names in the head and body, it follows that $\xi_1 \not\rightsquigarrow_\Sigma \xi_1$ and $\xi_2 \not\rightsquigarrow_\Sigma \xi_2$. That is the relation $\rightsquigarrow_\Sigma$ does not induce any cycle, following that Σ is super-weakly acyclic. Moreover, it can be seen that Σ is not safe as between the affected positions $(R, 1)$ and $(S, 2)$ there exists a cycle through an existential edge in the corresponding propagation graph.

Marnette [38] showed that the membership problem *Is* $\Sigma \in \mathsf{SwA}$*?* is polynomial in the size of Σ. Spezzano and Greco [47] also proved that $\mathsf{SD} \subset \mathsf{SwA}$, that is the super-weak acyclic class properly contains the safe dependencies. The class SwA is closed in adding atoms to the body of dependencies. Thus given a set of tgds Σ, then any set of dependencies Σ' obtained from Σ by adding new atoms in the body of any dependency remains super-weakly acyclic.

▶ **Theorem 45.** [38] *Let* $\Sigma \in \mathsf{SwA}$ *and let* I *be an instance. Then there exists a polynomial in the size of* I *that bounds the length of every semi-oblivious-chase sequence of* I *and* Σ*. Thus,* $\mathsf{Swa} \subset \mathsf{CT}^{\mathsf{sobl}}_{\forall\forall}$.

This concludes that the super-weakly acyclic class of tgd sets is sufficient for the termination of the semi-oblivious, standard and core-chase algorithms for all input instances.

4.5 Stratification

The stratified model of dependencies was introduced by Deutsch et al. in [16]. This class relaxes the condition imposed by weak acyclicity by stratifying the dependencies and check for weak acyclicity on each of these strata instead of checking for the entire set.

▶ **Definition 46.** [16] Let ξ_1 and ξ_2 be two tgds , we write $\xi_1 \prec \xi_2$, if there exist the instances I, J and a vector $\bar{a} \subseteq dom(J)$ such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists an active trigger (ξ_1, h), such that $I \xrightarrow{(\xi_1, h)} J$, and
3. $J \not\models \xi_2(\bar{a})$.

▶ **Example 47.** Consider $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{array}{lcl} \xi_1 & = & R(x,y) \to S(x), \text{ and} \\ \xi_2 & = & S(x) \to R(x,x). \end{array}$$

With the instance $I = \{R(a,b)\}$ and the vector $\bar{a} = (a)$ we have that $I \models \xi_2(a)$; and for the homomorphism $h = \{x/a, y/b\}$ we have $I \xrightarrow{(\xi_1,h)} J$, where $J = \{R(a,b), S(a)\}$. Because $J \not\models \xi_2(a)$, it follows that $\xi_1 \prec \xi_2$. On the other hand, $\xi_2 \not\prec \xi_1$ because for any instance I and vector of constants $\bar{b}$ such that $I \models \xi_1(\bar{b})$ and $I \xrightarrow{(\xi_2,h)} J$, for some active trigger (ξ_2, h), it follows that $J \models \xi_2(\bar{b})$.

The authors of [16] claimed that testing if $\xi_1 \prec \xi_2$ is in NP, as we will show next, this cannot be true unless NP = coNP.

▶ **Theorem 48.** [26] *Given two tgds ξ_1 and ξ_2, the problem of testing if $\xi_1 \prec \xi_2$ is* coNP*-hard.*

Proof. We will use a reduction from the graph 3-colorability problem that is known to be NP-complete. It is also known that a graph G is 3-colorable if and only if there exists a homomorphism from G to K_3, where K_3 is the complete graph with 3 vertices.

A graph $G = (V, E)$, where $V = n$ and $E = m$, is identified with the sequence $G(x_1, \ldots, x_n) = E(x_{i_1}, y_{i_1}), \ldots, E(x_{i_m}, y_{i_m})$ and treat the elements in V as variables. Similarly, we identify the graph K_3 with the sequence

$$K_3(z_1, z_2, z_3) = E(z_1, z_2), E(z_2, z_1), E(z_1, z_3), E(z_3, z_1), E(z_2, z_3), E(z_3, z_2)$$

where z_1, z_2, and z_3 are variables. With these notations, given a graph $G = (V, E)$, we construct tgd's ξ_1 and ξ_2 as follows:

$$\begin{array}{l} \xi_1 = \quad R(z) \to \exists z_1, z_2, z_3\ K_3(z_1, z_2, z_3), \text{ and} \\ \xi_2 = E(x,y) \to \exists x_1, \ldots, x_n\ G(x_1, \ldots, x_n). \end{array}$$

Clearly the reduction is polynomial in the size of G. We will now show that $\xi_1 \prec \xi_2$ iff G is not 3-colorable.

First, suppose that $\xi_1 \prec \xi_2$. Then there exists an instance I and homomorphisms h_1 and h_2, such that $I \models h_2(\xi_2)$. Consider J, where $I \xrightarrow{(\xi_1,h_1)} J$. Thus R^I had to contain at least one tuple, and E^I had to be empty, because otherwise the monotonicity property of the chase we would imply that $J \models h_2(\xi_2)$.

On the other hand, we have $I \xrightarrow{(\xi_1,h_1)} J$, where $J = I \cup \{K_3(h_1'(z_1), h_1'(z_2), h_1'(z_3))\}$, and h_1' is a distinct extension of h_1. Since $E^I = \emptyset$, and we assumed that $J \not\models h_2(\xi_2)$, it follows that there is no homomorphism from G into J, i.e. there is no homomorphism from $G(h_2'(x_1), \ldots, h_2'(x_n))$ to $K_3(h_1'(z_1), h_1'(z_2), h_1'(z_3))$, where h_2' is a distinct extension of h_2. Therefore the graph G is not 3-colorable.

For the other direction, let us suppose that graph G is not 3-colorable. This means that clearly there is no homomorphism from G into K_3. In fact, with these assumptions let us consider $I = \{R(a)\}$, and the two homomorphisms $h_1 = \{z/a\}$ and $h_2 = \{x/h_1'(z_1), y/h_1'(z_2)\}$. It is easy to verify that I, h_1 and h_2 satisfy the three conditions for $\xi_1 \prec \xi_2$. ◀

The obvious upper bound for the problem $\xi_1 \prec \xi_2$ is Σ_2^P. In [26] it is shown that this upper bound can be lowered to Δ_2^P. To the best of our knowledge these are the tidiest bounds found so far for the given problem.

Given a set of tgds Σ, the *chase graph* associated with Σ is a directed graph $G = (V, E)$, where $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec \xi_2$.

▶ **Definition 49.** [16] A set of tgds Σ is said to be *stratified* if the set of dependencies in every simple cycle in the corresponding chase graph is weakly acyclic. The set of all stratified tgd sets is denoted by Str.

In [26] it is shown that the complexity of testing if $\Sigma \in \mathsf{Str}$, for a given Σ, is in Π_2^P. Meier et al. [41] shown that $\mathsf{Str} \not\subseteq \mathsf{CT}_{\forall\forall}^{\mathsf{std}}$ but actually $\mathsf{Str} \subseteq \mathsf{CT}_{\forall\exists}^{\mathsf{std}}$, that is the stratification guarantees the termination only on some standard-chase execution branches for all input instances.

▶ **Example 50.** [41] Consider $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$, where:

$$\begin{aligned}
\xi_1 &= R(x) \to S(x,x);\\
\xi_2 &= S(x,y) \to \exists z\, T(y,z);\\
\xi_3 &= S(x,y) \to T(x,y), T(y,z); \text{ and}\\
\xi_4 &= T(x,y), T(x,z), T(z,x) \to R(y).
\end{aligned}$$

We have $\Sigma \in \mathsf{Str}$ is stratified since $\xi_1 \prec \xi_2$, $\xi_1 \prec \xi_3 \prec \xi_4 \prec \xi_1$, and the set $\{\xi_1, \xi_3, \xi_4\}$ is weakly acyclic. Let $I = \{R(a)\}$. The standard-chase execution branch that triggers repeatedly dependencies ξ_1, ξ_2, ξ_3 and ξ_4 never terminates. On the other hand, the chase sequences that never trigger ξ_2 will terminate.

Meier et al. [41] changed the stratification definition in order to guarantee termination on all execution branches for all instances.

▶ **Definition 51.** [41] Let ξ_1 and ξ_2 be two tgds we write $\xi_1 \prec_{\mathsf{c}} \xi_2$, if there exist instances I, J and tuple $\bar{a} \subseteq dom(J)$, such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists trigger (ξ_1, h), such that $I \xrightarrow{*,(\xi_1,h)} J$, and
3. $J \not\models \xi_2(\bar{a})$.

Given a set of tgds Σ, the *c-chase graph* associated with Σ is a directed graph $G_{\mathsf{C}} = (V, E)$. With $V = \Sigma$ and $(\xi_1, \xi_2) \in E$ iff $\xi_1 \prec_{\mathsf{C}} \xi_2$. A set of tgds Σ is said to be *c-stratified* if the set of dependencies in every simple cycle in the c-chase graph is weakly acyclic. The set of all c-stratified tgd sets is denoted by CStr.

▶ **Theorem 52.** [41] *Let $\Sigma \in \mathsf{CStr}$ and let I be an instance. Then there exists a polynomial, in size of I, that bounds the length of every standard chase sequence of I and Σ.*

Using the same reduction as in Theorem 48, it can be shown that the problem of testing if $\xi_1 \prec_{\mathsf{C}} \xi_2$ is coNP-hard and it is in Δ_2^P. Also the problem of testing if $\Sigma \in \mathsf{CStr}$, for a given Σ, it is in Π_2^P.

Meier et al. [41] showed that $\mathsf{WA} \subset \mathsf{CStr}$ and that $\mathsf{SD} \nparallel \mathsf{CStr}$[1]. Also Spezzano and Greco [47] proved that $\mathsf{SwA} \nparallel \mathsf{CStr}$, that is the super-weakly acyclic class is not comparable with the c-stratified class. Also based on the observation that CStr is closed under semi-enrichment, it follows that $\mathsf{CStr} \subset \mathsf{CT}_{\forall\forall}^{\mathsf{sobl}}$.

[1] The notation $A \nparallel B$ is shorthand for $A \not\subseteq B$ and $A \not\supseteq B$

4.6 Inductively restricted dependencies

Another class of dependencies that guarantees the standard-chase termination is the inductively restricted set of tgds. Note that the stratification method lifts the weakly acyclic class of dependencies to the class of c-stratified dependencies. The inductively restricted class generalizes the stratification method while still keeping the termination property for the standard-chase algorithm. This generalization is done using the so-called *restriction systems* [41]. With the help of the restriction systems, Meier et al. [41] define the new sufficient condition called *inductive restriction* that guarantees the standard-chase algorithm termination on all execution branches for all instances. From this condition a new hierarchy of classes of dependencies is revealed with the same termination property, called the *T-hierarchy*. Note that the inductive restriction condition presented here is given from the erratum (http://arxiv.org/abs/0906.4228) and not from [41], where the presented condition, as mentioned in the erratum, does not guarantee the standard-chase termination on all branches for all instances.

Let Σ be a set of tgds, I an instance and A a set of nulls. The set of all positions (R, i) such that there exists a tuple in I that contains a variable from A in position (R, i) is denoted by null-pos(A, I).

Similarly to relation "$\prec$" for the stratified dependencies, the binary relation "$\prec_P$" is defined for the inductive restriction condition, where P is a set of positions.

▶ **Definition 53.** [41] Let Σ be a set of tgds and P a set of positions. Let ξ_1, ξ_2 be two dependencies in Σ. It is said that $\xi_1 \prec_P \xi_2$ if there exist instances I,J and vector $\bar{a} \subseteq dom(J)$, such that:

1. $I \models \xi_2(\bar{a})$, and
2. there exists a trigger (ξ_1, h), such that $I \xrightarrow{*,(\xi_1,h)} J$, and
3. $J \not\models \xi_2(\bar{a})$, and
4. there exists $X \in \bar{a} \cap \mathsf{Null}$ in the head of $\xi_2(\bar{a})$, such that null-pos$(\{X\}, I) \subseteq P$.

▶ **Example 54.** Consider Σ containing a single tgd $\xi = R(x, y) \to \exists z\, R(y, z)$. In Section 3 we saw that there are instances I such that standard-chase algorithm does not terminate on all branches for I and Σ. It is easy to see that with instances $I = \{R(a, b)\}$ and $J = \{R(a, b), R(b, X)\}$, and vector $\bar{a} = (b, X)$, conditions 1,2 and 3 from the previous definition are fulfilled. For the 4th condition, consider $X \in \bar{a}$, then we have $\xi(\bar{a})$ which represents the formula $R(a, X) \to \exists z R(X, z)$. Thus, X occurs in the head of $\xi(\bar{a})$. On the other hand, null-pos$(\{X\}, I) = \emptyset$, instance I does not contain any labeled nulls, hence for any set P, null-pos$(\{X\}, I) \subseteq P$. Thus, $\xi \prec_P \xi$, for any set of positions P.

▶ **Definition 55.** [41] Let P be a set of positions and ξ a tgd. By aff-cl(ξ, P) is denoted the set of positions (R, i) from the head of ξ such that one of the following holds:

1. for all $x \in Var^{\forall}(\xi)$[2], with x occurs in (R, i), x occurs in the body of ξ only in positions from P, or
2. position (R, i) contains a variable $x \in Var^{\exists}(\xi)$.

For the tgd in Example 54, we have aff-cl$(\xi, P) = \{(R, 1), (R, 2)\}$, where $P = \{(R, 2)\}$. Given a set of dependencies Σ, the set of all positions in Σ is written as $pos(\Sigma)$.

[2] Recall that by $Var^{\forall}(\xi)$ we denote the set of all universally quantified variables in ξ and by $Var^{\exists}(\xi)$ the set of all existentially quantified variables in ξ.

▶ **Definition 56.** [41] A *2-restriction system* is a pair $(G(\Sigma), P)$, where $G(\Sigma)$ is a directed graph (Σ, E) and $P \subseteq pos(\Sigma)$ such that:

1. for all $(\xi_1, \xi_2) \in E$, aff-cl$(\xi_1, P) \cap pos(\Sigma) \subseteq P$ and aff-cl$(\xi_2, P) \cap pos(\Sigma) \subseteq P$, and
2. for all $\xi_1 \prec_P \xi_2$, $(\xi_1, \xi_2) \in P$.

A 2-restriction system is *minimal* if it is obtained from $((\Sigma, \emptyset), \emptyset)$ by a repeated application of constraints 1 and 2, from the previous definition, such that P is extended only by those positions that are required to satisfy condition 1. Let us denote by $part(\Sigma, 2)$ the set that contains the sets of all strongly connected components in a minimal 2-restriction system.

▶ **Example 57.** Returning to our dependency from Example 54, the minimal 2-restriction system is computed as follows. Consider pair $((\{\xi\}, \emptyset), \emptyset)$. Previously we showed that $\xi \prec_P \xi$, for any set of positions P, by particularization we have $\xi \prec_\emptyset \xi$. Thus, we add edge (ξ, ξ) to E. Using condition 1 from Definition 56 we have aff-cl$(\xi, \emptyset) = \{(R, 2)\}$. That is we add position $(R, 2)$ to P. By repeating this process once again with $P = \{(R, 2)\}$, we add to P the position $(R, 1)$ too. Hence, the minimal 2-restriction system is $((\Sigma, \{(\xi, \xi)\}, \{(R, 1), (R, 2)\}\})$. The only connected component in this restriction system is $\{\xi\}$.

In [41], Meier et al. provide a simple algorithm that computes the set $part(\Sigma, 2)$.

▶ **Definition 58.** [41] A set Σ of tgds is called *inductively restricted* iff every $\Sigma' \in part(\Sigma, 2)$ is safe. The set of all inductively restricted tgd sets is denoted by IR.

Using the same reduction from the proof of Theorem 48, it can be shown that the problem of testing if $\xi_1 \prec_P \xi_2$, for a given ξ_1, ξ_2 and P, is coNP-hard and it can be solved in Σ_2^P. Similarly to the stratification case it can be shown that the complexity of testing if $\Sigma \in \mathsf{IR}$ is in Π_3^P. From the definition, it directly follows that $\mathsf{SD} \subset \mathsf{IR}$. To this Meier et al. [41] also showed that $\mathsf{Str} \nparallel \mathsf{IR}$ and that $\mathsf{CStr} \subset \mathsf{IR}$. On the other hand, the classes SwA and IR are incomparable [47], that is $\mathsf{SwA} \nparallel \mathsf{IR}$.

▶ **Example 59.** [41] Consider the following set of tgds Σ:

$$\begin{aligned}\xi_1 &= S(x), E(x,y) \rightarrow E(y,x), \text{ and} \\ \xi_2 &= S(x), E(x,y) \rightarrow \exists z\ E(y,z), E(z,x).\end{aligned}$$

It can be easily observed that Σ is neither stratified nor safe, but it is inductively restricted.

▶ **Theorem 60.** [41] *Let $\Sigma \in \mathsf{IR}$ and let I be an instance. Then there exists a polynomial, in size of I, that bounds the length of every standard-chase sequence of I and Σ.*

Meier, Schmidt and Lausen [41] observed that the inductive restriction criterion can be extended to form a hierarchy of classes that ensure the standard-chase termination on all branches for all instances. Intuitively, the lowest level of this hierarchy, noted $T[2]$, is the class of inductively restricted dependencies. Level $T[k]$, $k > 2$ is obtained by extending the binary relation $\prec_P$ to a k-ary relation $\prec_{k,P}$. Intuitively, $\prec_{k,P}$ $(\xi_1, \ldots, \xi_k)$ means that there exists a standard-chase sequence such that firing ξ_1 will also cause ξ_2 to fire. This in turn will cause ξ_3 to fire and so on until ξ_k. Based on this new relation, the set $part(\Sigma, k)$ is computed similarly to $part(\Sigma, 2)$. The algorithm that computes $part(\Sigma, k)$ can be found in [41]. For all $k \leq 2$, it is shown that $T[k] \subset T[k+1]$.

It is easy to check that the previous hierarchy is closed under semi-enrichment, following from Corollary 27 that for any k we have $T[k] \subset \mathsf{CT}_{\forall\forall}^{\mathsf{sobl}}$. Also in [44] it is shown that $T[k] \nparallel \mathsf{CT}_{\forall\forall}^{\mathsf{obl}}$. More recently, the $T[k]$ hierarchy of classes was extended by Meier et al. [42]

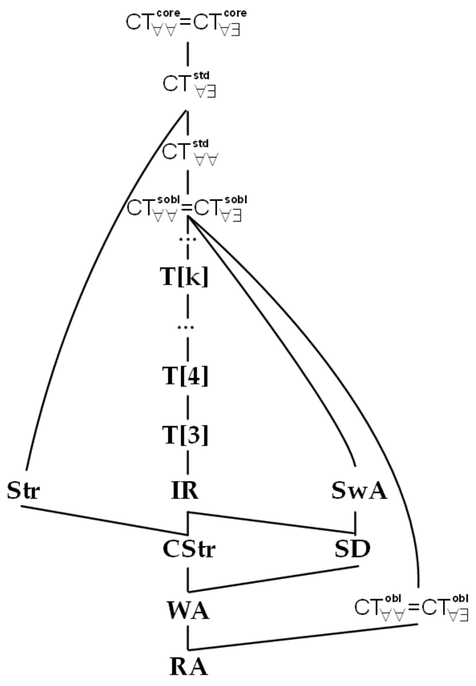


Figure 5 Relationship between chase termination classes.

to the $\forall\exists - T[k]$ hierarchy of classes that ensures the standard-chase termination on at least one execution branch and it showed that $T[k] \subset \forall\exists - T[k]$, for any $k > 1$.

Figure 5 illustrates, as a Hasse diagram, the subset relationship between the termination classes presented.

Before concluding this subsection, we need to mention that more recently Greco et al. [27] extended the classes of dependencies that ensure the standard-chase termination to new large classes based on a stratification based method called local stratification.

4.7 The rewriting approach

Spezzano and Greco [47] noticed that all the previous classes may be extended by using a rewriting technique. Intuitively, if $\mathbf{T}$ is one of the classes $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$, then instead of directly checking if a set of dependencies $\Sigma \in \mathbf{T}$, we check if $\mathrm{Adn}(\Sigma) \in \mathbf{T}$, where $\mathrm{Adn}(\Sigma)$ is an adornment based rewriting of Σ such that, if $\mathrm{Adn}(\Sigma) \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$, then $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. Where the adornment of a predicate p of arity m is a string of the length m over the alphabet $\{b, f\}$. An adorned atom is of the form $p^{\alpha_1,\alpha_2,\ldots,\alpha_m}(x_1, x_2, \ldots, x_m)$; if $\alpha_i = b$, then variable x_i is considered bounded, otherwise the variable is considered free.

Due to the space constraints we will present this method following a simple example.

Consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$ [47]:

$$\begin{aligned} \xi_1 &= N(x) \to \exists y\, E(x, y), \text{ and} \\ \xi_2 &= S(x), E(x, y) \to N(y). \end{aligned}$$

The affected positions in Σ are $(E, 1)$,$(E, 2)$ and $(N, 1)$. As the corresponding propagation graph contains a cycle, through an existential edge, involving positions $(N, 1)$ and $(E, 2)$, it follows that $\Sigma \notin \mathsf{SD}$. Construct the set of dependencies $\mathrm{Adn}(\Sigma)$ as follows:

1. For all predicate symbols p of arity m in Σ add the tgd:

$$\forall x_1, x_2, \ldots, x_m\ p(x_1, x_2, \ldots, x_m) \to p^{\alpha_1, \alpha_2, \ldots, \alpha_m}(x_1, x_2, \ldots, x_m)$$

   where, for all positive $i \leq m$, $\alpha_i = b$.
   In our example Σ contains the following predicate symbols $\{E, S, N\}$, that is we add to $\text{Adn}(\Sigma)$ the following set of tgds:
   $\xi'_1 \ = \ E(x, y) \to E^{b\,b}(x, y);$
   $\xi'_2 \ = \ N(x) \to N^{b}(x);$ and
   $\xi'_3 \ = \ S(x) \to S^{b}(x).$
2. Repeat to create new adornment predicate symbols based on the existing dependencies, until none can be added. That is, if a variable in the head is marked as bounded (free) and if it occurs only bounded (free) places in the body. All existential variables in the head are marked as free.
   Returning to our example and using ξ_1 from Σ and the new adornment N^b, we add the following dependency to $\text{Adn}(\Sigma)$:
   $\xi'_4 \ = \ N^{b}(x) \to \exists y\ E^{b\,f}(x, y).$
   Similarly, based on tgd ξ_2 from Σ and new adornments S^b and $E^{b\,b}$, we add the following dependency to $\text{Adn}(\Sigma)$:
   $\xi'_5 \ = \ S^{b}(x), E^{b\,b}(x, y) \to N^{b}(y).$
   Repeating this process, we add the following tgds to $\text{Adn}(\Sigma)$:
   $\xi'_6 \ = \ S^{b}(x), E^{b\,f}(x, y) \to N^{f}(y),$ and
   $\xi'_7 \ = \ N^{f}(x) \to \exists y\ E^{f\,f}(x, y).$
   After this point no other adornments can be created.
3. Finally, for each of the adornment predicate p^α in $\text{Adn}(\Sigma)$, add a new dependency in $\text{Adn}(\Sigma)$ that "copies" p^α to a new $\hat{p}$ predicate symbol. In this example the following new dependencies are added:
   $\xi'_8 \ = \ N^{b}(x) \to \hat{N}(x);$
   $\xi'_9 \ = \ N^{f}(x) \to \hat{N}(x);$
   $\xi'_{10} \ = \ S^{b}(x) \to \hat{S}(x);$
   $\xi'_{11} \ = \ E^{b\,b}(x, y) \to \hat{E}(x, y);$
   $\xi'_{12} \ = \ E^{b\,f}(x, y) \to \hat{E}(x, y);$ and
   $\xi'_{13} \ = \ E^{f\,f}(x, y) \to \hat{E}(x, y).$

In [47], it is proved that $\Sigma \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$ if and only if $\text{Adn}(\Sigma) \in \mathsf{CT}^{\mathsf{std}}_{\forall\forall}$. Returning to our example, it can be noted that the set $\text{Adn}(\Sigma)$ is safe. Thus, using the previous observation, it results that even if the set Σ was not safe, the standard chase will terminate on all branches on Σ with any instances.

▶ **Theorem 61.** [41] *Let* $\mathbf{T}$ *be one of the classes* $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$*, let* Σ *be a set of tgds and let* I *be an instance. Then, if* $Adn(\Sigma) \in \mathbf{T}$*, there exists a polynomial, in size of* I*, that bounds the length of every standard-chase sequences of* I *and* Σ*.*

Even more, Spezzano and Greco [47] proved that these rewritings strictly extend the classes of dependencies.

▶ **Theorem 62.** [41] *Let* $\mathbf{T}$ *be one of the classes* $\{\mathsf{WA}, \mathsf{SD}, \mathsf{SwA}, \mathsf{Str}, \mathsf{CStr}\}$ *and let denote by* $\mathsf{Adn}\mathbf{T}$ *the set of all* Σ *such that* $Adn(\Sigma) \in \mathbf{T}$*. Then,* $\mathbf{T} \subset \mathsf{Adn}\mathbf{T}$*.*

More recently, this rewriting method was further improved by Greco et al. [27] by indexing the adornment used to specify the free positions. This method ensures that we may equate only variables that have the adornment with the same index.

5 The chase and data exchange

The previous section was mainly focused on presenting different chase algorithms and their termination criteria. This section is dedicated to the instance returned by the chase algorithm and to how it can be used in the data-exchange problem. As we will see, the finite instance returned by any chase variation is strongly related to the notion of *universal model*. Such instances represent a good candidate to be materialized under the target schema in data exchange. Beside computing a general solution for the data-exchange problem, the chase procedure also plays an important role in some related problems as the inverse, recovery [18, 22, 8], and composition of schema mappings [21, 6].

For a complete and coherent introduction to the application of the chase procedure in data exchange, we first present the notion of universal models and its relation with the chase algorithm. Need to mention that the notion *universal models* [16] was introduced as a generalization of *universal solutions* [19] in data exchange. This first part will be followed by a short review of the data-exchange problem and the link between universal models and query answering in data exchange. In the final part of this section we will review the query answering problem in case there are no universal models.

5.1 Universal models

Beside the data-exchange problem, universal models play an important role in many other database problems as: testing for conjunctive query containment under functional and inclusion dependencies [31], data integration [33], and query answering over ontologies [14].

▶ **Definition 63.** [16] Given an instance I and Σ a set of dependencies, a finite instance J is said to be a *model* for I and Σ if $J \models \Sigma$, and $I \to J$.

▶ **Example 64.** Consider $I = \{R(a,b), R(b,c)\}$ and $\Sigma = \{R(x,y), R(y,z) \to R(x,z)\}$. The instance $J = \{R(a,b), R(b,c), R(a,c)\}$ is a model of I and Σ, so is instance $J_1 = J \cup \{R(a,X)\}$, with X a labeled null from Null. On the other hand, $J_2 = \{R(a,b), R(a,c)\}$ is not a model of I and Σ, even if $J_2 \models \Sigma$, since there is no homomorphism from I into J_2.

The conclusion of this example is that, in general, there may be an infinite number of models of I and Σ. Still, some of these models are more general than the others in the sense that they have a homomorphism into all the other models. Such models are called, of course, universal models.

▶ **Definition 65.** [16] A finite instance U is said to be a *weak universal model* of I and Σ if U is a model of I and Σ, and if for any finite model J of I and Σ, it is that $U \to J$. If $U \to J$. Also for all infinite models J of I and Σ, then U is said to be a *strong universal model* or simply a *universal model*.

▶ **Example 66.** Considering the instance I and the dependency Σ from Example 64, it is clear that both instances J and J_1 are strong universal models. Moreover, the model $J_3 = \{R(a,b), R(b,c), R(a,c), R(a,a)\}$ is neither a strong nor weak universal model as there does not exist a homomorphism from J_3 to model J.

▶ **Theorem 67.** [19, 16] *Let I be an instance and Σ a set of tgds and egds. Then any finite instance returned by the standard-chase algorithm is a universal model of I and Σ.*

Intuitively, the theorem says that whenever the standard chase terminates and it does not fail, it gives a universal model of I and Σ. From this theorem, it follows that if $chase^{\mathbf{std}}{}_{\Sigma}(I) \neq \perp$ then $chase^{\mathbf{std}}{}_{\Sigma}(I)$ is a universal model for I and Σ. In the finite case, the instance returned by the standard-chase algorithm is homomorphically equivalent with the finite result of any chase variations. It follows that for any of the previously presented chase variations, when they terminate and do not fail, they return a universal model.

▶ **Corollary 68.** *Let I be an instance, Σ a set of tgds and egds, and $* \in \{\mathbf{obl}, \mathbf{sobl}, \mathbf{core}\}$. If $chase^{*}{}_{\Sigma}(I) \neq \perp$, then $chase^{*}{}_{\Sigma}(I)$ is a universal model of I and Σ.*

This result ensures that the standard, oblivious, semi-oblivious and core chase are sound in finding universal models. Naturally the following question raises: *Are these algorithms also complete in finding universal models?* The following example shows that the standard, oblivious and semi-oblivious-chase algorithms are not complete.

▶ **Example 69.** Let us consider the same instance $I = \{R(a,b)\}$ and set $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned}\xi_1 &= R(x,y) \rightarrow \exists z\ R(y,z), \text{ and}\\ \xi_2 &= R(x,y), R(y,z) \rightarrow R(y,y).\end{aligned}$$

It is easy to see that there is no terminating branch for the standard chase for Σ and I. Similarly, the oblivious and semi-oblivious algorithms with the same input will not terminate. On the other hand, there exists universal model $J = \{R(a,b), R(b,b)\}$ of I and Σ. Thus the standard chase is not complete in finding universal models.

The result below shows that the core chase is complete in finding universal models.

▶ **Theorem 70.** [16] *Let I be an instance and Σ a set of tgds and egds. Then there exists a universal model of I and Σ iff the core-chase algorithm terminates and does not fail on input I and Σ.*

We know from the definition of the universal models that all universal models are also weak universal models. The following example shows that the converse does not hold.

▶ **Example 71.** [16] Let us consider instance $I = \{T(a)\}$ and $\Sigma = \{\xi_1, \xi_2, \xi_3\}$, where:

$$\begin{aligned}\xi_1 &= T(x) \rightarrow \exists y, z\ E(y,z);\\ \xi_2 &= E(x,y) \rightarrow \exists z\ E(y,z); \text{ and}\\ \xi_3 &= E(x,y), E(y,z) \rightarrow E(x,z).\end{aligned}$$

Consider the relation E to contain the edges of a graph. Clearly all models have an infinite walk. From this it follows that every finite model has a cycle in the corresponding graph. From this and ξ_3, it also follows that any finite model has a self loop. Besides, the instance $J = \{T(a), E(X,X)\}$ is a model of I and Σ containing a self loop. Consequently, J is a weak universal model of I and Σ. On the other hand, the transitive closure of an infinite path also satisfies Σ, however no finite instance with cycle has a homomorphism into it. This means that J is not a strong universal model of I and Σ.

Deutsch et al. [16] showed that it is undecidable to test if an instance U is a strong (weak) universal model for a given instance I and Σ a set of tgds. Even more, they demonstrated that there is no complete chase based procedures for finding weak universal models.

5.2 Data exchange

Data exchange is an old database problem that only recently earned more formal treatment. More precisely, it is the problem of transforming data structured under a source schema to data structured under a different target schema. Formally, a data-exchange setting is a quadruple $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where $\mathbf{S}$ represents the source schema, $\mathbf{T}$ represents the target schema, Σ_{st} is a set of constraints representing the relationship between the source and target schema, and Σ_t represents a set of constraints over the target schema. Given a data-exchange setting $(\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and the instance I over the source schema $\mathbf{S}$, the data-exchange problem is to find instances J over the target schema $\mathbf{T}$, such that $I \cup J$ is a model for I and $\Sigma_{st} \cup \Sigma_t$. An instance J with the previous properties is called a *solution* to the data-exchange problem, or simply a solution. This problem was first formalized by Fagin et al. in [19]. Most of the data-exchange problems consider Σ_{st} to be a set of tgds and Σ_t to be a set of tgds and egds. From now on, if not mentioned otherwise, we assume that the data-exchange settings are of this format.

As there is an infinite number of solutions to the data-exchange problem, a natural question raises: *Which solution or finite set of solutions should be materialized on the target?* There is no simple answer to this question as there may be different representations of the target depending on the semantics of the queries used over the target instance. The semantics considered in this subsection, also most prominent in the literature, is the certain answer semantics for union of conjunctive queries (UCQ) over the target instance. This can be formalized by the following definition:

▶ **Definition 72.** Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, let I be a source instance and Q a query in UCQ over $\mathbf{T}$. The certain answer of Q for I and σ is defined as

$$cert_\sigma(Q, I) =^{def} \bigcap_{J,\ I \cup J \models \Sigma_{st} \cup \Sigma_t} Q(J).$$

Fagin et al. [19] showed that the *universal solution* is a good candidate to be materialized in data-exchange problem under the certain UCQ answer semantics. Where the universal solution for a data-exchange setting $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ and the instance I is a universal model for I and $\Sigma_{st} \cup \Sigma_t$ restricted to schema $\mathbf{T}$. We need to mention that Fagin et al. considered as solutions only finite instances which is the more important case in practice. This means that all results specified in Subsection 5.1 also hold for universal solutions. In particular, it means that the universal solution can be computed by the chase algorithms and that it is undecidable if the universal solution exists for a given data-exchange setting and a given source instance. Marnette [38] showed that it is undecidable to test if the oblivious chase will terminate for a given data-exchange setting for all input instances. This result can be enhanced to all chase variations, including core chase. Thus, it is undecidable to test if, for a given data-exchange setting for all the input instances, there exists a universal solution.

In [19], Fagin et al. described a sufficient condition for the universal solution to not exist, as the following theorem shows it:

▶ **Theorem 73.** [19] *Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data-exchange setting and I a source instance such that there is a failing branch for the standard chase with input I and $\Sigma_{st} \cup \Sigma_t$. Then there is no universal solution for I and σ.*

In data exchange we may also have the case when there exists a solution but there is no universal solution. Let us consider the next example:

▶ **Example 74.** Consider the following data-exchange setting:

$$\sigma = (\{S\}, \{R\}, \{S(x,y) \to R(x,y)\}, \{R(x,y) \to \exists z\ R(y,z)\})$$

and the source instance $I = \{S(a,b)\}$. Clearly there is no universal solution for this setting, but there exists solution $J = \{R(a,b), R(b,b)\}$.

As shown by Kolaitis et al. in [32], it is undecidable to check for a given instance I and a data-exchange setting σ, if there exists a solution for I and σ.

Before presenting the computation of the certain answer for a data-exchange setting using a universal model, we need to introduce the notion of naïve evaluation. Let I be an instance, possible with null values, and Q be a query. The $Q_{\text{naïve}}(I)$ is defined by evaluating Q on I and by treating each null as a new distinct constants, and then by eliminating from the result all the tuples with nulls.

▶ **Theorem 75.** [19] *Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$ be a data-exchange setting and I an instance over the source instance that does not contain nulls such that there exists a universal solution J for I and σ. Then, $cert_\sigma(Q, I) = Q_{naïve}(J)$ for any $Q \in$* UCQ.

In [35] Libkin showed that UCQ is the largest class of queries with the property that the certain answers may be computed using the naïve evaluation. We conclude this subsection by reiterating the idea that within the infinite set of universal solutions there exists a universal solution which is minimal in size. Such universal solution is called core and, as noted in [20], it is unique up to variable renaming. Hence, in case there exists a universal solution, the core chase will terminate and return the core.

5.3 Data exchange beyond universal solutions

For the data-exchange setting and the source instance presented in Example 74 we know that there is no universal solution. On the other hand, when considering the query $Q(x) \leftarrow \exists y\ R(x,y)$, the certain answers is the set of tuples $\{(a), (b)\}$. In [12], Cali, Gottlob and Kifer investigate the problem of conjunctive query answering when the universal solution is not guaranteed to exist. For this, the authors unravel two classes of tgds , namely *guarded tuple generating dependencies* (gtgd) and *weakly guarded tuple generating dependencies* (wgtgd), for which the problem of conjunctive query evaluation is decidable. Intuitively, a tgd is guarded if its body contains an atom called *guard* which covers all the variables in the body. Clearly LAV tgds are gtgds. A set of tgds is weakly guarded, if for each tgd, its body contains one atom which covers all the variables that appear in the affected position, that is, predicate positions that may contain new labeled nulls generated during the chase process.

▶ **Example 76.** Let us consider the following dependencies:

$$\begin{aligned} \xi_1 &= S(x), R(x,y) \to \exists z\ R(y,z), \text{ and} \\ \xi_2 &= R(x,z), R(z,y) \to R(y,x). \end{aligned}$$

In ξ_1, the atom $R(x,y)$ covers all the variables in the body, meaning that it is a gtgd. Clearly, ξ_2 is not gtgd as there is no atom to cover all variables from the body. The affected position in the set $\{\xi_1, \xi_2\}$ is $(R, 2)$, that is we may introduce new labeled nulls during the chase process only in the second position of the predicate R. As in ξ_2, the atom $R(z,y)$ covers both variables that appear in affected position in ξ_2. It follows that ξ_2 is a wgtgd.

Cali et al. [12] give complexity bounds for the conjunctive query answering problem, that is: *Does a tuple t belong to the certain answer?* The complexity bounds discovered are the following: (1) for a fixed set gtgds the conjunctive query answering problem is NP-complete; (2) for atomic queries the problem becomes polynomial; (3) in case the fixed dependencies are wgtgds, the conjunctive query answering problem becomes EXPTIME-complete. Need to mention here that in [29] Hernich showed that if the data-exchange setting contains only guarded tgds, it is decidable if for the given setting and a given instance there exists a universal solution.

In the certain answer semantics for UCQ queries over the target schema a universal solution is enough to compute certain answer for any UCQ query. Therefore another question comes up naturally: *Is this semantics also a good model for general queries?* As shown in [4] and [34], this semantics is not suitable for general queries, as it may give unintuitive answers even for simple copying data-exchange settings.

▶ **Example 77.** Let us consider a data-exchange setting $\sigma = (\{R\}, \{R'\}, \Sigma_{st}, \emptyset)$, where Σ_{st} simply copies the source into target: $R(x, y) \to R'(x, y)$. Consider the source instance $I = \{R(a, b)\}$ and the query over the target schema $Q(x, y) \leftarrow R'(x, y) \wedge \neg R'(x, x)$. As one of the solution is the instance $J = \{R'(a, b), R'(a, a)\}$, it follows that $cert_\sigma(Q, I) = \emptyset$. Now, when applying the same query on the source instance (by replacing relation name R' with R), it returns the set of tuples $\{(a, b)\}$. Clearly this is not the expected behavior as the target instance is supposed to be a copy of the source instance.

To avoid such cases, Fagin et al. [20] proposed a new semantics for the certain answers to existential queries. Where existential queries $Q(\bar{x})$ is a formula of the form $\exists \bar{y}\ \varphi(\bar{x}, \bar{y})$, where φ is a *safe* quantifier-free formula. Under this semantics, instead of evaluating the query on all solutions, the query is evaluated on universal solutions only.

▶ **Definition 78.** Let $\sigma = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, let I be a source instance and Q a query over the schema $\mathbf{T}$. The *u-certain* answer of Q for I and σ is defined as

$$u\text{-}cert_\sigma(Q, I) =^{def} \bigcap \{Q(J)\ :\ J \text{ universal solution for } I \text{ and } \sigma\}.$$

Clearly $cert_\sigma(Q, I) \subseteq u\text{-}cert_\sigma(Q, I)$, for any data-exchange setting σ, instance I and query Q. Also, as shown in [20], $cert_\sigma(Q, I) = u\text{-}cert_\sigma(Q, I)$ whenever $Q \in \mathsf{UCQ}$. The u-certain semantics is shown [20] to be adequate for existential queries. Even more, it is proved that in case J is a universal solution for data-exchange setting σ and instance I, and Q is an existential query, then the answer under u-certain semantics can be computed as: $u\text{-}cert_\sigma(Q, I) = Q_{\text{naïve}}(core(J))$. Returning to the previous example, the core universal solution is $J = \{R(a, b)\}$, hence the certain answer to query Q will be the expected set of tuples $\{(a, b)\}$.

Later on new closed world semantics was proposed in order to deal with general queries for the data-exchange problem [34, 30, 28, 24]. Libkin [34] considered data-exchange settings without target dependencies and computed CWA-solutions which are used afterwards to compute certain answers for FO queries. Hernich and Schweikardt [30] introduced a new chase based algorithm, called the α-chase, to compute CWA-solutions when the data-exchange setting also contains target dependencies. In [24] Grahne and O. introduce a chase algorithm on conditional tables in order to strongly represent a closed world semantics called the constructible solutions. A similar chase process for conditional tables that considers only source to target dependencies was also introduced in [7].

6 Chase extensions

The chase algorithms presented in the previous sections considered only tgds and egds as constraints. In this section we will describe extensions of the chase algorithms needed in order to deal with *negation disjunctive embedded dependencies* (NDED). As we will see, the chase procedure on NDEDs helps finding universal solution sets which are used afterwards in computing certain answers to more general queries such as $\mathsf{UCQ}^{\neg,\neq}$. Disjunctive dependencies are also investigated by Marnette and Geerts in [40].

Before introducing the chase process for NDEDs, we need to extend the universal solution notion to *universal solution set*. Given two instances I and J, we write $I \dashrightarrow J$ if there exists an embedding from I to J. Let $\mathcal{I}$, $\mathcal{J}$ be two sets of instances, we write $\mathcal{I} \dashrightarrow \mathcal{J}$ if for all $J \in \mathcal{J}$ there exists $I \in \mathcal{I}$ such that $I \dashrightarrow J$.

▶ **Definition 79.** [16] A set $\mathcal{I}$ of finite instances is an *emb-universal model set* for a set of instances $\mathcal{J}$ if it satisfies the following conditions:

1. $\mathcal{I} \dashrightarrow \mathcal{J}$.
2. $\mathcal{I} \subseteq \mathcal{J}$.
3. $\mathcal{I}$ is finite.
4. there is no $\mathcal{I}' \subset \mathcal{I}$ such that $\mathcal{I}' \dashrightarrow \mathcal{J}$.

Let us first review the extended chase step for disjunctive embedded dependencies. A disjunctive embedded dependency (DED) [15] is a constraint of the form:

$$\xi: \ \forall \bar{x}\, \alpha(\bar{x}) \rightarrow \bigvee_{1 \leq i \leq n} \exists \bar{z}_i\, \beta_i(\bar{x}_i, \bar{z}_i)$$

where, $\bar{x}_i \subseteq \bar{x}$, for every $1 \leq i \leq n$. Formulae α and each β_i are conjunctions of relational symbols and equality atoms. For each $1 \leq i \leq n$, let us denote by ξ_i the dependency $\forall \bar{x}\, \alpha(\bar{x}) \rightarrow \exists \bar{z}_i\, \beta_i(\bar{x}_i, \bar{z}_i)$. The extended chase step on DED is defined as follows [16]. Let I be an instance and a homomorphism h such that $h(\alpha(\bar{x})) \subseteq I$. If $I \xrightarrow{(\xi_i,h)} \perp$, for all $1 \leq i \leq n$, then we say that the extended chase step on I with (ξ, h) *failed*, and it is denoted as $I \xrightarrow{(\xi,h)} \perp$. Otherwise, let $\mathcal{J}$ be the set containing all instances J_i, such that $I \xrightarrow{(\xi_i,h)} J_i$. If $\mathcal{J}$ is empty, it is said that the extended chase step on I with (ξ, h) is *not applicable*. For convenience we write this as $I \xrightarrow{(\xi_i,h)} I$. Finally, if $\mathcal{J} \neq \emptyset$, then $\mathcal{J}$ is said to be obtained from I in one extended chase step with (ξ, h) and denoted as $I \xrightarrow{(\xi,h)} \mathcal{J}$

▶ **Example 80.** Consider the following DED:

$$\xi \ = \ R(x,y), R(y,z) \rightarrow R(x,z) \vee x = y \vee \exists v\, R(v,z).$$

Let $I = \{R(a,b), R(b,c)\}$. Let $h = \{x/a, y/b, z/c\}$ be the homomorphism that maps the body of ξ to I. The three disjuncts from the head of ξ give the following dependencies:

$$\begin{aligned} \xi_1 &= R(x,y), R(y,z) \rightarrow R(x,z); \\ \xi_2 &= R(x,y), R(y,z) \rightarrow x = y; \text{ and} \\ \xi_3 &= R(x,y), R(y,z) \rightarrow \exists v\, R(v,z). \end{aligned}$$

For these dependencies, we have $I \xrightarrow{(\xi_1,h)} J$, where $J = I \cup \{R(a,c)\}$, $I \xrightarrow{(\xi_2,h)} \perp$ and $I \models \xi_3$. Thus $I \xrightarrow{(\xi,h)} \mathcal{J}$, where $\mathcal{J} = \{J\}$.

A dependency ξ of the form $\alpha(\bar{x}) \rightarrow \perp$, where α is a conjunction of atoms, is called *denial constraint* or *falsehood* . If for an instance I there exists a homomorphism h, such that $h(\alpha(\bar{x})) \subseteq I$, then it is said that the extended chase *failed* on I with (ξ, h) and it is denoted by $I \xrightarrow{(\xi,h)} \perp$.

If we add inequalities to DEDs, we obtain DED$^{\neq}s$ [15]. Let Σ be a set of DED$^{\neq}s$ over the schema $\mathbf{R}$. The set of dependencies Σ is replaced by $\Sigma^{\neq}$, in which each inequality of the from $x \neq y$ from Σ is replaced by the atom $N(x, y)$, where N is a new predicate which does not appear in Σ. Also in $\Sigma^{\neq}$ are added the following dependencies: $\rightarrow x = y \vee N(x, y)$, and $x = y \wedge N(x, y) \rightarrow \perp$. It may be noticed that in the new schema, $\Sigma^{\neq}$ contains one extra predicate compared to the schema of Σ and also it contains two new dependencies.

Finally, by adding to DED$^{\neq}s$ negation we obtain NDEDs. Let Σ be a set of NDEDs over the schema $\mathbf{R}$. By $\Sigma^{\neq,\neg}$ is denoted the set of dependencies $\Sigma^{\neq}$ in which each negated literal of the form $\neg R(\bar{x})$ is replaced by a new literal $\hat{R}(\bar{x})$, and also for each predicate $R \in \mathbf{R}$ the following two dependencies are added in $\Sigma^{\neq,\neg}$: $R(\bar{x}) \vee \hat{R}(\bar{x})$, and $R(\bar{x}) \wedge \hat{R}(\bar{x}) \rightarrow \perp$. It can be noted that for any set Σ of NDED, $\Sigma^{\neq,\neg}$ is a set of DED.

▶ **Example 81.** Consider the following set of dependencies $\Sigma = \{\xi_1, \xi_2\}$, where:

$$\begin{aligned} \xi_1 &= R(x,y) \rightarrow x \neq y, \text{ and} \\ \xi_2 &= R(x,y), S(x) \rightarrow \neg S(y). \end{aligned}$$

The corresponding $\Sigma^{\neq,\neg}$ will contain the dependencies:

$$\begin{aligned} \xi_1 &= R(x,y) \rightarrow N(x,y); \\ \xi_2 &= R(x,y), S(x) \rightarrow \hat{S}(y); \\ \xi_3 &= x = y \vee N(x,y); \\ \xi_4 &= x = y \wedge N(x,y) \rightarrow \perp; \\ \xi_3 &= R(x,y) \vee \hat{R}(x,y); \\ \xi_4 &= R(x,y) \wedge \hat{R}(x,y) \rightarrow \perp; \\ \xi_5 &= S(x) \vee \hat{S}(x); \text{ and} \\ \xi_6 &= S(x) \wedge \hat{S}(x) \rightarrow \perp. \end{aligned}$$

Using the previous notations we are now ready to present the *extended-core-chase* algorithm introduced by Deutsch, Nash and Remmel in [16] which has as input an instance I and a set Σ of NDED.

```
EXTENDED-CORE-CHASE(I,Σ)
1 $\mathcal{L}_0 = \{I\}$; $i := 0$;
2 Compute in parallel for each instance $J \in \mathcal{L}_i$ the set $\mathcal{K}_J$
  where $K \in \mathcal{K}_J$ iff $J \xrightarrow{(\xi,h)} K$ for some $\xi \in \Sigma^{\neq,\neg}$ and homomorphism $h$
3 $\mathcal{L}' = \bigcup_{J \in \mathcal{L}_i} \bigcup_{K \in \mathcal{K}_J} \{core(K)\}$
4 compute $\mathcal{L}_{i+1}$ by removing from $\mathcal{L}'$ all $K$ such that $\exists L \in \mathcal{L}'$, $L \rightarrow K$; i = i +1;
5 if $\mathcal{L}_i = \mathcal{L}_{i-1}$
6     then return the set of instances from $\mathcal{L}_i$ restricted to the schema of $I$
7     else goto 2
```

▶ **Example 82.** Consider $\Sigma = \{T(x) \rightarrow R(x)\}$ over the schema $\{R, S, T\}$ and consider the instance $I = \{T(a)\}$ over the same schema. With this input, the value of $\mathcal{L}_1$ after executing step 4 is $\mathcal{L}_1 = \{\{T(a), R(a), S(a)\}, \{T(a), R(a), \hat{S}(a)\}\}$, thus the algorithm will return the set $\{\{T(a), R(a), S(a)\}, \{T(a), R(a)\}\}$.

Let us denote by $\Sigma(I)$ the set of all models for I and Σ. The following theorem, due to [16], ensures that the returned set of instances, if it terminates, is an emb-universal model set for the set of all models of I and Σ.

▶ **Theorem 83.** [16] *Let Σ be a set of NDEDs over the schema $\mathbf{R}$ and let I be an instance over the same schema, such that the extended-core-chase algorithm terminates with the input I and Σ returning the set of instances $\mathcal{L}$. Then $\mathcal{L}$ is an emb-universal model set for $\Sigma(I)$.*

As shown in [16], emb-universal model sets can be used to compute the certain answers to $\mathsf{UCQ}^{\neq,\neg}$.

▶ **Theorem 84.** [16] *Let $\mathcal{U}$ be a emb-universal model set for $\Sigma(I)$, and let Q be a $\mathsf{UCQ}^{\neq,\neg}$ query. Then $cert_{\Sigma}(Q, I) = \bigcap_{J \in \mathcal{U}} Q(J)$.*

Let us consider the dependencies and the instance from Example 82 and also consider the query $Q(x) \leftarrow R(x) \wedge \neg S(x)$. When computing query Q against the emb-universal model set from Example 82, $cert_{\Sigma}(Q, I) = \emptyset$. The previous result does not hold for general FO queries. For this consider the boolean query $Q' \leftarrow (\forall x\ S(x) \rightarrow R(x))$. In this case $Q'(J)$ is **true** for all instance J from the emb-universal model set. On the other hand, the instance $J = \{T(a), R(a), S(b)\}$ is a model for I and $Q(J) =$ **false**, that is $cert_{\Sigma}(Q, I) =$ **false**. In order to cope with general FO queries in data exchange, several closed world semantics have been proposed [34, 30, 28, 24].

7 Conclusion

This chapter was intended to be a review of the chase based algorithms and also to highlight their use in data exchange. One of the main issues with the chase algorithms is the termination problem, that is:

- *Is there a branch for which the algorithm terminates for a given input I and Σ?*
- *Does the chase algorithm terminate on all branches for a given I and Σ?*

As presented, both these problems are undecidable in general. We also saw that the problem of testing if the core chase terminates for all input instances is undecidable in general. The undecidability result holds for the standard chase as well, in case we allow at least one denial constraint. Testing if the standard chase terminates for a given set of tgds on all instances remains however an open problem. Note that this problem is not the same as testing if there exists a universal solution for a given data-exchange setting with all input instances that is known to be an RE-complete problem [39].

Section 4 was dedicated to presenting large decidable classes of tgds for which it is known that the standard chase algorithm terminates on all branches for all input instances. As shown, all these classes actually ensure the termination for the "less expensive" (complexity based) semi-oblivious-chase algorithm, making this chase variation a better choice when dealing with sets of dependencies from those classes.

In case the chase based algorithm terminates, the instance computed is guaranteed to be homomorphic equivalent to any instance computed by any other chase variations. This property of the chase algorithms plays an important role in data exchange, especially in choosing the right instance on the target which should be materialized. Under the certain answers semantics for UCQ queries, the finite instances returned from any of the chase algorithms presented in Section 3 are good candidates to be materialized on the target. These instances, which are universal solutions, can be used together with the naïve evaluation to obtain the certain answers to any UCQ query over the target schema. In case a universal solution exists, the certain answers computation for UCQ queries is polynomial. In [12] it is shown that for some special classes of tgds, even if the universal solution is not guaranteed to exist, we may compute the certain answers to conjunctive queries. In these cases the complexity of computing the certain answers may grow as high as EXPTIME-hard.

To the best of our knowledge, the only chase based algorithm known to be complete in finding universal solutions for the data-exchange problem is the core chase. However, the core chase is the most expensive, complexity wise. This is because at each step it involves finding all the active triggers as well as computing the core of the produced instance. As shown in [20], the core-identification problem (i.e. given instances I and J, *Is I the core of J?*) is DP-complete. This leaves us with the open question if there exist other, less complex, chase based algorithms which are complete in finding universal solutions.

In this chapter we only focused on the cases where the chase algorithms terminate. This is mainly because in data exchange the infinite chase is not so important. If one is interested in the infinite chase, a good starting point would be [12].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 Foto N. Afrati and Phokion G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.

3 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.

4 Marcelo Arenas, Pablo Barceló, Ronald Fagin, and Leonid Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

5 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

6 Marcelo Arenas, Ronald Fagin, and Alan Nash. Composition with target constraints. In *ICDT*, pages 129–142, 2010.

7 Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data exchange beyond complete data. In *PODS*, pages 83–94, 2011.

8 Marcelo Arenas, Jorge Pérez, and Cristian Riveros. The recovery of a schema mapping: bringing exchanged data back. In *PODS*, pages 13–22, 2008.

9 Renée J. Miller Ariel Fuxman, Phokion G. Kolaitis and Wang Chiew Tan. Peer data exchange. In *PODS*, pages 160–171, 2005.

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

11 Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.

12 Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.

13 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. Datalog$^{\pm}$: a unified approach to ontologies and integrity constraints. In *ICDT*, pages 14–30, 2009.

14 Andrea Calì, Georg Gottlob, and Thomas Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *PODS*, pages 77–86, 2009.

15 Alin Deutsch. Fol modeling of integrity constraints (dependencies). In *Encyclopedia of Database Systems*, pages 1155–1161, 2009.

16 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

17 Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

18 Ronald Fagin. Inverting schema mappings. In *PODS*, pages 50–59, 2006.

19 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

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

21 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. In *PODS*, pages 83–94, 2004.

22 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Quasi-inverses of schema mappings. In *PODS*, pages 123–132, 2007.

23 Gösta Grahne and Adrian Onet. Data correspondence, exchange and repair. In *ICDT*, pages 219–230, 2010.

24 Gösta Grahne and Adrian Onet. Closed world chasing. In *LID*, pages 7–14, 2011.

25 Gösta Grahne and Adrian Onet. On conditional chase termination. In *AMW*, 2011.

26 Gösta Grahne and Adrian Onet. Anatomy of the chase. In *to appear*, 2013.

27 Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.

28 André Hernich. Answering non-monotonic queries in relational data exchange. In *ICDT*, pages 143–154, 2010.

29 André Hernich. Computing universal models under guarded tgds. In *ICDT*, pages 222–235, 2012.

30 André Hernich and Nicole Schweikardt. Cwa-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.

31 David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

32 Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The complexity of data exchange. In *PODS*, pages 30–39, 2006.

33 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

34 Leonid Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

35 Leonid Libkin. Incomplete information and certain answers in general data models. In *PODS*, pages 59–70, 2011.

36 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.

37 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies (abstract). In *SIGMOD Conference*, page 152, 1979.

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

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

40 Bruno Marnette and Floris Geerts. Static analysis of schema-mappings ensuring oblivious termination. In *ICDT*, pages 183–195, 2010.

41 Michael Meier, Michael Schmidt, and Georg Lausen. On chase termination beyond stratification. *PVLDB*, 2(1):970–981, 2009.

42 Michael Meier, Michael Schmidt, Fang Wei, and Georg Lausen. Semantic query optimization in the presence of types. In *PODS*, pages 111–122, 2010.

43 Alberto O. Mendelzon. Database states and their tableaux. In *XP2 Workshop on Relational Database Theory*, 1981.

44 Adrian Onet. *The chase procedure and its applications.* PhD thesis, Concordia University, 2012.

45 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

46 V Rutenberg. Complexity of generalized graph coloring. In *Proceedings of the 12th symposium on Mathematical foundations of computer science 1986*, pages 537–581, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

47 Francesca Spezzano and Sergio Greco. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.

48 Larry J. Stockmeyer. The polynomial-time hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

49 Moshe Y. Vardi. Inferring multivalued dependencies from functional and join dependencies. *Acta Inf.*, 19:305–324, 1983.

Algorithms for Core Computation in Data Exchange [*]

Vadim Savenkov

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

Abstract

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

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

Keywords and phrases data integration, data exchange, core computation

Digital Object Identifier 10.4230/DFU.Vol5.10452.39

1 Introduction

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

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

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

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

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

Target dependencies are given by the two tgds:

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

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

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

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

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

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

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

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

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

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

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

Table 1 Development of Algorithms for Core Computation.

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

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

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

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

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

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

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

2 Preliminaries

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

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

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

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

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

instead of

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3 Complexity of core computation

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

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

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

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

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

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

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

Input: An instance J
Output: The core of J

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

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

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

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

3.1 Parameterized complexity

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Core computation as a post-processing

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

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

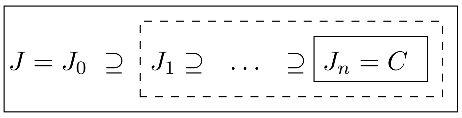


Figure 1 Recursive core approximation.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1 No target constraints

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

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

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

4.2 Target egds

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

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

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

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

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

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

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

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

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

4.3 Target tgds

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

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

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

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

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

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

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

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

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

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

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

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

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

Procedure Extend

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

4.3.1 Core computation for skolemized mappings

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

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

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

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

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

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

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

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

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

4.4 Target tgds and egds

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

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

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

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

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

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

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

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

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

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

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

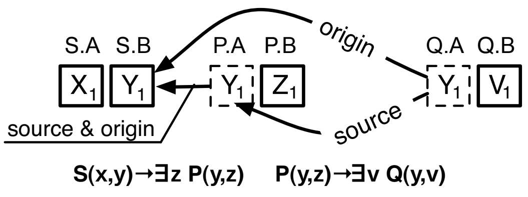


Figure 2 Parent relation over facts.

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

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

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

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

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

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

5 Direct core computation

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

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

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

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

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

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

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

5.1 No target dependencies

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

5.1.1 Core schema mappings

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.1.2 Laconic schema mappings

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Procedure CONVERTTOLACONIC

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

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

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

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

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

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

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

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

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

5.1.3 Discussion

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

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

5.2 Complexity and expressiveness

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

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

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

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

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

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

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

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

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

5.3 Target constraints

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Procedure Spicy-FD

Input: Schema mapping $\mathcal{M} = \mathbf{S}, \mathbf{T}, \Sigma_{st} \cup \Sigma_t$ where Σ_t as set of FDs
Output: Mappings $(\mathcal{R}_F, \mathcal{R}_C)$.
/ The core universal solution for I can be found as chase$(I \cup chase(I, \mathcal{R}_F), \mathcal{R}_C)$ */*

```
(1)  Generate $\Sigma_{st}^{ovl}$ and Set $\mathcal{R} := \textsc{Skolemize}\left(\textsc{AddNeg}(\Sigma_{st} \cup \Sigma_{st}^{ovl})\right)$ or $fail$
(2)  Let $\mathbf{F}$ be the schema $\{F_\epsilon \mid \epsilon\colon R\langle i_1 \dots i_m, j\rangle \in \Sigma_t\}$, $F_\epsilon$ fresh symbol of arity $m+1$
(3)  Let $\mathcal{R}_F = \emptyset$
(4)  for each $\phi(\vec{x}) \to \psi(\vec{x})$ in $\mathcal{R}$, $\epsilon\colon \langle i_1 \dots i_m, j\rangle \in \Sigma_t$ and $R(t_1, \dots t_n)$ in $\psi$
(5)      Set $\mathcal{R}_F := \mathcal{R}_F \cup \{\phi(\vec{x}) \to F_\epsilon(t_{i_1}, \dots, t_{i_m}, t_m)\}$
     /* Eliminate rigid ∃-variables */
(6)  Set $\Sigma_{st}^F := \Sigma_{st} \cup \Sigma_{st}^{ovl}$
(7)  while fixpoint is reached do
(8)      for each $\tau\colon \phi(\vec{x}) \to \exists y, \vec{z}\ \psi(\vec{x}, y, \vec{z})$ in $\Sigma'_{st}$ and $\epsilon : R\langle i_1, \dots i_m \to j\rangle \in \Sigma_t$
(9)          and each atom $R(t_1, \dots t_n)$ in $\psi$ such that $t_j = y$ and $\{t_{i_1}, \dots, t_{i_m}\} \in \vec{x}$
(10)             Replace $\tau$ in $\Sigma_{st}^F$ by $\forall\vec{x}\forall y\, (F(t_{i_1}, \dots t_{i_m}, y) \wedge \phi(\vec{x}) \to \exists z\ \psi(\vec{x}, y, \vec{z}))$
     /* Apply algorithms from Section 5.1.1 or Section 5.1.2 */
(11) Convert $\Sigma_{st}^F$ into a Laconic or Core mapping $\mathcal{R}_C$
(12) return $(\mathcal{R}_F, \mathcal{R}_C)$
```

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

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

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

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

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

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

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

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

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

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

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

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

6 Performance

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

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

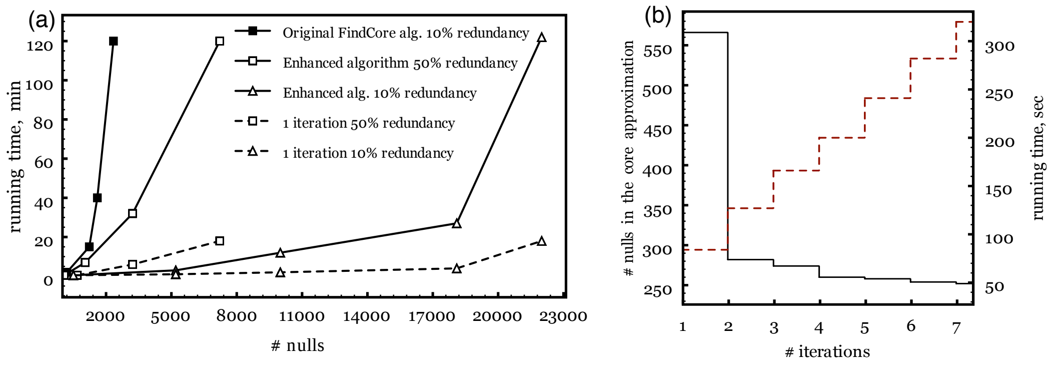
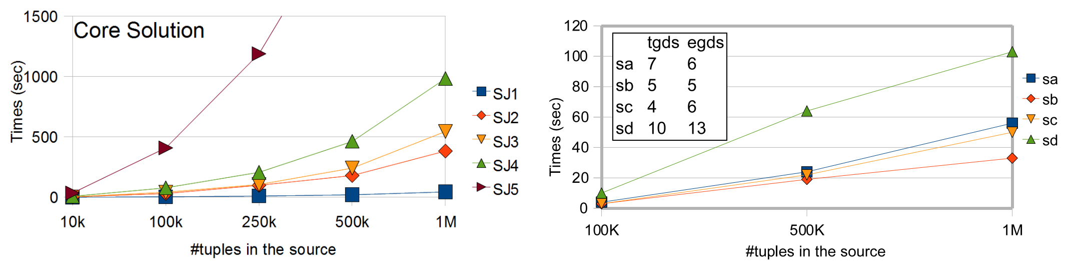


Figure 3 Performance (a) and the progress (b) of core computation [20].



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

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

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

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

7 Conclusion

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The Inverse of a Schema Mapping

Jorge Pérez

Department of Computer Science, Universidad de Chile
Blanco Encalada 2120, Santiago, Chile
`jperez@dcc.uchile.cl`

Abstract

The inversion of schema mappings has been identified as one of the fundamental operators for the development of a general framework for data exchange, data integration, and more generally, for metadata management. Given a mapping $\mathcal{M}$ from a schema $\mathbf{S}$ to a schema $\mathbf{T}$, *an inverse* of $\mathcal{M}$ is a new mapping that describes the *reverse* relationship from $\mathbf{T}$ to $\mathbf{S}$, and that is *semantically consistent* with the relationship previously established by $\mathcal{M}$. In practical scenarios, the inversion of a schema mapping can have several applications. For example, in a data exchange context, if a mapping $\mathcal{M}$ is used to exchange data from a source to a target schema, an inverse of $\mathcal{M}$ can be used to exchange the data back to the source, thus *reversing* the application of $\mathcal{M}$.

The formalization of a clear semantics for the inverse operator has proved to be a very difficult task. In fact, during the last years, several alternative notions of inversion for schema mappings have been proposed in the literature. This chapter provides a survey on the different formalizations for the inverse operator and the main theoretical and practical results obtained so far. In particular, we present and compare the main proposals for inverting schema mappings that have been considered in the literature. For each one of them we present their formal semantics and characterizations of their existence. We also present algorithms to compute inverses and study the language needed to express such inverses.

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

Keywords and phrases Schema mappings, data exchange, inverse

Digital Object Identifier 10.4230/DFU.Vol5.10452.69

1 Introduction

A schema mapping is a specification that describes how data from a source schema is to be mapped to a target schema. Schema mappings are of fundamental importance in data management today. In particular, they have proved to be the essential building block for several data-interoperability tasks such as data exchange, data integration and peer data management.

In recent years, the research on the schema mapping area has mainly focused on performing data-interoperability tasks using schema mappings. However, as Bernstein [12] pointed out, many information-system problems involve not only the design and integration of complex application artifacts, but also their subsequent manipulation. Notice that the creation of a schema mapping may imply considerable work by an expert who needs to know the semantics of the schema components. Only an expert can establish a meaningful high-level correspondence between those components. Thus, a schema mapping reflects the knowledge of the expert about the relationship between the schemas. This knowledge could, in principle, be reused beyond the interoperability tasks for which the mapping was initially created. Driven by these considerations, Bernstein [12] proposed a general framework for managing and reusing schema mappings.

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 69–95
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

In Bernstein's framework [12], schema mappings are first class citizens, and high-level algebraic operators are used to manipulate and reuse them. One of the most fundamental operators in schema mapping management is the *inversion* of schema mappings. Given a mapping $\mathcal{M}$ from a schema $\mathbf{A}$ to a schema $\mathbf{B}$, *an inverse* of $\mathcal{M}$ is a new mapping that describes the *reverse* relationship from $\mathbf{B}$ to $\mathbf{A}$, and that is *semantically consistent* with the relationship previously established by $\mathcal{M}$. Notice that this is a very general idea of what an inverse of a schema mapping should be. In fact, even the formalization of a clear semantics for the inverse operator has proved to be a very difficult task [16, 17, 20, 21, 10, 11, 22, 6]. This chapter provides a survey on the different formalizations in the literature for the inverse operator on schema mappings and the study of the theoretical problems that arise. Before going into the details of these works, let us give a bit more intuition on how the inverse of schema mappings can be useful in practice.

In practical scenarios, the inversion of schema mappings can have several applications. In a data exchange context [18], if a mapping $\mathcal{M}$ is used to exchange data from a source to a target schema, an inverse of $\mathcal{M}$ can be used to exchange the data back to the source, thus *reversing* the application of $\mathcal{M}$. As a second application, consider a peer data management system (PDMS) [14, 26]. In a PDMS, a peer can act as a data source, a mediator, or both, and the system relates peers by establishing mappings between the peers' schemas. Mappings between peers are usually *directional*, and are used to reformulate queries. For example, if there is a mapping $\mathcal{M}$ from peer P_1 to peer P_2 and a query over P_2, a PDMS can use $\mathcal{M}$ to reformulate the query by using P_1 as a source. Hence, an inverse of $\mathcal{M}$ would allow the PDMS to reformulate a query over P_1 in terms of P_2, thus considering this time P_2 as a source. Another application is schema evolution, where the inverse together with the *composition* play a crucial role [13, 23]. Consider a mapping $\mathcal{M}$ between schemas $\mathbf{A}$ and $\mathbf{B}$, and assume that schema $\mathbf{A}$ evolves into a schema $\mathbf{A}'$. This evolution can be expressed as a mapping $\mathcal{M}'$ between $\mathbf{A}$ and $\mathbf{A}'$. Thus, the relationship between the new schema $\mathbf{A}'$ and schema $\mathbf{B}$ can be intuitively obtained by inverting mapping $\mathcal{M}'$ and then composing the result with mapping $\mathcal{M}$.

As we have mentioned before, in the study of the inverse operator, one of the key issues is to provide a *good* semantics for this operator, which turned out to be a difficult problem. After defining a semantics, some of the important questions that need to be answered are:

Existence For which classes of mappings is the inverse guaranteed to exist?
Expressiveness What is the mapping language needed to specify an inverse?
Algorithmic How can we effectively construct an inverse?

In this chapter, we present and compare the main proposals for inverting schema mappings that have been considered in the literature, and for each one of them we present the main results regarding these three issues. In Section 2 we present the notion of inverse proposed by Fagin [16], that we call here *Fagin-inverse*[1], which is the first formal notion of inverse proposed in the literature. In Section 3 we present the notion of *quasi-inverse* [20, 21] which is obtained by relaxing the notion of Fagin-inverse. Section 4 presents the notions of *recovery* and *maximum recovery* [10, 11] which were proposed as alternative notions for inverting schema mappings. In Section 5 we present procedures to compute inverses and discuss expressiveness issues, more importantly, the issue of what the language needed to

[1] Fagin [17] named his notion just as *inverse* of a schema mapping. Since in this chapter we are introducing several different semantics for the *inverse* operator, we reserve the term *inverse* to refer to this operator in general, and use the name *Fagin-inverse* for the notion proposed by Fagin [17].

express inverses is. In Section 6 we present a relaxation of the notions of recovery and maximum recovery based on certain answers, which gives alternative definitions of inverses when one is focused on retrieving information with a particular class of queries. In Section 7 we report on extensions to the previous notions that deal with incomplete information in source instances. Conclusions are presented in Section 8. We begin by giving a bit of general notation for the chapter.

Preliminary notions and notation

In the study of the inverse operator, we use a general notion of schema mapping (or just mapping in our context). We assume that a mapping from schema $\mathbf{S}$ to schema $\mathbf{T}$ is simply a set of pairs (I, J) where I is an instance of $\mathbf{S}$ and J is an instance of $\mathbf{T}$. As usual in data exchange, given a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ and an instance I of $\mathbf{S}$, we denote by $\text{Sol}_{\mathcal{M}}(I)$ the set of *solutions* for I under $\mathcal{M}$, that is $\text{Sol}_{\mathcal{M}}(I) = \{J \mid (I, J) \in \mathcal{M}\}$.

Notice that a mapping in this general setting is just a binary relation, and thus one can define some general operators over mappings that inherits from binary relations. One such particular operator that plays a crucial role in this chapter is mapping *composition*. Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{R}$. The composition of $\mathcal{M}$ and $\mathcal{M}'$, denoted by $\mathcal{M} \circ \mathcal{M}'$, is defined as the composition of binary relations, that is $\mathcal{M} \circ \mathcal{M}' = \{(I, K) \mid \text{there exists } J \text{ such that } (I, J) \in \mathcal{M} \text{ and } (J, K) \in \mathcal{M}'\}$ [33, 19].

We usually specify mappings by using logical languages. One particular language of special interest is the language of *source-to-target tuple-generating dependencies* (st-tgds) [18]. An st-tgd from $\mathbf{S}$ to $\mathbf{T}$ is a First-Order formula of the form

$$\forall \bar{x} \forall \bar{y} \big(\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z})\big) \tag{1}$$

in which $\bar{x}$, $\bar{y}$ and $\bar{z}$ are tuple of variables, $\varphi(\bar{x}, \bar{y})$ is a conjunction of relational atoms over $\mathbf{S}$ (mentioning all the variables $\bar{x}$ and $\bar{y}$), and $\psi(\bar{x}, \bar{z})$ is a conjunction of relational atoms over $\mathbf{T}$ (mentioning all the variables $\bar{x}$ and $\bar{z}$). The left-hand side of the implication in formula (1) is called the *premise*, and the right-had side the *conclusion* of the st-tgd. For simplicity, we omit the universal quantifiers when writing st-tgds. That is, we just write $\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z})$ for an st-tgd of the form (1). Given an st-tgd σ of the form $\varphi(\bar{x}, \bar{y}) \rightarrow \exists \bar{z}\ \psi(\bar{x}, \bar{z})$ from $\mathbf{S}$ to $\mathbf{T}$, and a pair (I, J) with I an instance of $\mathbf{S}$ and J an instance of $\mathbf{T}$, we say that (I, J) *satisfies* σ if for every pair of tuples $\bar{a}$, $\bar{b}$ such that I satisfies $\varphi(\bar{a}, \bar{b})$, there exists a tuple $\bar{c}$ such that J satisfies $\psi(\bar{a}, \bar{c})$.

We say that a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ is *specified* by a set Σ of st-tgds, if for every pair of instances I of $\mathbf{S}$ and J of $\mathbf{T}$ we have that $(I, J) \in \mathcal{M}$ if and only if (I, J) satisfies every st-tgd in Σ. We consider two types of values when defining mappings, constant and null values, and we assume the existence of a special predicate $\mathbf{C}(\cdot)$ to differentiate them. In particular, $\mathbf{C}(u)$ holds if and only if u is a constant value. As is usual in the data exchange context [18], when defining a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds we assume that source instances (instances of $\mathbf{S}$) contain only constant values, while target instances (instances of $\mathbf{T}$) may contain constant and null values. Notice that an inverse of $\mathcal{M}$ is a mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$, and thus $\mathcal{M}'$ has constant and null values in its source schema (schema $\mathbf{T}$), while only constants in its target schema (schema $\mathbf{S}$). In Section 7 we drop this assumption, and study inversion of mappings that may contain constant and nulls in source and target instances.

2 Fagin-inverse

The first notion of inverse in the literature was proposed by Fagin [16]. This notion is based on the algebraic intuition that a mapping composed with its inverse should be equal to the *identity*. Since we can unambiguously define the composition of two schema mappings (based on the composition of binary relations), we only needed to define a notion of identity for schema mappings.

Fagin was specially interested in defining an inverse for mappings specified by st-tgds thus, he defined an intuitive identity in terms of st-tgds as follows. Let $\mathbf{S}$ be a schema, and $\hat{\mathbf{S}} = \{\hat{R} \mid R \in \mathbf{S}\}$, that is, $\hat{\mathbf{S}}$ is a copy of $\mathbf{S}$. The set of *copying* st-tgds over $\mathbf{S}$ is defined as

$$\Sigma_{\mathbf{S}\text{-copy}} = \{\ R(x_1, \ldots, x_k) \rightarrow \hat{R}(x_1, \ldots, x_k) \mid R \text{ is a } k\text{-ary relation symbol in } \mathbf{S}\}.$$

The idea is that $\Sigma_{\mathbf{S}\text{-copy}}$ essentially copies every source relation from the source to the target. Notice that we need to use $\hat{\mathbf{S}} = \{\hat{R} \mid R \in \mathbf{S}\}$ in the definition of $\Sigma_{\mathbf{S}\text{-copy}}$ and not simply $\mathbf{S}$ since, otherwise, the semantics of the tgds would be trivial. Consider now the mapping $\mathcal{M}_{\mathbf{S}\text{-copy}}$ from $\mathbf{S}$ to $\hat{\mathbf{S}}$, which is specified by $\Sigma_{\mathbf{S}\text{-copy}}$. Given the definition of mapping $\mathcal{M}_{\mathbf{S}\text{-copy}}$, it its a natural identity in our context, and thus, the notion of Fagin-inverse is formulated as follows.

▶ **Definition 1** ([16]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\hat{\mathbf{S}}$. Then $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ if $\mathcal{M} \circ \mathcal{M}' = \mathcal{M}_{\mathbf{S}\text{-copy}}$.

Notice that in the above definition, a Fagin-inverse of a mapping from $\mathbf{S}$ to $\mathbf{T}$ is not a mapping from $\mathbf{T}$ to $\mathbf{S}$ but from $\mathbf{T}$ to $\hat{\mathbf{S}}$. This is because we were specially interested in defining the identity mapping with a set of st-tgds. But we can reformulate the above notion to use only schema $\mathbf{S}$. Let I, J be instances of $\mathbf{S}$ and $\hat{J}$ be a copy of J over schema $\hat{\mathbf{S}}$. Then we have that $(I, \hat{J}) \in \mathcal{M}_{\mathbf{S}\text{-copy}}$ if and only if $I \subseteq J$. Thus we can redefine the identity mapping as a mapping $\overline{\text{Id}}_{\mathbf{S}}$ from $\mathbf{S}$ to $\mathbf{S}$ given by

$$\overline{\text{Id}}_{\mathbf{S}} = \{(I, J) \mid I, J \text{ are instances of } \mathbf{S} \text{ and } I \subseteq J\}.$$

With this new identity mapping we can reformulate the notion of Fagin-inverse as follows.

▶ **Definition 2** ([16]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Then $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ if $\mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$.

In the rest of the chapter we use Definition 2 for the notion of Fagin-inverse. Moreover, if mapping $\mathcal{M}$ has a Fagin-inverse, then we say that $\mathcal{M}$ is *Fagin-invertible*. It is important to notice that $\overline{\text{Id}}_{\mathbf{S}}$ is not exactly the identity relation over instances of schema $\mathbf{S}$. In [17], Fagin formally justified the use of $\overline{\text{Id}}_{\mathbf{S}}$ as the identity when inverting mappings specified by st-tgds, instead of the more natural $\text{Id}_{\mathbf{S}} = \{(I, I) \mid I \text{ is an instance of } \mathbf{S}\}$. As Fagin proved, no composition of st-tgds can be equal to $\text{Id}_{\mathbf{S}}$ (see Proposition 5.2 in [17]).

▶ **Example 3.** Let $\mathbf{S}$ be a source schema composed of a binary relation $A(\cdot, \cdot)$, and $\mathbf{T}$ a target schema with a ternary relation $B(\cdot, \cdot, \cdot)$. Consider now the mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ specified by the st-tgd $A(x, y) \rightarrow B(x, x, y)$. Then the mapping $\mathcal{M}_1$ specified by $B(x, u, y) \rightarrow A(x, y)$ is a Fagin-inverse of $\mathcal{M}$.

To see why $\mathcal{M}_1$ is a Fagin-inverse of $\mathcal{M}$, assume that $(I, J) \in \mathcal{M} \circ \mathcal{M}_1$. We know that there exists an instance K of schema $\mathbf{T}$ such that $(I, K) \in \mathcal{M}$ and $(K, J) \in \mathcal{M}_1$. Then, if $A(a, b)$ is a fact in I with a and b arbitrary values, then $B(a, a, b)$ is a fact in K, which, by the definition of $\mathcal{M}_1$ implies that $A(a, b)$ is a fact in J. We have shown that every fact in I is also a fact in J, and thus $I \subseteq J$. On the other hand, assume that $I \subseteq J$, and consider the instance L of $\mathbf{T}$ such that $B(a, a, b)$ is a fact in L if and only if $A(a, b)$ is a fact in I. Then it is straightforward that $(I, L) \in \mathcal{M}$ and $(L, J) \in \mathcal{M}_1$, which implies that $(I, J) \in \mathcal{M} \circ \mathcal{M}_1$. We have shown that $(I, J) \in \mathcal{M} \circ \mathcal{M}_1$ if and only if $I \subseteq J$, thus implying that $\mathcal{M} \circ \mathcal{M}_1 = \overline{\text{Id}}_{\mathbf{S}}$.

Consider now the mapping $\mathcal{M}_2$ specified by $B(u, x, y) \rightarrow A(x, y)$, and the mapping $\mathcal{M}_3$ specified by $B(x, x, y) \rightarrow A(x, y)$. Then both $\mathcal{M}_2$ and $\mathcal{M}_3$ are also Fagin-inverses of $\mathcal{M}$. This example shows that Fagin-inverses need not to be unique up to logical equivalence [17]. ◀

In the above example it was quite simple to obtain a Fagin-inverse. In fact, as mapping $\mathcal{M}_3$ shows, just *reversing the arrows* in the definition of $\mathcal{M}$ generates a Fagin-inverse. In the following examples, we show that Fagin-inverses are not always as easy to construct. In particular, Example 4 shows that reversing the arrows does not always produce a Fagin-inverse, and Example 5 shows that in some cases we need inequalities when specifying Fagin-inverses.

▶ **Example 4** ([17]). Consider a schema $\mathbf{S}$ with two unary relations $A(\cdot)$ and $B(\cdot)$, and a schema $\mathbf{T}$ with three unary relations $S(\cdot)$, $T(\cdot)$, and $U(\cdot)$. Let $\mathcal{M}$ be the mapping specified by the following set of st-tgds

$$\begin{array}{lcl} A(x) & \rightarrow & S(x) \\ A(x) & \rightarrow & T(x) \\ B(x) & \rightarrow & U(x) \\ B(x) & \rightarrow & T(x) \end{array}$$

Let $\mathcal{M}'$ be the mapping obtained from the specification of $\mathcal{M}$ by just reversing the arrows, that is, $\mathcal{M}'$ is specified by the set of tgds $S(x) \rightarrow A(x)$, $T(x) \rightarrow A(x)$, $U(x) \rightarrow B(x)$ and $T(x) \rightarrow B(x)$. It is easy to see that $\mathcal{M}'$ is not a Fagin-inverse of $\mathcal{M}$. Consider the instance $I = \{A(1)\}$. Then for every $K \in \text{Sol}_{\mathcal{M}}(I)$ we have that $T(1)$ is a fact in K. Thus, given that $T(x) \rightarrow B(x)$ is in the specification of $\mathcal{M}'$, we have that for every $J \in \text{Sol}_{\mathcal{M}'}(K)$ it holds that $B(1)$ is a fact in J. This implies that for every J such that $(I, J) \in \mathcal{M} \circ \mathcal{M}'$ it holds that $B(1)$ is a fact in J, and thus, $(I, I) \notin \mathcal{M} \circ \mathcal{M}'$, which shows that $\mathcal{M} \circ \mathcal{M}' \neq \overline{\text{Id}}_{\mathbf{S}}$.

The problem in this case is that dependencies $A(x) \rightarrow T(x)$ and $B(x) \rightarrow T(x)$ are somehow *mixing* the data of relations A and B in target relation T. Thus, to obtain a Fagin-inverse of $\mathcal{M}$, we cannot use T to recover the data of relations A and B. In fact, a Fagin-inverse of $\mathcal{M}$ can be constructed by using only target relations S and U as follows:

$$\begin{array}{lcl} S(x) & \rightarrow & A(x) \\ U(x) & \rightarrow & B(x) \end{array}$$

It can be easily shown that the mapping defined by the above dependencies is a Fagin-inverse of $\mathcal{M}$. ◀

▶ **Example 5** ([21]). Consider a schema $\mathbf{S}$ with a binary relation $A(\cdot, \cdot)$ and a unary relation $B(\cdot)$, and a schema $\mathbf{T}$ with a binary relation $S(\cdot, \cdot)$ and two unary relations $T(\cdot)$ and $U(\cdot)$. Let $\mathcal{M}$ be the mapping specified by the following set of st-tgds

$$\begin{array}{rcl} A(x, y) & \rightarrow & S(x, y) \\ B(x) & \rightarrow & S(x, x) \\ B(x) & \rightarrow & T(x) \\ A(x, x) & \rightarrow & U(x) \end{array}$$

Notice that in this case the mapping is translating tuples of the form $A(a, a)$ and $B(a)$ into the same target relation S, thus, as in Example 4, mapping $\mathcal{M}$ is somehow mixing the source information when translating it to the target. In this case we can solve this issue by using inequalities to specify a Fagin-inverse of $\mathcal{M}$. Consider the mapping $\mathcal{M}'$ specified by the following dependencies

$$\begin{aligned} S(x,y) \wedge x \neq y &\;\rightarrow\; A(x,y) \\ T(x) &\;\rightarrow\; B(x) \\ U(x) &\;\rightarrow\; A(x,x) \end{aligned}$$

Then, it can be shown that $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$. In fact, Fagin et al. showed [20, 21] that inequalities are strictly needed to specify Fagin-inverses of mappings given by st-tgds (we make this statement precise in Section 5). ◀

On the existence of Fagin-inverses

As we explained in the introduction, a first important question to answer for every definition of inverse of schema mappings, is for which class of mappings the inverse is guaranteed to exist. As we show next, there are several mappings specified by st-tgds that do not admit Fagin-inverses.

Consider the following mappings specified by st-tgds (in every case, source and target schemas are implicit in the dependencies).

$$\begin{array}{lrcl} \mathcal{M}_1: & A(x,y) & \rightarrow & S(x) \\ \\ \mathcal{M}_2: & A(x,y) & \rightarrow & S(x) \wedge T(y) \\ \\ \mathcal{M}_3: & A(x) & \rightarrow & S(x) \\ & B(x) & \rightarrow & S(x) \end{array} \tag{2}$$

As pointed out by Fagin [17], Fagin-invertibility for a mapping intuitively coincide with *no loss of information.* Thus, considering this intuition, none of the above mappings should be Fagin-invertible. For instance, mapping $\mathcal{M}_1$ is only transferring the first component of relation A from source to target, and thus, we are losing the second component when transferring the source data. In the case of $\mathcal{M}_2$, although it is actually transferring both components of A from source to target, these components are being stored in independent relations in the target thus loosing the relationships that they had in the source. For $\mathcal{M}_3$ the problem is a little bit different. In this case all the data in both A and B is being transferred but, since all the information is stored in the same relation in the target, it is impossible to reconstruct the initial source instances.

The question is how to formally prove that the above mappings have no Fagin-inverses? To answer this, Fagin [16] proposed a very simple condition that a mapping specified by st-tgds needs to satisfy in order to have a Fagin-inverse. This property is called the *unique-solutions property* and is formalized as follows.

▶ **Definition 6** ([16]). A mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ satisfies the *unique-solutions property* if for every pair of instances I_1, I_2 of $\mathbf{S}$, it holds that $\mathrm{Sol}_{\mathcal{M}}(I_1) = \mathrm{Sol}_{\mathcal{M}}(I_2)$ implies $I_1 = I_2$.

▶ **Theorem 7** ([16]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds. If $\mathcal{M}$ has a Fagin-inverse then $\mathcal{M}$ satisfies the unique-solutions property.*

The proof of the theorem is very simple. Assume that we have a mapping $\mathcal{M}$ and that $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$. Now let I_1 and I_2 be instances such that $\mathrm{Sol}_{\mathcal{M}}(I_1) = \mathrm{Sol}_{\mathcal{M}}(I_2)$. Since $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$, we know that $\mathcal{M} \circ \mathcal{M}' = \overline{\mathrm{Id}}_{\mathbf{S}}$ and thus $(I_2, I_2) \in \mathcal{M} \circ \mathcal{M}'$.

This implies that there exists an instance K such that $(I_2, K) \in \mathcal{M}$ and $(K, I_2) \in \mathcal{M}'$. Now, since $\text{Sol}_{\mathcal{M}}(I_1) = \text{Sol}_{\mathcal{M}}(I_2)$ and $K \in \text{Sol}_{\mathcal{M}}(I_2)$, we have that $(I_1, K) \in \mathcal{M}$ and then $(I_1, I_2) \in \mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$ which implies that $I_1 \subseteq I_2$. With a symmetric argument we can show that $I_2 \subseteq I_1$ and thus $I_1 = I_2$.

With this tool we can formally prove that the mappings in (2) have no Fagin-inverses. For the case of $\mathcal{M}_1$, consider instances $I_1 = \{A(1,2)\}$ and $I_2 = \{A(1,3)\}$. The instances are different but $\text{Sol}_{\mathcal{M}_1}(I_1) = \text{Sol}_{\mathcal{M}_2}(I_2)$. For the case of $\mathcal{M}_2$ we can use instances $I_1 = \{A(1,2), A(3,4)\}$ and $I_2 = \{A(1,4), A(3,2)\}$ which satisfy that $\text{Sol}_{\mathcal{M}_2}(I_1) = \text{Sol}_{\mathcal{M}_2}(I_2)$. For the mapping $\mathcal{M}_3$ and the instances $I_1 = \{A(1)\}$ and $I_2 = \{B(1)\}$, we have that $\text{Sol}_{\mathcal{M}_3}(I_1) = \text{Sol}_{\mathcal{M}_3}(I_2)$. Thus neither $\mathcal{M}_1$ nor $\mathcal{M}_2$ nor $\mathcal{M}_3$ satisfy the unique-solutions property which implies that they have no Fagin-inverse.

The natural question at this point is whether the unique-solutions property is also a sufficient condition to test Fagin-invertibility for mappings specified by st-tgds. It can be shown that it is not [21][2]. Fortunately, Fagin et al. [20] introduced another property, called the *subset property*, which characterizes Fagin-invertibility for the case of mappings specified by st-tgds.

▶ **Definition 8** ([20]). A mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ satisfies the *subset property* if for every pair of instances I_1, I_2 of $\mathbf{S}$ we have that $\text{Sol}_{\mathcal{M}}(I_1) \subseteq \text{Sol}_{\mathcal{M}}(I_2)$ implies $I_2 \subseteq I_1$.

▶ **Theorem 9** ([20]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds. Then $\mathcal{M}$ has a Fagin-inverse if and only if $\mathcal{M}$ satisfies the subset property.*

We have shown that there are several mappings specified by st-tgds that have no Fagin-inverse, thus the question at this point is whether we can find some relaxed notions that can give natural and useful reverse mappings when Fagin-inverses do not exist. In the next two sections we introduce the notions of *quasi-inverse* [20, 21] and *maximum recovery* [10, 11] proposed to deal with this issue.

3 Quasi-inverse

As we have shown in the previous section, there are many simple mappings specified by st-tgds that do not possess Fagin-inverses. Nevertheless, in many cases there are very simple and natural ways of specifying useful reverse mappings. Thus, there is a need for a weaker notion of inverse to handle these cases. Towards solving this problem, Fagin et al. [20] proposed the notion of a quasi-inverse of a schema mapping.

Intuitively, the notion of quasi-inverse is obtained from the notion of Fagin-inverse by not differentiating between source instances that are equivalent for data-exchange purposes. Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and define the equivalence relation $\sim_{\mathcal{M}}$ between instances of $\mathbf{S}$ as follows: $I_1 \sim_{\mathcal{M}} I_2$ if and only if $\text{Sol}_{\mathcal{M}}(I_1) = \text{Sol}_{\mathcal{M}}(I_2)$. That is, I_1 and I_2 are considered equivalent if they have the same space of solutions under $\mathcal{M}$. For instance, for the mapping $\mathcal{M}$ specified by the st-tgds $A(x,y) \to S(x)$, and the instances $I_1 = \{A(1,2)\}$ and $I_2 = \{A(1,3)\}$, we have that $I_1 \sim_{\mathcal{M}} I_2$.

Informally, $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$ if the equation $\mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$ holds *modulo* $\sim_{\mathcal{M}}$. To make this statement precise, let us introduce some notation. Let D be a binary

[2] Fagin proved that for LAV mappings, that is, mappings specified by st-tgds in which only one atom is mentioned in the premises of dependencies, the unique-solutions property is necessary and sufficient to characterize Fagin-invertibility [17].

relation over instances of a source schema $\mathbf{S}$ (that is, a mapping from $\mathbf{S}$ to $\mathbf{S}$), and let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to a schema $\mathbf{T}$. Then we define the relation $D[\sim_{\mathcal{M}}]$ as follows:

$$D[\sim_{\mathcal{M}}] = \{(I_1, I_2) \mid \text{ there exists } I_1' \text{ and } I_2' \text{ such that } I_1 \sim_{\mathcal{M}} I_1', I_2 \sim_{\mathcal{M}} I_2' \text{ and } (I_1', I_2') \in D\}.$$

Now we can formally introduce the notion of quasi-inverse of a schema mapping.

▶ **Definition 10** ([20]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Then $\mathcal{M}'$ is a *quasi-inverse* of $\mathcal{M}$ if $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$.

▶ **Example 11.** Consider a source schema $\mathbf{S} = \{A(\cdot,\cdot)\}$ and a target schema $\mathbf{T} = \{S(\cdot)\}$, and let $\mathcal{M}$ be the mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by $A(x, y) \to S(x)$. We showed in the previous section that $\mathcal{M}$ has no Fagin-inverse. Consider now the mapping $\mathcal{M}'$ specified by $S(x) \to \exists u\, A(x, u)$. We now show that $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$. To see why this is the case, consider first the inclusion:

$$\overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}] \quad \subseteq \quad (\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}]. \tag{3}$$

If $(I_1, I_2) \in \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$, then there exist instances I_1', I_2' of $\mathbf{S}$ such that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ and $(I_1', I_2') \in \overline{\text{Id}}_{\mathbf{S}}$. Thus, we have that $I_1' \subseteq I_2'$. Let J_1' be an instance of $\mathbf{T}$ such that $S(a)$ is a fact in J_1' if and only if $A(a, b)$ is a fact in I_1' (for arbitrary values a and b). Then we have that $(I_1', J_1') \in \mathcal{M}$, and also that $(J_1', I_1') \in \mathcal{M}'$ by the definitions of $\mathcal{M}$ and $\mathcal{M}'$. Moreover, given that $I_1' \subseteq I_2'$ we have that (J_1', I_2') also satisfies the tgds defining $\mathcal{M}'$, and thus $(J_1', I_2') \in \mathcal{M}'$. Hence, we conclude that $(I_1', I_2') \in (\mathcal{M} \circ \mathcal{M}')$, which implies that $(I_1, I_2) \in (\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}]$ (since $I_1 \sim_{\mathcal{M}} I_1'$ and $I_2 \sim_{\mathcal{M}} I_2'$). Thus, we have shown that inclusion (3) holds, and it only remains to prove that the following inclusion holds:

$$(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] \quad \subseteq \quad \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]. \tag{4}$$

If $(I_1, I_2) \in (\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}]$, then there exist instances I_1', I_2' of $\mathbf{S}$ such that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ and $(I_1', I_2') \in (\mathcal{M} \circ \mathcal{M}')$. Thus, we have that there exists an instance K of $\mathbf{T}$ such that $(I_1', K) \in \mathcal{M}$ and $(K, I_2') \in \mathcal{M}'$. By the definitions of $\mathcal{M}$ and $\mathcal{M}'$ we conclude that for every fact $A(a, b)$ in I_1', there exists an element c such that $A(a, c)$ is a fact in I_2'. From this last property we conclude that the instance $I^\star = I_1' \cup I_2'$ is such that $I^\star \sim_{\mathcal{M}} I_2'$. Moreover, since $I_2 \sim_{\mathcal{M}} I_2'$ and $I_2' \sim_{\mathcal{M}} I^\star$, we have that $I_2 \sim_{\mathcal{M}} I^\star$. Notice that $I_1' \subseteq I^\star$, and thus we have that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I^\star$ and $(I_1', I^\star) \in \overline{\text{Id}}_{\mathbf{S}}$, which implies that $(I_1, I_2) \in \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$. Thus, we have shown that (4) holds, which proves that $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$. ◀

As the previous example shows, there are mappings that are not Fagin-invertible but have a quasi-inverse. This, plus the following result, show that the notion of quasi-inverse is a strict generalization of the notion of Fagin-inverse. In particular, the result shows that if a mapping has a Fagin-inverse, then the notions of Fagin-inverse and quasi-inverse coincide.

▶ **Theorem 12** ([20]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds, and assume that $\mathcal{M}$ has a Fagin-inverse. Then $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ if and only if $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$.*

It is not difficult to see why the theorem holds. Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and assume that $\mathcal{M}$ is specified by st-tgds and has a Fagin-inverse. If $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ then $\mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$, which implies that $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$, and thus $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$. Assume now that $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$, that is, $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$. Given that $\mathcal{M}$ has a Fagin-inverse, from Theorem 7 we know that $\mathcal{M}$ satisfies the unique-solutions property. Thus, we have that for every pair of instances

I_1, I_2 of $\mathbf{S}$, it holds that if $I_1 \sim_{\mathcal{M}} I_2$ (or equivalently $\text{Sol}_{\mathcal{M}}(I_1) = \text{Sol}_{\mathcal{M}}(I_2)$) then $I_1 = I_2$. Notice that this implies that $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = \mathcal{M} \circ \mathcal{M}'$ and $\overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}] = \overline{\text{Id}}_{\mathbf{S}}$. Thus, since $(\mathcal{M} \circ \mathcal{M}')[\sim_{\mathcal{M}}] = \overline{\text{Id}}_{\mathbf{S}}[\sim_{\mathcal{M}}]$ we obtain that $\mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$ which implies that $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$.

On the existence of quasi-inverses

Consider the mappings in (2) in the previous section. In Example 11 we showed that for mapping $\mathcal{M}_1$ specified by $A(x, y) \to S(x)$, the mapping specified by $S(x) \to \exists u\, A(x, u)$ is a quasi-inverse. Consider $\mathcal{M}_2$, which is specified by $A(x, y) \to S(x) \wedge T(y)$. In this case it can be proved that the mapping specified by $S(x) \wedge T(y) \to \exists u\, A(x, u) \wedge \exists v\, A(v, y)$ is a quasi-inverse of $\mathcal{M}_2$. Moreover, for the case of mapping $\mathcal{M}_3$ specified by $A(x) \to S(x)$ and $B(x) \to S(x)$, it can be proved that $S(x) \to A(x) \vee B(x)$ is a quasi-inverse. Although none of these mappings admit a Fagin-inverse, all of them admit a quasi-inverse. This gives rise to the interesting question of whether every mapping specified by sets of st-tgds has a quasi-inverse. Unfortunately, it was shown by Fagin et al. [20, 21] that the answer is negative. To formalize this result we next introduce a property that characterizes when a mapping specified by st-tgds has a quasi-inverse. This property is obtained by relaxing the subset-property that characterizes Fagin-inverses.

▶ **Definition 13** ([20]). A mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ satisfies the $(\sim_{\mathcal{M}})$-*subset property*, when for every pair I_1, I_2 of instances of $\mathbf{S}$, if $\text{Sol}_{\mathcal{M}}(I_1) \subseteq \text{Sol}_{\mathcal{M}}(I_2)$ then there exist instances I_1' and I_2' such that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ and $I_2' \subseteq I_1'$.

▶ **Theorem 14** ([20]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds. Then $\mathcal{M}$ has a quasi-inverse if and only if $\mathcal{M}$ satisfies the $(\sim_{\mathcal{M}})$-subset property.*

With the above tool we can show that there are mappings specified by st-tgds that do not have a quasi-inverse.

▶ **Example 15** ([20, 21]). Consider a source schema $\mathbf{S}$ consisting of a binary relation $A(\cdot, \cdot)$, a target schema $\mathbf{T}$ consisting of a binary relation $S(\cdot, \cdot)$ and a unary relation $T(\cdot)$, and the mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ specified by the st-tgd

$$A(x, z) \wedge A(z, y) \quad \to \quad S(x, y) \wedge T(z). \tag{5}$$

Fagin et al. showed [20, 21] that $\mathcal{M}$ does not satisfy the $(\sim_{\mathcal{M}})$-subset property, from which follows that $\mathcal{M}$ has no quasi-inverse. We next show why $\mathcal{M}$ does not satisfy the $(\sim_{\mathcal{M}})$-subset property.

Let I_1, I_2 be instances of $\mathbf{S}$ such that:

$$\begin{aligned} I_1 &= \{A(1,4), A(4,3), A(1,2), A(2,5), A(4,2)\} \\ I_2 &= \{A(1,2), A(2,3)\} \end{aligned}$$

Moreover, let J_1, J_2 be the following instances over $\mathbf{T}$:

$$\begin{aligned} J_1 &= \{S(1,3), S(1,2), S(1,5), S(4,5), T(2), T(4)\} \\ J_2 &= \{S(1,3), T(2)\} \end{aligned}$$

That is, J_1 and J_2 are the *canonical solutions* [18, 3] of I_1 and I_2, respectively. In this case, given the definition of $\mathcal{M}$, it is not difficult to see that they are also *minimal* and characterizes their space of solutions. For instance, we have that $K \in \text{Sol}_{\mathcal{M}}(I_1)$ if and only if $J_1 \subseteq K$. Similarly for I_2 we have that $K \in \text{Sol}_{\mathcal{M}}(I_2)$ if and only if $J_2 \subseteq K$. Thus, given that $J_2 \subseteq J_1$, we conclude that $\text{Sol}_{\mathcal{M}}(I_1) \subseteq \text{Sol}_{\mathcal{M}}(I_2)$. Next we show that there are no instances I_1', I_2' of **S** such that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ and $I_2' \subseteq I_1'$, which implies that $\mathcal{M}$ does not satisfy the $(\sim_{\mathcal{M}})$-subset property.

For the sake of contradiction, assume that there exist instances I_1', I_2' of **S** such that $I_1 \sim_{\mathcal{M}} I_1'$, $I_2 \sim_{\mathcal{M}} I_2'$ and $I_2' \subseteq I_1'$, and let J_1', J_2' be the canonical solutions for I_1' and I_2' under $\mathcal{M}$, respectively. That is

$$\begin{aligned} J_1' &= \{S(a,b) \mid \text{ there exists } c \text{ s.t. } A(a,c), A(c,b) \in I_1'\} \cup \\ &\quad\ \{T(c) \mid \text{ there exist } a, b \text{ s.t. } A(a,c), A(c,b) \in I_1'\}, \\ J_2' &= \{S(a,b) \mid \text{ there exists } c \text{ s.t. } A(a,c), A(c,b) \in I_2'\} \cup \\ &\quad\ \{T(c) \mid \text{ there exist } a, b \text{ s.t. } A(a,c), A(c,b) \in I_2'\}, \end{aligned}$$

Given that $I_2 \sim_{\mathcal{M}} I_2'$, we have that $\text{Sol}_{\mathcal{M}}(I_2) = \text{Sol}_{\mathcal{M}}(I_2')$ and, therefore, $J_2 = J_2'$ by the definition of $\mathcal{M}$. Thus, given that $S(1,3) \in J_2$, we have that $S(1,3) \in J_2'$ and, hence, there exists an element m such that $A(1,m), A(m,3) \in I_2'$. Notice that this implies that $T(m)$ should be a fact in J_2' and then since $J_2' = J_2$, we obtain that m must be equal to 2. Thus, we have that $A(1,2), A(2,3) \in I_2'$. Now given that $I_2' \subseteq I_1'$, we conclude that:

$$A(2,3) \in I_1'. \tag{6}$$

Given that $I_1 \sim_{\mathcal{M}} I_1'$, we have that $\text{Sol}_{\mathcal{M}}(I_1) = \text{Sol}_{\mathcal{M}}(I_1')$ and, therefore, $J_1 = J_1'$ by the definition of $\mathcal{M}$. Thus, given that $S(4,5) \in J_1$, we have that $S(4,5) \in J_1'$ and, hence, there exists an element n such that $A(4,n), A(n,5) \in I_1'$. Notice that this implies that $T(n)$ should be a fact in J_1' and then since $J_1' = J_1$, we obtain that n must be equal to either 2 or 4. We show that in both cases we obtain a contradiction. Assume that $n = 4$, then we have that $A(4,4) \in I_1'$ implying that $S(4,4) \in J_1'$ which leads to a contradiction since $J_1 = J_1'$ and $S(4,4) \notin J_1$. Assume that $n = 2$, then $A(4,2), A(2,5) \in I_1'$. But we know from (6) that $A(2,3) \in I_1'$ concluding that $S(4,3) \in J_1'$ (since $A(4,2) \in I_1'$), from which we obtain a contradiction since $J_1 = J_2'$ and $S(4,3) \notin J_1$. ◀

Although numerous non-Fagin-invertible schema mappings possess natural and useful quasi-inverses, the previous example shows that there still exist simple mappings specified by st-tgds that have no quasi-inverse. This leaves as an open problem the issue of finding a notion of inversion for st-tgds which ensures that every mapping in this class is invertible. This is the main motivation for the introduction of the notion of inversion discussed in the following section.

4 Recovery and Maximum Recovery

In this section we introduce the notions of recovery and maximum recovery proposed by Arenas et al. [10] as alternative notions for inverting mappings. As we show in this section, the notion of maximum recovery strictly generalizes the notion of Fagin-inverses, but has the desirable property that every mapping specified by st-tgds admits a maximum recovery, thus solving the open problem left by the notion of quasi-inverse.

Let us start by considering the mapping $\mathcal{M}$ in Example 15, that is, $\mathcal{M}$ is specified by $A(x,z) \wedge A(z,y) \rightarrow S(x,y) \wedge T(z)$. Notice that although mapping $\mathcal{M}$ does not have a quasi-inverse, there is a very natural reverse mapping in this case. Consider the mapping $\mathcal{M}'$ defined by the tgds

$$\begin{aligned} S(x,y) &\rightarrow \exists u \, \big(A(x,u) \wedge A(u,y)\big) \\ T(z) &\rightarrow \exists v \exists w \, \big(A(v,z) \wedge A(z,w)\big) \end{aligned}$$

$\mathcal{M}'$ is essentially doing its *best effort* to recover the data initially stored in the source schema. This is the main intuition behind the notions of recovery and maximum recovery. Intuitively, a recovery of $\mathcal{M}$ is a mapping that is capable of recovering *sound data* with respect to $\mathcal{M}$, and a maximum recovery of $\mathcal{M}$ is a mapping that is capable to recover *the maximum amount of sound data* with respect to $\mathcal{M}$. We next formalize both notions.

Let $\mathcal{M}$ be a mapping from a schema $\mathbf{S}$ to a schema $\mathbf{T}$, and $\overline{\text{Id}}_{\mathbf{S}}$ the *identity mapping* over $\mathbf{S}$, that is,

$$\overline{\text{Id}}_{\mathbf{S}} = \{(I, I) \mid I \text{ is an instance of } \mathbf{S}\}.$$

Notice the difference between $\overline{\text{Id}}_{\mathbf{S}}$ and $\text{Id}_{\mathbf{S}}$; mapping $\text{Id}_{\mathbf{S}}$ is the classical identity of binary relations. When trying to invert $\mathcal{M}$, the ideal would be to find a mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ such that, $\mathcal{M} \circ \mathcal{M}' = \overline{\text{Id}}_{\mathbf{S}}$. If such a mapping exists, we know that if we use $\mathcal{M}$ to exchange data, the application of $\mathcal{M}'$ gives as result exactly the initial source instance. Unfortunately, in most cases this ideal is impossible to reach. For example, it is impossible to obtain such an inverse if $\mathcal{M}$ is specified by a set of st-tgds [16]. The main problem with such an ideal definition of inverse is that, in general, no matter what $\mathcal{M}'$ we choose, we will have not one but many solutions for a source instance under $\mathcal{M} \circ \mathcal{M}'$.

If for a mapping $\mathcal{M}$, there is no mapping $\mathcal{M}_1$ such that $\mathcal{M} \circ \mathcal{M}_1 = \overline{\text{Id}}_{\mathbf{S}}$, at least we would like to find a schema mapping $\mathcal{M}_2$ that *does not forbid* the possibility of recovering the initial source data. That is, we would like that for every source instance I, the space of solutions for I under $\mathcal{M} \circ \mathcal{M}_2$ contains I itself. Such a schema mapping $\mathcal{M}_2$ is called a *recovery* of $\mathcal{M}$.

▶ **Definition 16** ([10])**.** Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Then $\mathcal{M}'$ is a *recovery* of $\mathcal{M}$ if $(I, I) \in \mathcal{M} \circ \mathcal{M}'$ for every instance I of $\mathbf{S}$.

▶ **Example 17.** Let $\mathbf{S} = \{A(\cdot,\cdot)\}$, and $\mathbf{T} = \{S(\cdot,\cdot), T(\cdot)\}$. Consider the mapping $\mathcal{M}$ in Example 15, that is $\mathcal{M}$ is a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by the following st-tgd:

$$A(x,z) \wedge A(z,y) \quad \rightarrow \quad S(x,y) \wedge T(z), \tag{7}$$

Let $\mathcal{M}_1$ be a mapping from $\mathbf{T}$ to $\mathbf{S}$ specified by tgd:

$$S(x,y) \quad \rightarrow \quad \exists u \left(A(x,u) \wedge A(u,y)\right).$$

It is straightforward to prove that $\mathcal{M}_1$ is a recovery of $\mathcal{M}$. Let I be an arbitrary instance of $\mathbf{S}$, and J the *canonical solution* [18] for I, that is,

$$\begin{aligned} J \quad = \quad & \{S(a,b) \mid \text{ there exists } c \text{ s.t. } A(a,c), A(c,b) \in I\} \ \cup \\ & \{T(c) \mid \text{ there exist } a, b \text{ s.t. } A(a,c), A(c,b) \in I\}. \end{aligned}$$

Then in this case we have that $(I, J) \in \mathcal{M}$ and $(J, I) \in \mathcal{M}_1$, from which we conclude that $(I, I) \in \mathcal{M} \circ \mathcal{M}_1$. This implies that $\mathcal{M}_1$ is a recovery of $\mathcal{M}$. Similarly, if $\mathcal{M}_2$ is a mapping from $\mathbf{T}$ to $\mathbf{S}$ specified by tgd:

$$T(z) \quad \rightarrow \quad \exists v \exists w \left(A(v,z) \wedge A(z,w)\right),$$

then we also have that $\mathcal{M}_2$ is a recovery of $\mathcal{M}$. On the other hand, if $\mathcal{M}_3$ is a mapping from $\mathbf{T}$ to $\mathbf{S}$ specified by tgd:

$$S(x,y) \wedge T(z) \quad \rightarrow \quad A(x,z) \wedge A(z,y), \tag{8}$$

then we have that $\mathcal{M}_3$ is not a recovery of $\mathcal{M}$. To see why this is the case, consider the instance $I = \{A(1,1), A(2,2)\}$. Next we show that $(I, I) \notin \mathcal{M} \circ \mathcal{M}_3$. ForL the sake of contradiction, assume that $(I, I) \in \mathcal{M} \circ \mathcal{M}_3$, and let K be an instance such that $(I, K) \in \mathcal{M}$ and $(K, I) \in \mathcal{M}_3$. Given that (I, K) satisfies st-tgd (7), we have that $S(1,1), S(2,2)$ and $T(1), T(2)$ are facts in K. But then given that (K, I) satisfies tgd (8), we conclude that $A(1,2)$, and $A(2,1)$ are facts in I, which is a contradiction. ◀

Being a recovery is a sound but mild requirement. Indeed, a schema mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ always has as recoveries, for example, mappings $\mathcal{M}_1 = \{(J, I) \mid J$ is an instance of $\mathbf{T}$ and I is an instance of $\mathbf{S}\}$, and $\mathcal{M}_2 = \mathcal{M}^{-1} = \{(J, I) \mid (I, J) \in \mathcal{M}\}$. If one has to choose between $\mathcal{M}_1$ and $\mathcal{M}_2$ as a recovery of $\mathcal{M}$, then one would probably choose $\mathcal{M}_2$ since the space of possible solutions for a source instance I under $\mathcal{M} \circ \mathcal{M}_2$ is smaller than under $\mathcal{M} \circ \mathcal{M}_1$. In fact, if there exists a mapping $\mathcal{M}_3$ such that $\mathcal{M} \circ \mathcal{M}_3 = \mathrm{Id}_{\mathbf{S}}$, then one would definitely prefer $\mathcal{M}_3$ over $\mathcal{M}_1$ and $\mathcal{M}_2$.

In general, if $\mathcal{M}'$ is a recovery of $\mathcal{M}$, then the smaller the space of solutions generated by $\mathcal{M} \circ \mathcal{M}'$, the more informative $\mathcal{M}'$ is about the initial source instances. This notion induces an *order* among recoveries. If $\mathcal{M}_1$ and $\mathcal{M}_2$ are recoveries of $\mathcal{M}$ and $\mathcal{M} \circ \mathcal{M}_2 \subseteq \mathcal{M} \circ \mathcal{M}_1$ then we say that $\mathcal{M}_2$ is *more(-or-equally) informative* than $\mathcal{M}_1$ as a recovery of $\mathcal{M}$. This naturally gives rise to the notion of maximum recovery. If for a mapping $\mathcal{M}$ there exists a recovery $\mathcal{M}'$ that is more informative than any other recovery of $\mathcal{M}$, then $\mathcal{M}'$ is the best option to bring exchanged data back, among all the recoveries. Intuitively, such a mapping $\mathcal{M}'$ recovers the maximum amount of sound information. Such a mapping $\mathcal{M}'$ is called a maximum recovery of $\mathcal{M}$.

▶ **Definition 18** ([10])**.** Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Then $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$ if:

1. $\mathcal{M}'$ is a recovery of $\mathcal{M}$, and
2. for every recovery $\mathcal{M}''$ of $\mathcal{M}$, it holds that $\mathcal{M} \circ \mathcal{M}' \subseteq \mathcal{M} \circ \mathcal{M}''$.

Notice that the definition of maximum recovery implies a quantification over all the possible recoveries of a mapping $\mathcal{M}$. Thus, the process of proving that a particular mapping is indeed a maximum recovery of $\mathcal{M}$ seems to be very a difficult task (compare it with the definitions of Fagin-inverse, quasi-inverse and recovery). Fortunately, Arenas et al. [11] provide a toolbox to deal with maximum recoveries. In particular, the following general characterization is useful to prove that a mapping is a maximum recovery of another mapping.

▶ **Theorem 19** ([11])**.** *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ and $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Then $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$ if and only if $\mathcal{M} = \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$.*

▶ **Example 20.** Let $\mathbf{S} = \{A(\cdot,\cdot)\}$, and $\mathbf{T} = \{S(\cdot,\cdot), T(\cdot)\}$. Consider again the mapping $\mathcal{M}$ in Example 15 specified by:

$$A(x,z) \wedge A(z,y) \quad \rightarrow \quad S(x,y) \wedge T(z),$$

and let $\mathcal{M}'$ be the mapping from $\mathbf{T}$ to $\mathbf{S}$ specified by the following tgds:

$$\begin{aligned} S(x,y) \quad &\rightarrow \quad \exists u \, \big(A(x,u) \wedge A(u,y)\big), \\ T(z) \quad &\rightarrow \quad \exists v \exists w \, \big(A(u,z) \wedge A(z,w)\big), \end{aligned}$$

Next we use Theorem 19 to show that $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$. Given that $\mathcal{M}'$ is a recovery of $\mathcal{M}$ (see Example 17), we have that $\mathcal{M} \subseteq \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$. Thus, by using Theorem 19, in order to show that $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$, we only need to show that $\mathcal{M} \circ \mathcal{M}' \circ \mathcal{M} \subseteq \mathcal{M}$.

Let $(I, J) \in \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$. To prove that $(I, J) \in \mathcal{M}$, we need to show that (I, J) satisfies the st-tgd that specifies $\mathcal{M}$. Let $A(a, b)$ and $A(b, c)$ be facts in I, with a, b, c arbitrary elements. Then we need to prove that $S(a, c), T(b) \in J$. To prove this, first notice that given that $(I, J) \in \mathcal{M} \circ \mathcal{M}' \circ \mathcal{M}$, there exist instances K of $\mathbf{T}$ and L of $\mathbf{S}$ such that $(I, K) \in \mathcal{M}$, $(K, L) \in \mathcal{M}'$ and $(L, J) \in \mathcal{M}$. Thus, given that $A(a, b), A(b, c) \in I$ and $(I, K) \in \mathcal{M}$, we conclude that $S(a, c), T(b) \in K$. Hence, from the definition of $\mathcal{M}'$ and the fact that $(K, L) \in \mathcal{M}'$, we conclude that there exist elements d, e and f such that

$$A(a, d), A(d, c), A(e, b), A(b, f) \quad \in \quad L.$$

Therefore, given that $(L, J) \in \mathcal{M}$, we conclude that $S(a, c), T(b) \in J$ which was to be shown. ◀

As for the case of the quasi-inverse, it can be shown that the notion of maximum recovery strictly generalizes the notion of Fagin-inverse for mappings specified by st-tgds.

▶ **Theorem 21** ([10]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds, and assume that $\mathcal{M}$ has a Fagin-inverse. Then, $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ if and only if $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$.*

The relationship between maximum recoveries and quasi-inverses is a little bit more complicated and is given in the following result.

▶ **Theorem 22** ([10]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds, and assume that $\mathcal{M}$ has a quasi-inverse.*

1. *If $\mathcal{M}'$ is a maximum-recovery of $\mathcal{M}$ then $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$.*
2. *If $\mathcal{M}'$ is a quasi-inverse and a recovery of $\mathcal{M}$, then $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$.*

On the existence of maximum recoveries

One of the main results regarding maximum recoveries is that they exist for every mapping specified by st-tgds [10]. To show this, we next introduce the notion of *witness solution* that can be used to characterize when a general mapping has a maximum recovery.

▶ **Definition 23** ([10]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ and I an instance of $\mathbf{S}$. Then an instance $J \in \text{Sol}_{\mathcal{M}}(I)$ is a *witness solution* for I under $\mathcal{M}$, if for every other instance I' such that $J \in \text{Sol}_{\mathcal{M}}(I')$ it holds that $\text{Sol}_{\mathcal{M}}(I) \subseteq \text{Sol}_{\mathcal{M}}(I')$.

Witness solutions are in a sense identifiers for spaces of solutions. In particular, if there are two instances I_1 and I_2 that share a witness solution, then $\text{Sol}_{\mathcal{M}}(I_1) = \text{Sol}_{\mathcal{M}}(I_2)$. Arenas et al. [10], proved the following general characterization of the existence of maximum recoveries.

▶ **Theorem 24** ([10]). *Let $\mathcal{M}$ be a general mapping from $\mathbf{S}$ to $\mathbf{T}$ (not necessarily specified by st-tgds). Then $\mathcal{M}$ has a maximum recovery if and only if every instance I of $\mathbf{S}$ has a witness solution under $\mathcal{M}$.*

It should be noticed that as opposed to the characterizations for the existence of Fagin-inverses and quasi-inverse shown in Theorems 9 and 14, respectively, the characterization for maximum recoveries can be applied to general mappings, not necessarily specified by st-tgds. We can now use Theorem 24 to show that mappings specified my st-tgds always have maximum recovery. For this we need to recall the notion of *universal solutions* in data exchange [18]. Universal solutions were introduced as desirable solutions for data exchange.

Essentially, a universal solution for an instance I under a mapping $\mathcal{M}$ is, in a precise sense, the most general solution for I and can be *embedded* in any other solution for I [18]. In particular, for mappings specified by st-tgds, universal solutions can be obtained by using the *chase* procedure (see Chapter 1, for a comprehensive study of the chase procedure). In our context, the two most important properties of universal solutions are stated in the following lemma.

▶ **Lemma 25** ([10, 18]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds.*
1. *If J is a universal solution for I under $\mathcal{M}$ then J is a witness solution for I under $\mathcal{M}$.*
2. *For every instance I of $\mathbf{S}$ there exists a universal solution for I under $\mathcal{M}$.*

Then from Theorem 24 and Lemma 25 we obtain the following.

▶ **Corollary 26** ([10]). *Every mapping $\mathcal{M}$ specified by st-tgds has a maximum recovery.*

5 Computing Inverses

Up to this point we have presented three alternative notions for inverting mappings. For every one of them we have discussed their formal definitions, characterizations and the existence problem. But we have not discussed the most important practical problem regarding inverses of schema mappings: how to compute an inverse. In this section we present a general algorithm that can be used to compute all the notions of inverses introduced so far. We also discuss expressiveness issues. In particular, what is the language needed to express these inverses which is directly related to the language used in the output of the algorithm.

5.1 An algorithm for inverting mappings

The algorithm presented in this section is based on *query rewriting* and thus we first introduce the necessary terminology and some preliminary results. A fundamental notion in this section is the notion of *certain answers*. Given a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$, a source instance I, and a query $Q_{\mathbf{T}}$ over $\mathbf{T}$, the set of *certain answers* of $Q_{\mathbf{T}}$ over I, denoted by $\text{certain}_{\mathcal{M}}(Q_{\mathbf{T}}, I)$ is the set

$$\text{certain}_{\mathcal{M}}(Q_{\mathbf{T}}, I) = \bigcap_{J \in \text{Sol}_{\mathcal{M}}(I)} Q_{\mathbf{T}}(J).$$

That is, a tuple $\bar{a}$ is a certain answer if $\bar{a} \in Q_{\mathbf{T}}(J)$ for every solution J of I. With the notion of certain answers we can define the notion of *source rewritability*. Given a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$, and a query $Q_{\mathbf{T}}$ over $\mathbf{T}$, we say that $Q_{\mathbf{S}}$ over $\mathbf{S}$ is a *source rewriting* of $Q_{\mathbf{T}}$ under $\mathcal{M}$ if for every instance I of $\mathbf{S}$ it holds that

$$Q_{\mathbf{S}}(I) = \text{certain}_{\mathcal{M}}(Q_{\mathbf{T}}, I).$$

That is, if $Q_{\mathbf{S}}$ is a source rewriting of $Q_{\mathbf{T}}$, in order to compute the certain answers of $Q_{\mathbf{T}}$ one only needs to compute $Q_{\mathbf{S}}(I)$.

The computation of a source rewriting of a conjunctive query is a basic step in the first algorithm presented in this section. This problem has been extensively studied in the database area [30, 31, 15, 1, 35] and, in particular, in the data integration context [24, 25, 29]. It can be shown that given a mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ specified by a set of st-tgds, and a conjunctive query $Q_{\mathbf{T}}$ over $\mathbf{T}$, then a rewriting of $Q_{\mathbf{T}}$ over the source can always be expressed as a union of conjunctive queries with equality predicates ($\mathrm{UCQ}^=$). As an example, consider a mapping given by the following tgds:

$$\begin{aligned} A(x,y) &\rightarrow S(x,y), \\ B(x) &\rightarrow S(x,x), \end{aligned}$$

and let $Q_{\mathbf{T}}$ be the target query given by formula $S(x,y)$. Then a rewriting of $Q_{\mathbf{T}}$ over the source is given by $A(x,y) \vee (B(x) \wedge x = y)$, which is a query in $\mathrm{UCQ}^=$. Notice that in this rewriting, we do need disjunction and the equality $x = y$. Moreover, it is known that source rewritings of conjunctive queries can be computed in exponential time. We formalize the above discussion in the following lemma.

▶ **Lemma 27** ([11]). *There exists a procedure* Source-Rew *that given a set* Σ *of st-tgds from* $\mathbf{S}$ *to* $\mathbf{T}$*, and a conjunctive query* $Q_{\mathbf{T}}$ *over* $\mathbf{T}$*, computes (in exponential time) a query in* $\mathrm{UCQ}^=$ *which is a source rewriting of* $Q_{\mathbf{T}}$ *under the mapping* $\mathcal{M}$ *specified by* Σ*.*

The following algorithm, proposed in [11], uses procedure Source-Rew to compute inverses. In particular, the algorithm computes a maximum recovery of the input mapping. In the algorithm we use the special predicate $\mathbf{C}(\cdot)$ that differentiates constant values from labeled null values (that is $\mathbf{C}(u)$ holds if and only if u is a constant value). We also use $\mathbf{C}(\bar{x})$, with $\bar{x}$ a tuple of variables $(x_1, \ldots, x_k)$, as a shorthand of $\mathbf{C}(x_1) \wedge \cdots \wedge \mathbf{C}(x_k)$.

```
Algorithm Inverse
Input: A mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ specified by a set $\Sigma$ of st-tgds.
Output: A mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ specified by a set $\Sigma'$ of tgds with disjunctions, equalities
and predicate $\mathbf{C}(\cdot)$.
1. Start with $\Sigma'$ as the empty set.
2. For every st-tgd $\varphi(\bar{x}) \rightarrow \exists\bar{y}\,\psi(\bar{x},\bar{y})$ in $\Sigma$, do the following:
   a. Let $Q_{\mathbf{T}}$ be the conjunctive query defined by formula $\exists\bar{y}\,\psi(\bar{x},\bar{y})$.
   b. Use Source-Rew to compute a formula $\alpha(\bar{x})$ in $\mathrm{UCQ}^=$ that is a source rewriting of
      $Q_{\mathbf{T}}$ under mapping $\mathcal{M}$.
   c. Add dependency $\exists\bar{y}\,\psi(\bar{x},\bar{y}) \wedge \mathbf{C}(\bar{x}) \rightarrow \alpha(\bar{x})$ to $\Sigma'$.
3. Return the mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ specified by $\Sigma'$.
```

▶ **Example 28.** Let $\mathbf{S} = \{A(\cdot,\cdot), B(\cdot)\}$, and $\mathbf{T} = \{S(\cdot,\cdot)\}$, and let $\mathcal{M}$ be the mapping for $\mathbf{S}$ to $\mathbf{T}$ specified by the st-tgds

$$\begin{aligned} A(x,y) &\rightarrow S(x,y), \\ B(x) &\rightarrow S(x,x). \end{aligned}$$

With input $\mathcal{M}$, algorithm Inverse first considers the st-tgd $A(x,y) \rightarrow S(x,y)$ and computes a source rewriting of $S(x,y)$. From the discussion previous to the algorithm we know that $A(x,y) \vee (B(x) \wedge x = y)$ is a source rewriting of $S(x,y)$. Thus the algorithm includes in Σ' the dependency $S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \rightarrow A(x,y) \vee (B(x) \wedge x = y)$. Then the algorithm considers dependency $B(x) \rightarrow S(x,x)$ and computes a source rewriting of $S(x,x)$ which is given by the source query $A(x,x) \vee B(x)$. Then the algorithm includes dependency $S(x,x) \wedge \mathbf{C}(x) \rightarrow A(x,x) \vee B(x)$ in Σ'. Finally, the output of the algorithm is the mapping $\mathcal{M}'$ specified by the dependencies

$$\begin{aligned} S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) &\rightarrow A(x,y) \vee (B(x) \wedge x = y), \\ S(x,x) \wedge \mathbf{C}(x) &\rightarrow A(x,x) \vee B(x). \end{aligned}$$ ◀

▶ **Theorem 29** ([6, 11]). *Let $\mathcal{M}$ be a mapping specified by st-tgds. Then with input $\mathcal{M}$, algorithm* Inverse *computes a maximum recovery of $\mathcal{M}$.*

By Theorems 21 and 22 we obtain the following corollary regarding the computation of Fagin-inverses and quasi-inverses.

▶ **Corollary 30.** *Let $\mathcal{M}$ be a mapping specified by st-tgds. If $\mathcal{M}$ has a Fagin-inverse (quasi-inverse), then with input $\mathcal{M}$, algorithm* Inverse *computes a Fagin-inverse (quasi-inverse) of $\mathcal{M}$.*

In general, the set Σ' constructed in algorithm Inverse is of exponential size. Notice that this directly depends on the size of the source rewritings computed by Source-Rew which are in general exponential. Nevertheless, there are cases for which this process can be done more efficiently. In particular, if mapping $\mathcal{M}$ is specified by a set of st-tgds that do not use existential quantification in the conclusions of dependencies, also called *full st-tgds* [18], then Step (2b) of algorithm Inverse can be accomplished in polynomial time [11, 34].

For the case of the Fagin-inverse, Arenas et al. [7] proposed an alternative algorithm that uses *target rewritings*. Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds, and $Q_{\mathbf{S}}$ a conjunctive query over $\mathbf{S}$. Then, a query $Q_{\mathbf{T}}$ is a *target rewriting* of $Q_{\mathbf{S}}$ under $\mathcal{M}$ if $\text{certain}_{\mathcal{M}}(Q_{\mathbf{T}}, I) = Q_{\mathbf{S}}(I)$ for every source instance I. That is, $Q_{\mathbf{T}}$ is a target rewriting of $Q_{\mathbf{S}}$ if and only if $Q_{\mathbf{S}}$ is a source rewriting of $Q_{\mathbf{T}}$. Although the notions of source and target rewriting are tightly related, their associated algorithmic problems are not equivalent. For example, as opposed to the case of source rewritings, for a conjunctive query $Q_{\mathbf{S}}$ a target rewriting does not always exist [7]. Nevertheless, Arenas et al. [7] showed that if $\mathcal{M}$ is a mapping specified by st-tgds that has a Fagin-inverse, then every conjunctive source query is target rewritable. Moreover, it can be proved that a target rewriting can be computed in exponential time and can be expressed as a union of conjunctive queries with equalities and inequalities ($\text{UCQ}^{=,\neq}$) [7, 34]. We formalize this in the following lemma.

▶ **Lemma 31** ([7, 34]). *There exists a procedure* Target-Rew *that given a set Σ of st-tgds from $\mathbf{S}$ to $\mathbf{T}$, and a conjunctive query $Q_{\mathbf{S}}$ over $\mathbf{S}$ that is target rewritable, computes (in exponential time) a query in $\text{UCQ}^{=,\neq}$ which is a target rewriting of $Q_{\mathbf{S}}$ under the mapping $\mathcal{M}$ specified by Σ.*

With this procedure we can present the following algorithm to compute Fagin-inverses (which is implicit in the work by Arenas et al. [7](Proposition 5.3)). In the algorithm we also use the following property. A tgd from $\mathbf{S}$ to $\mathbf{T}$ of the form $\varphi_1(\bar{x}) \vee \varphi_2(\bar{x}) \to \psi(\bar{x})$ is equivalent to the set of tgds $\{\varphi_1(\bar{x}) \to \psi(\bar{x}), \varphi_2(\bar{x}) \to \psi(\bar{x})\}$. That is, one can always eliminate disjunctions from the premises of tgds. Another property that we use is that equalities in the premises of tgds can always be eliminated by making the necessary variable replacements. That is, the dependency $\varphi(\bar{x}) \wedge x = y \to \psi(\bar{x})$ is equivalent to $\varphi(\bar{x}') \to \psi(\bar{x}')$ where $\bar{x}'$ is the tuple obtained from $\bar{x}$ by replacing every occurrence of y by x.

Algorithm Fagin-Inverse

Input: A mapping $\mathcal{M}$ from $\mathbf{S}$ to $\mathbf{T}$ specified by a set Σ of st-tgds that has a Fagin-inverse.
Output: A mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ specified by a set Σ' of tgds with inequalities and predicate $\mathbf{C}(\cdot)$.

```
1. Start with $\Sigma'$ as the empty set.
2. For every source $k$-ary relation symbol $R$ do the following:
   a. Let $x = (x_1, \ldots, x_k)$ be a $k$-tuple of distinct variables, and $Q_{\mathbf{S}}$ the conjunctive query
      defined by formula $R(\bar{x})$.
```

   b. Use TARGET-REW to compute a formula $\alpha(\bar{x})$ in UCQ$^{=,\neq}$ that is a target rewriting of $Q_{\mathbf{S}}$ under mapping $\mathcal{M}$.
   c. For every disjunct $\beta(\bar{x})$ of $\alpha(\bar{x})$ add dependency $\beta(\bar{x}) \wedge \mathbf{C}(\bar{x}) \to R(\bar{x})$ to Σ'.

3. Eliminate all the equality predicates in Σ' by making the necessary variable replacements (and eliminating the remaining predicates $\mathbf{C}(x)$ for every replaced variable x).

4. Return the mapping $\mathcal{M}'$ from $\mathbf{T}$ to $\mathbf{S}$ specified by Σ'.

▶ **Example 32.** Let $\mathbf{S} = \{A(\cdot,\cdot), B(\cdot)\}$ and $\mathbf{T} = \{S(\cdot,\cdot), T(\cdot), U(\cdot)\}$, and let $\mathcal{M}$ be the mapping in Example 5, that is, $\mathcal{M}$ is specified by the set of st-tgds

$$\begin{array}{rcl} A(x,y) & \to & S(x,y) \\ B(x) & \to & S(x,x) \\ B(x) & \to & T(x) \\ A(x,x) & \to & U(x) \end{array}$$

With input $\mathcal{M}$, algorithm FAGIN-INVERSE first considers relation symbol A and in Step (2b) and computes a target rewriting of $A(x,y)$. It can be shown that the query given by $(S(x,y) \wedge x \neq y) \vee (U(x) \wedge x = y)$ is a target rewriting of $A(x,y)$. Then in Step (2c) the algorithm adds dependencies

$$\begin{array}{rcl} S(x,y) \wedge x \neq y \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \to & A(x,y) \\ U(x) \wedge x = y \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \to & A(x,y) \end{array}$$

to the set Σ'. Then the algorithm considers relation symbol B and computes a target rewriting of $B(x)$. It can be proved that $T(x)$ is a target rewriting in this case, thus, the algorithm adds dependency $T(x) \wedge \mathbf{C}(x) \to B(x)$. Finally, in Step (4) the algorithm eliminates the equalities by variable replacements to obtain the set of dependencies

$$\begin{array}{rcl} S(x,y) \wedge x \neq y \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) & \to & A(x,y) \\ U(x) \wedge \mathbf{C}(x) & \to & A(x,x) \\ T(x) \wedge \mathbf{C}(x) & \to & B(x) \end{array}$$

Notice that the obtained mapping is almost exactly the mapping that is claimed to be a Fagin-inverse of $\mathcal{M}$ in Example 5. ◀

The correctness of algorithm FAGIN-INVERSE is stated in the following theorem.

▶ **Theorem 33** ([7, 34]). *Let $\mathcal{M}$ be a mapping specified by st-tgds that has a Fagin-inverse. Then with input $\mathcal{M}$, algorithm* FAGIN-INVERSE *computes a Fagin-inverse of $\mathcal{M}$.*

It should be pointed out that the first algorithms proposed to compute quasi-inverses [21], and maximum recoveries [10], used ad-hoc techniques and were far more complicated that the one that we presented in this section. The algorithms that we have presented were proposed by Arenas et al. [11, 6, 7] and are based on query rewriting procedures which makes them suitable for optimizations and can be benefited from the vast amount of work on query rewriting in the data integration and data exchange contexts. Fagin et al. [21] also proposed a simple algorithm to compute Fagin-inverses based on the *chase* procedure. We do not explain all the details of this algorithm but describe the main idea. We assume some familiarity with the chase procedure for tgds (see Chapter 1 for details on the chase procedure). For every source atom $R(\bar{x})$, the algorithm by Fagin et al. [21] considers all the atoms obtained by considering all possible combinations of equalities among the variables in $\bar{x}$. Those atoms are called *prime atoms* in [21]. For example, for a source relation $R(\cdot, \cdot, \cdot)$ the algorithm considers the prime atoms $R(x_1, x_1, x_1)$, $R(x_1, x_1, x_2)$, $R(x_1, x_2, x_2)$, $R(x_1, x_2, x_1)$, and $R(x_1, x_2, x_3)$. Assume that a Fagin-inverse is to be computed for a mapping $\mathcal{M}$ specified by a set Σ of st-tgds. Moreover, given a prime atom α, let $\text{chase}_\Sigma(\alpha)$ be the result of chasing α with Σ. Then for every prime atom α the algorithm generates a formula σ_α of the form $\beta \wedge \delta \wedge \gamma \to \alpha$, where β is the conjunction of all the target atoms in $\text{chase}_\Sigma(\alpha)$, δ is a conjunction of inequalities of the form $x \neq y$ for every pair of different variables mentioned in α, and γ is a conjunction of formulas $\mathbf{C}(x)$ for every variable x mentioned in α. Fagin et al. [21] proved that if $\mathcal{M}$ has a Fagin-inverse, then the set of formulas $\{\sigma_\alpha \mid \alpha \text{ is a prime source atom}\}$ specifies a Fagin-inverse of $\mathcal{M}$ [21].

5.2 Languages for expressing inverses

In this section we study the question of what the language needed to express inverses is. In particular we survey the results in the literature that justify the languages used as output in the algorithms of the previous section. In particular a first result which is immediately obtained from the algorithms is the following.

▶ **Theorem 34** ([20, 10]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds.*

1. *$\mathcal{M}$ has a maximum recovery specified by a set of tgds from $\mathbf{T}$ to $\mathbf{S}$ with disjunctions and equalities in the conclusions and predicate $\mathbf{C}(\cdot)$ in the premises.*
2. *If $\mathcal{M}$ has a Fagin-inverse (quasi-inverse), then there exists a Fagin-inverse (quasi-inverse) of $\mathcal{M}$ specified by a set of tgds from $\mathbf{T}$ to $\mathbf{S}$ with disjunctions and equalities in the conclusions and predicate $\mathbf{C}(\cdot)$ in the premises.*
3. *If $\mathcal{M}$ has a Fagin-inverse, there exists a Fagin-inverse of $\mathcal{M}$ specified by a set of tgds from $\mathbf{T}$ to $\mathbf{S}$ with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises.*

Parts 1) and 2) of the theorem follow from algorithm INVERSE, while part 3) follows from algorithm FAGIN-INVERSE. Fagin et al. [20, 21] use a slightly different language to specify quasi-inverses of mappings specified by st-tgds. In particular, they use tgds with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises and disjunctions (without equalities) in the conclusions. It is not difficult to see that in the output of algorithm INVERSE one can replace the equalities in the conclusions of dependencies by inequalities in the premises as is outlined in the following example.

▶ **Example 35.** Consider the mapping $\mathcal{M}$ in Example 28, that is, $\mathcal{M}$ is specified by the dependencies $A(x, y) \to S(x, y)$ and $B(x) \to S(x, x)$. In that example, we compute a maximum recovery $\mathcal{M}'$ of $\mathcal{M}$ specified by the set of dependencies

$$\begin{aligned} S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) &\to A(x,y) \vee (B(x) \wedge x = y), \qquad (9)\\ S(x,x) \wedge \mathbf{C}(x) &\to A(x,x) \vee B(x). \end{aligned}$$

Notice that from (9) we can generate two formulas depending on whether $x = y$ or $x \neq y$ obtaining the set

$$\begin{aligned} S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \wedge x \neq y &\to A(x,y), \\ S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \wedge x = y &\to A(x,y) \vee B(x), \qquad (10)\\ S(x,x) \wedge \mathbf{C}(x) &\to A(x,x) \vee B(x). \qquad (11) \end{aligned}$$

Finally we can use variable substitutions to eliminate the equality in (10). In that case the obtained dependency is equivalent to (11), and thus the final set of dependencies is

$$\begin{array}{rcl} S(x,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \wedge x \neq y & \rightarrow & A(x,y), \\ S(x,x) \wedge \mathbf{C}(x) & \rightarrow & A(x,x) \vee B(x). \end{array}$$

◀

It was shown by Arenas et al. [6](Lemma 4.2) that if from the output of algorithm INVERSE we eliminate the equalities with the process outlined in the above example, then the obtained mapping is still a maximum recovery of $\mathcal{M}$. By a different procedure Fagin et al. [20] showed that for mappings specified by st-tgds that has a quasi-inverse, there exists a quasi inverse specified by a set of tgds with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises and disjunctions in the conclusions. Thus we have the following.

▶ **Theorem 36** ([6, 20]). *Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$ specified by st-tgds.*
1. *$\mathcal{M}$ has a maximum recovery specified by a set of tgds from $\mathbf{T}$ to $\mathbf{S}$ with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises and disjunctions in the conclusions.*
2. *If $\mathcal{M}$ has a quasi-inverse, then there exists a quasi-inverse of $\mathcal{M}$ specified by a set of tgds from $\mathbf{T}$ to $\mathbf{S}$ with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises and disjunctions in the conclusions.*

We know what languages are sufficient to specify inverses but, are all the features of these languages strictly needed to specify inverses? For example, do we really need disjunctions to specify maximum recoveries and quasi-inverses? Do we really need predicate $\mathbf{C}(\cdot)$ to specify Fagin-inverses? In what follows we answer these questions.

The first result that we report was proved by Fagin et al. [20, 21], and states that predicate $\mathbf{C}(\cdot)$ is strictly necessary to specify Fagin-inverses.

▶ **Theorem 37** (Necessity of $\mathbf{C}(\cdot)$ [20]). *There exists a mapping $\mathcal{M}$ specified by st-tgds that has a Fagin-inverse but does not have a Fagin-inverse specified by tgds with inequalities in the premises and disjunctions in the conclusions (without using predicate $\mathbf{C}(\cdot)$).*

▶ **Example 38.** Consider the mapping $\mathcal{M}$ specified by the st-tgds $A(x,y) \rightarrow \exists z\, \big(S(x,z) \wedge S(z,y)\big)$. It can be shown that $\mathcal{M}$ is Fagin-invertible. In fact, the mapping $\mathcal{M}'$ specified by $S(x,z) \wedge S(z,y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \rightarrow A(x,y)$ is a Fagin-inverse of $\mathcal{M}$. Fagin et al. [20] show that $\mathcal{M}$ does not have a Fagin-inverse that does not use $\mathbf{C}(\cdot)$. To see the intuition of the failure, lets show that if we delete the $\mathbf{C}(\cdot)$ predicates in the definition of $\mathcal{M}'$, the resulting mapping is no longer a Fagin-inverse of $\mathcal{M}$. Thus consider the mapping $\mathcal{M}''$ specified by $S(x,z) \wedge S(z,y) \rightarrow A(x,y)$ and assume that $\mathcal{M}''$ is a Fagin-Inverse of $\mathcal{M}$. Then for every source instance I we have that $(I,I) \in \mathcal{M} \circ \mathcal{M}''$. Now consider the instance $I = \{A(1,2), A(2,1)\}$. Since $(I,I) \in \mathcal{M} \circ \mathcal{M}''$ we know that there exists an instance K such that $(I,K) \in \mathcal{M}$ and $(K,I) \in \mathcal{M}''$. Thus, by the definition of $\mathcal{M}$, we have that there exists elements a, b such that $S(1,a), S(a,2), S(2,b), S(b,1) \in K$. Then by definition of $\mathcal{M}''$ we have that $A(a,b), A(b,a) \in I$ and thus, either $a = 1$ and $b = 2$, or $a = 2$ and $b = 1$. Assume first that $a = 1$ and $b = 2$, then we have that $S(1,1), S(2,2) \in K$ which implies that $A(1,1), A(2,2) \in I$ which is a contradiction. If we assume that $a = 2$ and $b = 1$ we obtain the same contradiction. Notice that we cannot obtain this contradiction with $\mathcal{M}'$ since I has as solution under $\mathcal{M}$ the instance $K' = \{A(1,n), A(n,2), A(2,m), A(m,2)\}$ with n and m different null values. Moreover, K' has I as solution under $\mathcal{M}'$ and thus $(I,I) \in \mathcal{M} \circ \mathcal{M}'$. ◀

One can prove a stronger result for the case of maximum recoveries, namely that predicate $\mathbf{C}(\cdot)$ is needed even if we allow the full power of First-Order logic.

▶ **Theorem 39** (Necessity of $\mathbf{C}(\cdot)$ for maximum recoveries [10]). *There exists a mapping $\mathcal{M}$ specified by st-tgds that has no maximum recovery specified by First-Order sentences that do not use predicate $\mathbf{C}(\cdot)$.*

The following result shows that we need either inequalities in the premises or equalities in the conclusions of dependencies in order to specify Fagin-inverses. This immediately implies the necessity of these features to specify quasi-inverses and maximum recoveries.

▶ **Theorem 40** (Necessity of either $=$ or $\neq$ [20]). *There exists a mapping $\mathcal{M}$ specified by st-tgds that has a Fagin-inverse but does not have a Fagin-inverse specified by tgds with predicate $\mathbf{C}(\cdot)$ in the premises and disjunctions in the conclusions.*

Fagin et al. [21] use the mapping $\mathcal{M}$ in Example 32 to show the necessity of either inequalities in the premises of equalities in the conclusions to specify Fagin-inverses. The only remaining property that we need to prove is that disjunctions are necessary for quasi-inverses and maximum recoveries.

▶ **Theorem 41** (Necessity of $\vee$ [34, 20]).

1. *There exists a mapping $\mathcal{M}$ specified by st-tgds that has no maximum recovery specified by tgds with predicate $\mathbf{C}(\cdot)$ in the premises and equalities in the conclusions.*
2. *There exists a mapping $\mathcal{M}$ specified by st-tgds that has a quasi-inverse but has no quasi-inverse specified by tgds with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises.*

6 Query Language-Based Inverses of Schema Mappings

In the data exchange scenario, the standard procedure used to exchange data with a mapping is based on the *chase* procedure [18] (See Chapter 1 for a comprehensive study of the chase procedure in data exchange). More precisely, given a mapping $\mathcal{M}$ and a source database I, a *canonical* translation of I according to $\mathcal{M}$ is computed by *chasing* I with the set of dependencies defining $\mathcal{M}$ [18]. Thus, when computing an inverse of $\mathcal{M}$, it would be desirable from a practical point of view to obtain a mapping $\mathcal{M}'$ where the chase procedure can be used to exchange data. Unfortunately, the notions of inverse that we have introduced in the previous sections, express inverses in some mapping languages which include features that are difficult to use in practice. The most important of those issues is the use of disjunctions in the conclusion of the mapping rules.

To provide a solution for the aforementioned issue, Arenas et. al [6] introduce a query-language based notion of inverse called $\mathcal{C}$-maximum recovery, with $\mathcal{C}$ a class of queries. The idea is that when one focuses on particular query languages one can obtain inverses with better properties regarding the languages needed to specify these inverses. In particular, Arenas et al. [6] proved that when one focuses on conjunctive queries, one can obtain inverses that can be expressed in a *chaseable* language.

The main intuition behind the notion proposed by Arenas et al. [6] is to use queries to measure the amount of information that a mapping $\mathcal{M}'$ can recover with respect to a mapping $\mathcal{M}$. Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, $\mathcal{M}'$ a mapping from $\mathbf{T}$ to $\mathbf{S}$. Notice that $\mathcal{M} \circ \mathcal{M}'$ is a mapping that goes from $\mathbf{S}$ to $\mathbf{T}$ and then to $\mathbf{S}$ again. Thus one can measure the amount of information recovered by $\mathcal{M}'$ by using queries over $\mathbf{S}$. Let Q be a query over $\mathbf{S}$, then we say that $\mathcal{M}'$ recovers sound information w.r.t. Q under $\mathcal{M}$ if for every instance I it holds that

$$\text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I) \subseteq Q(I).$$

Thus, by posing the query Q over the space of solutions for I under $\mathcal{M} \circ \mathcal{M}'$, one can only obtain tuples that are already in the evaluation of Q over the original instance I. This notion can be generalized to a class $\mathcal{C}$ of queries, which gives rise to the notion of $\mathcal{C}$-recovery.

▶ **Definition 42** ([6]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, $\mathcal{M}'$ a mappings from $\mathbf{T}$ to $\mathbf{S}$, and $\mathcal{C}$ a class of queries over $\mathbf{S}$. Then $\mathcal{M}'$ is a $\mathcal{C}$-recovery of $\mathcal{M}$ if for every query $Q \in \mathcal{C}$ and every source instance I it holds that

$$\text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I) \subseteq Q(I).$$

As for the definition of maximum recovery, one can compare different $\mathcal{C}$-recoveries. Let $\mathcal{M}'$ and $\mathcal{M}''$ be $\mathcal{C}$-recoveries of $\mathcal{M}$, and suppose that for every query $Q \in \mathcal{C}$ and source instance I, it holds that

$$\text{certain}_{\mathcal{M}\circ\mathcal{M}''}(Q, I) \subseteq \text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I) \subseteq Q(I).$$

Clearly, the mapping $\mathcal{M}'$ is better than $\mathcal{M}''$ to recover information w.r.t. queries in $\mathcal{C}$, since $\text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I)$ is *closer* to $Q(I)$ than $\text{certain}_{\mathcal{M}\circ\mathcal{M}''}(Q, I)$. This discussion naturally gives rise to the notion of $\mathcal{C}$-maximum recovery.

▶ **Definition 43** ([6]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, and $\mathcal{C}$ a class of queries over $\mathbf{S}$. Then $\mathcal{M}'$ is a $\mathcal{C}$-maximum recovery of $\mathcal{M}$ if
1. $\mathcal{M}'$ is a $\mathcal{C}$-recovery of $\mathcal{M}$, and
2. for every $\mathcal{C}$-recovery $\mathcal{M}''$ of $\mathcal{M}$, it holds that $\text{certain}_{\mathcal{M}\circ\mathcal{M}''}(Q, I) \subseteq \text{certain}_{\mathcal{M}\circ\mathcal{M}'}(Q, I)$.

Before stating some general results regarding $\mathcal{C}$-maximum recoveries and the relationship with the notions presented in the previous sections, let us show some examples on what is the influence of the class $\mathcal{C}$ of queries in the notion of $\mathcal{C}$-maximum recovery. Before presenting the example, we note that if $\mathcal{M}'$ is a maximum recovery of $\mathcal{M}$, then $\mathcal{M}'$ is a $\mathcal{C}$-maximum recovery of $\mathcal{M}$ for every class $\mathcal{C}$ of queries [8]. This is not difficult to show given the set of tools for maximum recoveries proposed by Arenas et al. [11] (see [34, 8] for details on the relationship between maximum recoveries and $\mathcal{C}$-maximum recoveries).

▶ **Example 44.** Let $\mathcal{M}$ be specified by these two st-tgds:

$$A(x, y) \;\rightarrow\; R(x, y), \qquad B(x) \;\rightarrow\; R(x, x).$$

It can be shown that mapping $\mathcal{M}_1$ specified by dependency:

$$R(x, y) \;\rightarrow\; A(x, y) \;\vee\; \big(B(x) \wedge x = y\big)$$

is a UCQ-maximum recovery of $\mathcal{M}$ (in fact $\mathcal{M}_1$ is a maximum recovery of $\mathcal{M}$). To specify $\mathcal{M}_1$, we have used a disjunction in the conclusion of the dependency. This disjunction is unavoidable if we use UCQ to retrieve information [8]. On the other hand, if we focus on CQ to retrieve information, then, intuitively, there is no need for disjunctions in the right-hand side of the rules as conjunctive queries cannot extract disjunctive information. In fact, it can be shown that a CQ-maximum recovery of $\mathcal{M}$ is specified by dependency:

$$R(x, y) \wedge x \neq y \;\rightarrow\; A(x, y). \qquad ◀$$

The example suggests that the notion of CQ-maximum recovery is a strict generalization of the notion of UCQ-maximum recovery. More importantly, it shows that for different choices of the class of queries used, we obtain different notions of inverses of schema mappings. The following results show that one can actually characterize the notions of Fagin-inverse and quasi-inverse for particular classes of queries. In the theorem we use UCQ$^{\neq}$ to denote the class of unions of conjunctive queries with inequalities.

▶ **Theorem 45** ([6, 8]). *Let $\mathcal{M}$ be a mapping specified by a set of st-tgds.*

1. *Assume that $\mathcal{M}$ has a Fagin-inverse. Then $\mathcal{M}'$ is a Fagin-inverse of $\mathcal{M}$ if and only if $\mathcal{M}'$ is a* UCQ$^{\neq}$*-maximum recovery of $\mathcal{M}$.*
2. *Assume that $\mathcal{M}$ has a quasi-inverse. There exists a class $\mathcal{C}_{\mathcal{M}}$ that depends on $\mathcal{M}$ such that $\mathcal{M}'$ is a quasi-inverse of $\mathcal{M}$ if and only if $\mathcal{M}'$ is a $\mathcal{C}_{\mathcal{M}}$-maximum recovery of $\mathcal{M}$.*

Arenas et al. [6, 8] provide several tools to work with $\mathcal{C}$-maximum recoveries including characterizations for the mappings that admit $\mathcal{C}$-maximum recoveries and a general necessary and sufficient condition for the existence of $\mathcal{C}$-maximum recoveries. We refer the reader to [34, 8] for a comprehensive study of $\mathcal{C}$-maximum recoveries, and in particular, for a definition of the class $\mathcal{C}_{\mathcal{M}}$ used in part 2) of Theorem 45.

The language of CQ-maximum recoveries

Arenas et al. [6] study several properties about $\mathcal{C}$-maximum recoveries when one focuses on CQ as the class $\mathcal{C}$ of queries. In particular, they provide an algorithm to compute CQ-maximum recoveries for st-tgds showing the following theorem.

▶ **Theorem 46** ([6]). *Every mapping specified by a set of st-tgds has a* CQ*-maximum recovery, which is specified by a set of tgds with inequalities and predicate* $\mathbf{C}(\cdot)$ *in the premises.*

Notice that the language needed to express CQ-maximum recoveries of st-tgds has the same good properties as st-tgds for data exchange. In particular, the language is *chaseable* in the sense that the standard chase procedure can be used to obtain a canonical solution. Thus, compared to the notions of Fagin-inverse, quasi-inverse, and maximum recovery, the notion of CQ-maximum recovery has two advantages: (1) every mapping specified by st-tgds has a CQ-maximum recovery (which is not the case for Fagin-inverses and quasi-inverses), and (2) such a CQ-maximum recovery can be specified in a mapping language with good properties for data exchange (which is not the case for quasi-inverses and maximum recoveries).

The algorithm proposed by Arenas et al. [6] to compute CQ-maximum recoveries is based on the algorithm for computing maximum recoveries reported in the previous section. After computing a maximum recovery, the algorithm does a post-processing step to eliminate the disjunctions in the conclusions of the dependencies by using a notion of *conjunctive-query products* [6, 34]. Given two conjunctive queries Q_1 and Q_2, the product query $Q_1 \times Q_2$ is, intuitively, the closest conjunctive query to both Q_1 and Q_2 in terms of *homomorphisms*. Let us to introduce some terminology to formalize this notion.

Let Q_1 and Q_2 be two n-ary conjunctive queries, and assume that $\bar{x}$ is the tuple of free variables of Q_1 and Q_2. The *product* of Q_1 and Q_2, denoted by $Q_1 \times Q_2$, is defined as a k-ary conjunctive query (with $k \leq n$) constructed as follows. Let $f(\cdot,\cdot)$ be a one-to-one function from pairs of variables to variables such that:

1. $f(x,x) = x$ for every variable x in $\bar{x}$, and
2. $f(y,z)$ is a fresh variable (mentioned neither in Q_1 nor in Q_2) in any other case.

Then for every pair of atoms $R(y_1,\dots,y_m)$ in Q_1 and $R(z_1,\dots,z_m)$ in Q_2, the atom $R(f(y_1,z_1),\dots,f(y_m,z_m))$ is included as a conjunct in the query $Q_1 \times Q_2$. Furthermore, the set of free variables of $Q_1 \times Q_2$ is the set of variables from $\bar{x}$ that are mentioned in $Q_1 \times Q_2$. For example, consider conjunctive queries:

$$\begin{aligned} Q_1(x_1,x_2) &\ :\ P(x_1,x_2) \wedge R(x_1,x_1), \\ Q_2(x_1,x_2) &\ :\ \exists y\,(P(x_1,y) \wedge R(x_2,x_2)). \end{aligned}$$

Then we have that $Q_1 \times Q_2$ is the conjunctive query:

$$(Q_1 \times Q_2)(x_1) \quad : \quad \exists z_1 \exists z_2 \, (P(x_1, z_1) \wedge R(z_2, z_2)).$$

In this case, we have used a function f such that $f(x_1, x_1) = x_1$, $f(x_2, y) = z_1$, and $f(x_1, x_2) = z_2$. As shown in the example, the free variables of $Q_1 \times Q_2$ do not necessarily coincide with the free variables of Q_1 and Q_2. The definition of the product of queries is motivated by the standard notion of *Cartesian product* of graphs. In fact, if Q_1 and Q_2 are Boolean queries constructed by using a single binary relation $E(\cdot, \cdot)$, then the product $Q_1 \times Q_2$ exactly resembles the graph-theoretical Cartesian product [27].

The product of queries is the key ingredient in the algorithm CQ-Max-Recovery proposed by Arenas et al. [6] to compute CQ-maximum recoveries. Given a mapping $\mathcal{M}$ specified by st-tgds, CQ-Max-Recovery first uses algorithm Inverse to compute a maximum recovery $\mathcal{M}'$ of $\mathcal{M}$. Then it eliminates equalities in the conclusions of the dependencies defining $\mathcal{M}'$ by adding the necessary inequalities in the premises of the dependencies (as outlined in Example 35). Finally, the algorithm replaces the remaining disjunctions $Q_1 \vee Q_2 \vee \cdots \vee Q_k$ in the conclusions of the tgds, by the conjunctive query $Q_1 \times Q_2 \times \cdots \times Q_k$. The final output of CQ-Max-Recovery is a set of tgds with inequalities and predicate $\mathbf{C}(\cdot)$ in the premises (without disjunctions in the conclusions).

▶ **Example 47.** Assume that the output of Inverse contains the dependency

$$A(x, y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \;\rightarrow\; \big(P(x, y) \wedge R(x, x)\big) \;\vee\; \exists z \,\big(P(x, z) \wedge R(y, y)\big).$$

Then algorithm CQ-Max-Recovery replaces this dependency by

$$A(x, y) \wedge \mathbf{C}(x) \wedge \mathbf{C}(y) \;\rightarrow\; \exists u \exists v \,\big(P(x, u) \wedge R(v, v)\big),$$

since $\exists u \exists v \, (P(x, u) \wedge R(v, v))$ is the product of $P(x, y) \wedge R(x, x)$ and $\exists z \, \big(P(x, z) \wedge R(y, y)\big)$. ◀

Arenas et al. [6] also study the minimality of the language used to express CQ-maximum recoveries, showing that inequalities and predicate $\mathbf{C}(\cdot)$ are both needed to express the CQ-maximum recoveries of mappings specified by st-tgds. Arenas et al. [8] also show that the class CQ is *optimal* to obtain the desired result of a notion of inverse with good properties for data exchange. In particular, if one uses either $\text{CQ}^{\neq}$ or $\text{UCQ}^{\neq}$ in the definition of $\mathcal{C}$-maximum recovery, then the language needed to express inverses is no longer *chaseable* [8].

7 Inversion in the Presence of Null Values in Source Instances

Fagin et al. [22] made the observation that almost all the literature about data exchange and, in particular, the literature about inverses of schema mappings, assume that source instances do not contain null values. Most of the results regarding inverses that we have reported so far are proved for the case of mappings in which the source instances contain only constant values while target instances may contain constant and null values. Fagin et al. [22] go a step further and propose new refined notions for inverting mappings that consider nulls in the source. In particular, they propose the notions of *extended inverse*, and of *extended recovery* and *maximum extended recovery*. In this section, we review the definitions of the latter two notions and compare them with the previously proposed notions of recovery and maximum recovery (for a comprehensive study of the notion of extended inverse see the work by Fagin et al. [22]).

The first observation to make is that since null values are intended to represent *missing* or *unknown* information, they should not be treated naively as constants [28]. In fact, as shown by Fagin et al. [22], if one treats nulls in that way, the existence of a maximum recovery for mappings given by st-tgds is no longer guaranteed.

▶ **Example 48.** Consider a source schema $\mathbf{S} = \{A(\cdot), B(\cdot)\}$ and a target schema $\mathbf{T} = \{S(\cdot)\}$, and let $\mathcal{M}$ be a mapping specified by the st-tgds

$$\begin{array}{lcl} A(x) & \rightarrow & \exists u S(u) \\ B(x) & \rightarrow & S(x) \end{array}$$

From Theorem 24, we know that if source instances only contain constant values, then $\mathcal{M}$ has a maximum recovery. This property holds since, under this assumption, every source instance I has a witness solution (see Definition 23 and Theorem 24). For example, for the instance $I = \{A(1)\}$ the target instance $J = \{S(n)\}$, with n a null value, is a witness solution of I. In fact, if I' is any source instance such that $J \in \text{Sol}_{\mathcal{M}}(I)$ then $\text{Sol}_{\mathcal{M}}(I) \subseteq \text{Sol}_{\mathcal{M}}(I')$. Assume now that instances of $\mathbf{S}$ may contain constant and null values. Then we have that J is no longer a witness solution of I under $\mathcal{M}$. To see this consider the source instance $I' = \{B(n)\}$. Then we have that $J \in \text{Sol}_{\mathcal{M}}(I')$ but, for example $J' = \{S(2)\}$ is a solution for I but not for I', therefore $\text{Sol}_{\mathcal{M}}(I) \not\subseteq \text{Sol}_{\mathcal{M}}(I')$, and thus J is not a witness solution of I. In fact, it can be proved that if source instances may contain null values then I has no witness solution under $\mathcal{M}$ implying that $\mathcal{M}$ has no maximum recovery if null are allowed in the source. ◀

Notice that in the above example, nulls in the source are considered as constants when evaluating the tgds. Since nulls should not be treated naively when exchanging data, Fagin et al. [22] proposed a new way to deal with null values based on homomorphisms. Recall that given instances I and I' containing constant and null values, a homomorphism from I to I' is a function h that is the identity over constant values, maps nulls to constants or null values, and is such that if $R(a_1, \ldots, a_k)$ is a fact in I, then $R(h(a_1), \ldots, h(a_k))$ is a fact in I'. Intuitively, in order to treat null values and constants differently, Fagin et al. [22] *close* mappings under homomorphisms. This idea is supported by the fact that nulls are intended to represent unknown data, thus, it should be possible to replace them by arbitrary values. Formally, the authors introduce the following concept.

▶ **Definition 49** ([22]). Let $\mathcal{M}$ be a mapping. The *homomorphic extension* of $\mathcal{M}$, denoted by $e(\mathcal{M})$, is the mapping

$$\begin{array}{rl} e(\mathcal{M}) = \{(I, J) \mid & \text{there exist } I', J' \text{ such that } (I', J') \in \mathcal{M} \text{ and there exist} \\ & \text{homomorphisms from } I \text{ to } I' \text{ and from } J' \text{ to } J\}. \end{array}$$

The idea is that for a mapping $\mathcal{M}$ that has nulls in source and target instances, one does not have to consider $\mathcal{M}$ but $e(\mathcal{M})$ as the mapping to deal with for exchanging data and computing mapping operators since $e(\mathcal{M})$ treats nulls in a meaningful way [22]. The following result shows that with this new semantics one can avoid anomalies as the one shown in Example 48.

▶ **Theorem 50** ([22]). *For every mapping $\mathcal{M}$ specified by a set of st-tgds and with nulls in source and target instances, $e(\mathcal{M})$ has a maximum recovery.*

As mentioned above, Fagin et al. [22] go a step further by introducing new notions of inverse for mappings that consider nulls in the source. More specifically, the authors introduce the following definitions

▶ **Definition 51** ([22]). Let $\mathcal{M}$ be a mapping from $\mathbf{S}$ to $\mathbf{T}$, in which source and target instances may contain null values. Mapping $\mathcal{M}'$ is an *extended recovery* of $\mathcal{M}$ if $(I, I) \in e(\mathcal{M}) \circ e(\mathcal{M}')$, for every instance I of $\mathbf{S}$. Then given an extended recovery $\mathcal{M}'$ of $\mathcal{M}$, the mapping $\mathcal{M}'$ is a *maximum extended recovery* of $\mathcal{M}$ if for every extended recovery $\mathcal{M}''$ of $\mathcal{M}$, it holds that $e(\mathcal{M}) \circ e(\mathcal{M}') \subseteq e(\mathcal{M}) \circ e(\mathcal{M}'')$.

At a first glance, one may think that the notions of maximum recovery and maximum extended recovery are incomparable. Nevertheless, as shown by Arenas et al. [5] there is a tight connection between these two notions.

▶ **Theorem 52** ([5]). *Let $\mathcal{M}$ be a mapping that may have nulls values in source and target instances. Then $\mathcal{M}$ has a maximum extended recovery if and only if $e(\mathcal{M})$ has a maximum recovery. Moreover, $\mathcal{M}'$ is a maximum extended recovery of $\mathcal{M}$ if and only if $e(\mathcal{M}')$ is a maximum recovery of $e(\mathcal{M})$.*

One of the main result of Fagin et al. [22] regarding maximum extended recoveries is that every mapping specified by st-tgds having nulls in source and target instances has a maximum extended recovery. This result is implied by Theorems 50 and 52, and we formalize it in the following theorem.

▶ **Theorem 53** ([22]). *Let $\mathcal{M}$ be a mapping specified by st-tgds in which source and target instances may contain null values. Then $\mathcal{M}$ has a maximum extended recovery.*

It was left as an open problem to identify what is the exact language needed to express maximum extended recoveries [22]. In fact, it is even open whether maximum extended recoveries can be specified if the full power of First-Order logic is allowed to construct mappings.

8 Conclusions

As many information-system problems involve not only the design and integration of complex application artifacts, but also their subsequent manipulation, the definition and implementation of some operators for schema mappings has been identified as a fundamental issue to be solved [12, 13]. Nowadays, the community recognizes the need to develop techniques to manipulate these mappings' specifications and, in particular, the inverse of a schema mapping has been identified as one of the fundamental operators to be studied in this area. In this chapter, we have surveyed the main definition for the inverse operator proposed in the literature and the results that have been obtained in the last years.

One very important and challenging problem is the interplay between the inverse operator and other schema mapping operators, in particular the composition of schema mappings [33, 19]. Arenas et al. [9] proved that the mapping that results from composing mappings specified by st-tgds is not always invertible (even considering the relaxed notion of CQ-maximum recovery). This opens the question on good notions for inversion and composition of schema mappings, and a language for expressing mappings, suitable to deal with the interplay between the two operators [5]. A first attempt and a partial solution for this problem was given in [9].

The definition of the appropriate semantics for the inverse operator has proven to be a non-trivial task, in which many *sensible* decisions had to be taken. In fact, the answer to each of these decisions has given rise to different semantics for the inverse operator. Some general questions that one might want to answer include whether we want inverses that are guaranteed to be *consistent* in a general scenario, or do we settle for relaxed notions that allow only answering conjunctive queries (or other restricted classes of queries)? Certainly, the latter question involves a tradeoff, since the more general operators usually require more expressive languages, and their computation is more complex.

The spread of new semantics for the schema mappings, either modified semantics for mappings specified by standard logical specifications over the relational model [32, 22], or mappings for data models beyond the relational model, such as XML, which need different mapping specification languages [4, 2, 36], originates several challenges. Under these new scenarios, previously defined mapping operators have to be re-studied. This shows the importance of having general notions of inverse that are not tied to a particular schema mapping semantic, language or data model. Among the notions that we have presented, only the notion of maximum recovery is defined in a general setting and can be applied to abstract mappings independent of the mapping specification language, the semantics used for logical specifications, or the data model used. Nevertheless, there is not yet clear consensus about which semantics for the inverse operator is the appropriate one in general, and we think that particular applications would need to use different inverses depending on their specific needs. We hope the definitions and results presented in this chapter would be useful to compare the proposals for inverses, their characteristics, and their applicability in different contexts.

References

1 S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.

2 S. Amano, L. Libkin, and F. Murlak. XML schema mappings. In *PODS*, pages 33–42, 2009.

3 M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *PODS*, pages 229–240, 2004.

4 M. Arenas and L. Libkin. XML data exchange: Consistency and query answering. *J. ACM*, 55(2), 2008.

5 M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Composition and inversion of schema mappings. *SIGMOD Record*, 38(3):17–28, 2009.

6 M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Inverting schema mappings: Bridging the gap between theory and practice. *PVLDB*, 2(1):1018–1029, 2009.

7 M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In *PODS*, pages 227–238, 2010.

8 M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Query language based inverses of schema mappings: Semantics, computation, and closure properties. *VLDB J.*, 21(6):823–842, 2012.

9 M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. The language of plain SO-tgds: Composition, inversion and structural properties. *J. Comput. Syst. Sci.*, 79(6):763–784, 2013.

10 M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: bringing exchanged data back. In *PODS*, pages 13–22, 2008.

11 M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: bringing exchanged data back. *TODS*, 34(4), 2009.

12 P. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

13 P. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD*, pages 1–12, 2007.

14 G. de Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.

15 O. Duschka and M. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997.

16 R. Fagin. Inverting schema mappings. In *PODS*, pages 50–59, 2006.

17 R. Fagin. Inverting schema mappings. *TODS*, 32(4), 2007.

18 R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89–124, 2005.

19 R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *TODS*, 30(4):994–1055, 2005.

20 R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Quasi-inverses of schema mappings. In *PODS*, pages 123–132, 2007.

21 R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Quasi-inverses of schema mappings. *TODS*, 33(2), 2008.

22 R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Reverse data exchange: coping with nulls. In *PODS*, pages 23–32, 2009.

23 R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Schema mapping evolution through composition and inversion. In Z. Bellahsene, A. Bonifati, and E. Rahm, editors, *Schema Matching and Mapping*, pages 191–222. Springer, 2011.

24 A. Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(1):40–47, 2000.

25 A. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

26 A. Halevy, Z. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

27 P. Hell and J. Nešetřil. *Graphs and Homomorphisms.* Oxford University Press, 2004.

28 T. Imielinski and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

29 M. Lenzerini. Data integration: a theoretical perspective. In *PODS*, pages 233–246, 2002.

30 A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *PODS*, pages 95–104, 1995.

31 A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.

32 L. Libkin. Data exchange and incomplete information. In *PODS*, pages 60–69, 2006.

33 S. Melnik. *Generic Model Management: concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science.* Springer, 2004.

34 J. Pérez. *Schema Mapping Management in Data Exchange Systems.* PhD thesis, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 2011.

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

36 J. Terwilliger, P. Bernstein, and S. Melnik. Full-fidelity flexible object-oriented XML access. *PVLDB*, 2(1):1030–1041, 2009.

Reasoning about Schema Mappings *

Emanuel Sallinger

Vienna University of Technology
sallinger@dbai.tuwien.ac.at

Abstract

Schema mappings are an important tool in several areas of database research. Recently, the topic of *reasoning about schema mappings* was given attention, in particular revolving around the central concepts of *equivalence* and *optimality*. In this chapter, we survey these results. First, we introduce *relaxed notions of logical equivalence* and show their potential for finding optimized schema mappings. We then look at applications of these concepts to optimization, normalization, and schema mapping management, as well as the boundaries of computability. We conclude by giving a glimpse at reasoning about schema mappings in a broader sense by looking at how to debug schema mappings.

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

Keywords and phrases data exchange, data integration, schema mappings, equivalence, optimality, normalization, schema mapping management

Digital Object Identifier 10.4230/DFU.Vol5.10452.97

1 Introduction

Schema mappings are high-level specifications that describe the relationship between two database schemas. They are an important tool in several areas of database research, notably in data exchange [16, 8] and data integration [17, 15]. Over the past years, schema mappings have been extensively studied.

In this chapter, we will focus on the topic of reasoning about schema mappings. Central to any reasoning task is the concept of *implication*, and its close relative, *equivalence*. Since schema mappings are usually specified by logical formulas, the natural starting point of finding equivalence between schema mappings is

- **logical equivalence**: schema mappings that are satisfied by the same database instances are treated as being equivalent.

So now that we have a notion of equivalence, a natural next step is to use it to *optimize* schema mappings. That is, finding out the "best" among all equivalent schema mappings given some optimality criterion. Fortunately, there are algorithms for computing such optimized forms for a broad range of optimality criteria, as we will explore in Section 5.

Unfortunately, it turns out that logical equivalence is a quite restrictive concept for common tasks of data exchange. In particular, we can find two schema mappings where one is clearly preferred to the other, they both work perfectly well for the given data exchange task, but they are not logically equivalent. Hence we would not find the better one using optimization procedures for logical equivalence. This is clearly unsatisfactory.

* This work was supported by the Vienna Science and Technology Fund (WWTF), projects ICT08-032 and ICT12-15, and by the Austrian Science Fund (FWF): P25207-N23.

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 97–127
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

To remedy this situation, Fagin, Kolaitis, Nash and Popa in [9] introduced **relaxed notions of equivalence**: Notions that are less strict than logical equivalence, and therefore admit more optimization potential. These are

- **data-exchange equivalence**: schema mappings which behave in the same way for data-exchange are seen as equivalent, and
- **conjunctive-query equivalence**: schema mappings which behave similarly for answering conjunctive queries on the target database are treated as equivalent.

Therefore, using these notions, two of the prevalent applications of schema mappings can be reasoned about. The question remains of course: In which cases are there, hopefully efficient, algorithms for optimization under these relaxed notions of equivalence? This is a complex question that depends on the class of schema mappings we are interested in. We will talk about optimization potential and boundaries of computability in Section 6.

The notions of equivalence discussed up to now were primarily concerned with the two crucial tasks of data exchange and query answering. But beyond that, the area of **schema mapping management** [3, 4] poses quite different challenges: Operators of schema mapping management allow one to e.g. *invert* schema mappings or *extract* the essential parts of mappings.

The task of finding useful notions of equivalence between schema mappings for the purposes of schema mapping management was taken on by Arenas, Pérez, Reutter and Riveros in [1]. There they introduced notions of

- **equivalence in terms of information transfer**: schema mappings, which transfer the same amount of information are seen as equivalent. This notion has two variants, transferring source information and covering target information.

We will see that important operators of schema mapping management can be characterized using these equivalence notions and corresponding order relations in Section 7.

Up to now, we have talked about reasoning about mappings in a very strict sense. In the broad sense, *reasoning* about schema mappings covers a number of tasks related to working with schema mappings. When understanding and designing mappings, questions such as "What is this schema mapping doing?" and "Why is this schema mapping not doing what is expected?" are some of the first that are asked. We will give a glimpse at such reasoning tasks in a broader sense, like *analyzing* and *debugging* schema mappings, in Section 8.

1.1 Organization

In Section 2, we will introduce the necessary concepts. The main parts of this chapter are:

- **Concepts**: Where we will introduce notions of *equivalence* in Section 3 and then continue to discuss notions of *optimality* in Section 4.
- **Applications**: Where we will talk about optimization under logical equivalence in Section 5. After that, we look at the boundaries of computability in Section 6. We will finish by discussing applications to schema mapping management in Section 7.
- **Reasoning in the Broad Sense**: Where we will look at analyzing and debugging schema mappings in Section 8.

We finish this chapter with a conclusion and outlook in Section 9.

2 Preliminaries

In these preliminaries, we will first introduce database *schemas* and the relationships such as *homomorphisms* that may exist between database instances. Building upon that, we will define *schema mappings* and *solutions* to problems concerning schema mappings. After that, we describe the logical formalisms, called *dependencies*, on which schema mappings can be based. We conclude this section by introducing an algorithm called *the chase*.

2.1 Schemas

A **schema** $R = \{R_1, \ldots, R_n\}$ is a set of relation symbols R_i. Each relation symbol R_i has a fixed arity. An *instance* I over a schema R associates a relation R_i^I to each relation symbol in R. We call a relation symbol, together with some position an *attribute*. Sometimes, we associate a name to such an attribute. If $\vec{v} \in R_i^I$, we call $\vec{v}$ a *tuple* of R_i in I and also say that the *atom* $R_i(\vec{v})$ is contained in I. Instances in the context of this chapter are always considered to be finite. If the meaning is clear, we will not distinguish between syntax and semantics, e.g. relation symbols and relations. For two instances I, J, we write $I \subseteq J$ to say that the set of atoms contained in I is a subset of the set of atoms contained in J.

The **domain** of an instance I consists of two types of values, *constants* and *variables*. We write $\mathsf{dom}(I)$ for the domain, $\mathsf{const}(I)$ for the constants and $\mathsf{var}(I)$ for the variables. Variables are also called *labeled nulls* or *marked nulls*. We assume that $\mathsf{dom}(I) = \mathsf{var}(I) \cup \mathsf{const}(I)$ and $\mathsf{var}(I) \cap \mathsf{const}(I) = \emptyset$. An instance is called *ground*, if $\mathsf{var}(I) = \emptyset$. Instances are considered ground, unless specifically noted otherwise. In the same way as for instances, we can also refer to the domain, variables and constants of an atom, and speak of ground atoms. We usually denote labeled nulls by italic font (x) and constant symbols by sans-serif font (a).

Let $S = \{S_1, \ldots, S_n\}$ and $T = \{T_1, \ldots, T_m\}$ be schemas with no relation symbols in common. We write (S, T) to denote the combined schema $\{S_1, \ldots, S_n, T_1, \ldots, T_m\}$. If I is an instance of S and J is an instance of T, then (I, J) denotes the instance of the schema (S, T), consisting of the combined relations.

Let I, I' be instances. A **substitution** σ is a function $\mathsf{dom}(I) \to \mathsf{dom}(I')$ which replaces variables by constants or variables, but leaves constants unchanged, i.e., for all $c \in \mathsf{const}(I)$ it holds that $\sigma(c) = c$. We write $\sigma = [x_1 \mapsto a_1, \ldots, x_n \mapsto a_n]$ if σ maps $x_i \in \mathsf{var}(I)$ to $a_i \in \mathsf{dom}(I)$ and for all $v \in \mathsf{dom}(I)$ not in $\{x_1, \ldots, x_n\}$, $\sigma(v) = v$.

A **homomorphism** $h\colon I \to I'$ is a substitution $\mathsf{dom}(I) \to \mathsf{dom}(I')$ (i.e. leaves constants unchanged) and for all atoms $R(\vec{x})$ it holds that $R(\vec{x}) \in I$ implies $R(h(\vec{x})) \in I'$. If there exists such an h, we write $I \to I'$. We say that I and I' are *homomorphically equivalent*, denoted $I \leftrightarrow J$, iff $I \to I'$ and $I' \to I$. If $I \to I'$ but not in the other direction, I is called *more general* than I', and I' is called *more specific* than I.

A homomorphism $h : I \to I'$ is called an *isomorphism*, iff h^{-1} is defined, and is a homomorphism from I' to I. If such an isomorphism exists, we write $I \cong I'$ and say that I and I' are *isomorphic*. A homomorphism $h : I \to I$ is called an *endomorphism*. An endomorphism is *proper* if it reduces the domain, that is, it is a surjective function (i.e. onto).

An instance $I^* \subseteq I$ is called a **core** of I , if $I \to I^*$ and I^* cannot be reduced by a proper endomorphism. That is, there is no $I' \subset I^*$ such that $I \to I'$. Cores have several important properties. The core is unique up to isomorphism, i.e. if I' and I'' are cores of I, then $I' \cong I''$. Therefore, we may talk about *the* core of I and refer to it as $\mathsf{core}(I)$. Furthermore, for instances I and I' it holds that $I \leftrightarrow I'$ iff $\mathsf{core}(I) \cong \mathsf{core}(I')$.

2.2 Schema mappings

We now introduce *schema mappings*, which specify the relationship between schemas. Based on the *data-exchange problem*, we will then explore specific instances called *solutions*, *universal solutions* and the *core of the universal solutions.* These are three of the key notions for working with schema mappings.

A **schema mapping** $\mathcal{M} = (S, T, \Sigma)$ is given by a *source schema* S, a *target schema* T, and a set Σ of dependencies over S and T in some logical formalism. An instance (I, J) of $\mathcal{M}$ is an instance of the schema (S, T), for which $(I, J) \models \Sigma$ holds. The instance I is called the *source instance* and is usually ground, whereas J is called the *target instance* and may contain variables. If the source schema S and the target schema T are clear, or the specific schema is not of primary interest, we will identify a schema mapping $\mathcal{M} = (S, T, \Sigma)$ with the set Σ of dependencies. We sometimes refer to schema mappings just as mappings.

A target instance J is called a **solution** *for I under $\mathcal{M}$* if $(I, J) \models \Sigma$. The set of all solutions for I under $\mathcal{M}$ is denoted by $\mathsf{Sol}(I, \mathcal{M})$.

Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and I a source instance. A target instance J is a **universal solution** for I under $\mathcal{M}$ iff it is a solution for I under $\mathcal{M}$ and for all $J' \in \mathsf{Sol}(I, \mathcal{M})$ we have that $J \to J'$. The set of all universal solutions is denoted by $\mathsf{UnivSol}(I, \mathcal{M})$. An important property of universal solutions is that if J and J' are universal solutions for a source instance I under $\mathcal{M}$, they are homomorphically equivalent, i.e. $J \leftrightarrow J'$. Therefore the cores of J and J' are isomorphic, that is $\mathsf{core}(J) \cong \mathsf{core}(J')$ (cf. [10]).

The **core of the universal solutions** $\mathsf{core}(I, \mathcal{M})$ is given by $\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(J)$ for any $J \in \mathsf{UnivSol}(I, \mathcal{M})$. Any universal solution J can be taken for computing the core, since the core of homomorphically equivalent instances is unique up to isomorphism. Note that the core of the universal solutions $\mathsf{core}(I, \mathcal{M})$ need not necessarily be a solution for I under $\mathcal{M}$.

2.3 Dependencies

Here we will focus on the logical formalisms called *dependencies* on which schema mappings are based. We will first define *embedded dependencies* and the important subtypes *tuple-generating* dependencies and *equality-generating* dependencies. These are all based on first-order logic. So after that, we will cover *second-order* dependencies. We conclude this section by a discussion of *conjunctive queries.*

An embedded **dependency** over a schema R is a first-order formula of the form

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

where φ is a conjunction of atoms over R called the *antecedent* and ψ is a conjunction of atoms over R and equalities called the *conclusion.* Furthermore, φ contains at least one atom, and each $x \in \vec{x}$ occurs at least once in φ.

The following notational convention will be adopted to save some space and ease reading. For dependencies, we will mostly omit universal quantifiers. All variables occurring in the antecedent are implicitly universally quantified. We will sometimes also omit existential quantifiers. All variables occurring just in conclusions are implicitly existentially quantified. Therefore, for the dependency $\forall \vec{x} \, (\varphi(\vec{x}) \to \exists \vec{y} \, \psi(\vec{x}, \vec{y}))$, we may write $\varphi(\vec{x}) \to \psi(\vec{x}, \vec{y})$.

A **tuple-generating dependency** (tgd) is an embedded dependency of the form

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

over a schema R where both φ and ψ are conjunctions of atoms over R. Tuple-generating dependencies can be viewed as generalizations of inclusion dependencies, in particular in the form of foreign-key constraints. However, tgds cannot express key constraints.

An **equality-generating dependency** (egd) is an embedded dependency of the form

$$\forall \vec{x}\, (\varphi(\vec{x}) \rightarrow x_i = x_j)$$

over a schema R where x_i and x_j are contained in $\vec{x}$. Equality-generating dependencies generalize functional dependencies. These are in particular used to express key constraints.

For a schema mapping $\mathcal{M} = (S, T, \Sigma)$, an embedded dependency $\phi(\vec{x}) \rightarrow \psi(\vec{x}, \vec{y})$ is called a **source-to-target dependency** (s-t dependency) if φ is defined over S and ψ over T. It is called a **target dependency** if both φ and ψ are defined over the T and called a *source dependency* if both are defined over S.

A **second-order tgd** (SO tgd) over source schema S and target schema T has the form

$$\exists \vec{f}\, (\tau_1 \wedge \ldots \wedge \tau_n) \qquad \text{where each } \tau_i \text{ has the form} \qquad \forall \vec{x}\, (\varphi(\vec{x}) \wedge \chi(\vec{x}) \rightarrow \psi(\vec{x}))$$

in which φ is conjunctions of atoms over S, ψ is a conjunction of atoms over T and χ is a conjunction of equalities. As values, atoms and equalities may contain function terms based on $\vec{f}$. That is, second-order tgds extend the notion of (first-order) s-t tgds by allowing existential quantification over function symbols. All variables from each $\vec{x}$ have to be safe. A variable is *safe*, if it occurs in the relational atoms of φ_i or is derived through equations or function applications from safe variables. Formal details can be found in [11].

By definition, it is clear that SO tgds are closed under conjunctions. A set of SO tgds can therefore be identified with a single SO tgd. The most important property of SO tgds is that they are also closed under composition [11].

Conjunctive queries. We conclude this section about logical formalisms by introducing *conjunctive queries*. They are important for one of the relaxed notions of equivalence that we are going to introduce.

A (Boolean) **conjunctive query** q over a schema R is a logical formula that has the form $\exists \vec{x}\, (A_1 \wedge \ldots \wedge A_n)$ where each A_i is an atom over relation symbols from R, and all variables occurring in q are from $\vec{x}$.

The **certain answers** to a (Boolean) conjunctive query q over T on a source instance I under a schema mapping $\mathcal{M}$, assuming that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ are given as follows:

$$\mathsf{cert}(q, I, \mathcal{M}) = \bigcap_{J \in \mathsf{Sol}(I,\mathcal{M})} q(J)$$

The certain answers to a (Boolean) conjunctive query q on I under $\mathcal{M}$ can be obtained directly from a universal solution [8], that is $J \in \mathsf{UnivSol}(I, \mathcal{M})$ implies $\mathsf{cert}(q, I, \mathcal{M}) = \mathsf{ground}(q(J))$ where $\mathsf{ground}(q(j))$ denotes the ground atoms of $q(j)$, i.e., the atoms not containing variables. This concept can be naturally generalized to non-Boolean conjunctive queries.

2.4 The chase

The *chase* procedure [2] is an algorithm that computes universal solutions for a variety of schema mappings based on different dependency formalisms [8]. For a schema mapping $\mathcal{M} = (S, T, \Sigma)$ based on (finite sets of) s-t tgds, target tgds and egds, and SO tgds, the following holds: Given a source instance I, the chase procedure returns a universal solution J for I under $\mathcal{M}$, if it terminates and a universal solution exists. This J is called the **canonical universal solution** for I under $\mathcal{M}$ and written as $\mathsf{chase}(I, \mathcal{M})$.

The chase procedure computes a universal solution by a series of *chase steps*, based on a source instance and an initially empty target instance. In every step, a single dependency or multiple dependencies that are violated (i.e. not satisfied) by the current source and target instance are *applied* by adding further tuples to the target instance to fulfill those dependencies. We then say that these dependencies *fire*.

In the presence of target tgds, the chase does not always terminate. A sufficient condition for termination is that the set of target tgds is *weakly acyclic*. Intuitively, this criterion describes that the target tgds may not enter cycles which create new labeled nulls at each pass through the cycle. We refer to [19] and [8] for detailed definitions and further pointers.

We will use three variants of the chase in this chapter. They are in general based on s-t tgds, target tgds and target egds:

- the *standard* chase: in which for each step, one dependency that is violated is applied
- the *parallel* chase: in which for each step, all dependencies that are violated are applied
- the *SO tgd* chase: which is the chase procedure for SO tgds

More details and formal definitions of different variants of the chase procedure can be found in Chapter 1 of this book dedicated to the chase procedure.

2.5 Summary

This section introduced the most important preliminaries for the remainder of this chapter. A brief summary can be found in Figure 1.

A **schema mapping** $\mathcal{M} = (S, T, \Sigma)$ is given by

- a source schema S
- a target schema T
- a set Σ of dependencies over S and T in some logical formalism

Important dependency formalisms are

- **tuple-generating dependencies** (tgds)
  $\forall \vec{x}\, (\varphi(\vec{x}) \to \exists \vec{y}\, \psi(\vec{x}, \vec{y}))$
- **equality-generating dependencies** (egds)
  $\forall \vec{x}\, (\varphi(\vec{x}) \to x_i = x_j)$
- **second-order tgds** (SO tgds)
  $\exists \vec{f}\, (\tau_1 \wedge \ldots \wedge \tau_n)$ where each τ_i has the form
  $\forall \vec{x}\, (\varphi(\vec{x}) \wedge \chi(\vec{x}) \to \psi(\vec{x}))$ and χ is a conjunction of equalities

Figure 1 Schema mappings and dependencies (short summary).

3 Equivalence

In this first part on *concepts*, we will introduce the central notions of *equivalence* (this section) and *optimality* (next section) for schema mappings. We therefore start by covering the fundamental notions of equivalence we can use to compare schema mappings. We first treat

- **logical equivalence** between schema mappings, which equates schema mappings that are satisfied by the same instances.

Building upon that, we look at *relaxed notions of equivalence* introduced by Fagin et al. [9]. These notions characterize schema mappings that are not necessarily logically equivalent, but nevertheless indistinguishable for a variety of purposes. The first such relaxation is

- **data-exchange equivalence** (DE-equivalence), which equates schema mappings that exhibit the same behavior for data exchange. After that, we discuss
- **conjunctive-query equivalence** (CQ-equivalence), which equates schema mappings that yield the same certain answers to conjunctive queries.

If we need to compare schema mappings which differ e.g. in their target schemas, the amount of source information transferred becomes important, independently of how exactly this information is represented in the target instance [1]. This gives rise to

- **equivalence w.r.t. source information transferred** (S-equivalence). The corresponding notion for differing source schemas is
- **equivalence w.r.t. target information covered** (T-equivalence) based on the amount of target information that can be reconstructed by the schema mappings.

3.1 Notions of equivalence

We will now motivate and define these notions of equivalence. We will always start with an example, seeing why the respective notion naturally arises, and after that formally define that notion. Let us start by looking at such an example.

▶ **Example 1.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependency:

- $P(x, y) \wedge P(z, y) \rightarrow Q(x, y)$

This tgd is very similar to a simple copy tgd from relation P to Q. However, the additional conjunct $P(z, y)$ in the antecedent seems superfluous. Indeed, since the variable z does not occur anywhere else in the dependency, and $P(z, y)$ is satisfied whenever $P(x, y)$ is satisfied, it is easy to see that the conjunct could just be left out.

To be more precise, the schema mapping $\mathcal{M}$ and the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by the dependency

- $P(x, y) \rightarrow Q(x, y)$

are satisfied by exactly the **same pairs of instances**. ◀

In the previous example, we saw *logical equivalence* at work. This is the most natural notion to start with for reasoning about schema mappings, since our schema mappings are based on dependencies given in a logical formalism.

▶ **Definition 2.** Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent* if for every source instance I and every target instance J,

$$(I, J) \models \Sigma \Leftrightarrow (I, J) \models \Sigma'$$

We denote logical equivalence by $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$. ◀

To avoid confusion, we do not use $\equiv$ without subscript in this chapter. The following formalization in terms of solutions for I under $\mathcal{M}$ characterizes the same notion. Two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are *logically equivalent*, if for every source instance I, it holds that

$$\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}')$$

This characterization follows immediately from the definition of solutions. The use of data exchange terminology for describing logical equivalence will be beneficial as we go on.

▶ **Example 3.** Over the source schema $S = \{P\}$ and target schema $T = \{Q, R\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, y)$
- $R(x, y) \to R(x, x)$

Let us look at the result of data exchange under this schema mapping. We consider the source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$ and compute the chase result $J = \mathsf{chase}(I, \mathcal{M}) = \{Q(\mathsf{a}, \mathsf{b})\}$. During this chase, the second dependency never fires. Even more striking, there is no universal solution for any I under $\mathcal{M}$ which ever materializes a tuple of R.

So, for the purposes of data exchange which is usually concerned with universal solutions, we would like to simplify $\mathcal{M}$ into the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by the dependency

- $P(x, y) \to Q(x, y)$

Unfortunately, $\mathcal{M}$ and $\mathcal{M}'$ are not logically equivalent, they do not have the same solutions for every source instance. Consider, for the source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, the solution $J = \{Q(\mathsf{a}, \mathsf{b}), R(\mathsf{a}, \mathsf{b})\}$. This clearly violates $\mathcal{M}$, since the tuple $R(\mathsf{a}, \mathsf{a})$ would be required by the second dependency.

But, as we have seen before, for the purposes of data exchange, the schema mappings are "just as good". To be more precise, the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ have the **same universal solutions** for all source instances. ◀

The previous example motivates the introduction of the first relaxed notion of equivalence introduced by Fagin et al. [9], *data-exchange equivalence* (DE-equivalence), which does not distinguish schema mappings which behave in the same way for the purposes of data exchange, or more formally:

▶ **Definition 4.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *data-exchange equivalent*, if for every source instance I,

$$\mathsf{UnivSol}(I, \mathcal{M}) = \mathsf{UnivSol}(I, \mathcal{M}')$$

We denote data-exchange equivalence by $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$. ◀

Since for logical equivalence *all solutions* coincide, and for data-exchange equivalence the *universal solutions* coincide, it is clear that from $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$, it follows that $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ for all schema mappings. With that, it is appropriate to talk about a *relaxation* of logical equivalence. Also, in Example 3, we have already seen that this relaxation is proper (i.e. both notions are distinct) for schema mappings based on s-t tgds and target tgds.

▶ **Example 5.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to Q(x, x)$

Now, compared to the previous example, the source-to-target dependency is not anymore a simple copy tgd, and the relation symbol occurring in the target dependency actually also occurs in the source-to-target one. Can we, as in the previous example, simply remove the second dependency, gaining the schema mapping $\mathcal{M}' = (S, T, \Sigma')$ given by

- $P(x, y) \to Q(x, y)$

while still upholding data-exchange equivalence? It is now a bit more subtle to see why $\mathcal{M}$ and $\mathcal{M}'$ do not have the same universal solutions. The more so as for the source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, both schema mappings have the same canonical universal solution $J = \mathsf{chase}(I, \mathcal{M}) = \mathsf{chase}(I, \mathcal{M}') = \{Q(\mathsf{a}, \mathsf{a})\}$.

But now consider $J' = \{Q(\mathsf{a}, \mathsf{a}), Q(u, v)\}$ where u and v are variables. It is clear that J' is still a universal solution under $\mathcal{M}'$, since it is a solution under $\mathcal{M}'$, and J' maps to the universal solution J through the homomorphism $[u \mapsto \mathsf{a}, v \mapsto \mathsf{a}]$. Yet to satisfy $\mathcal{M}$, since $u \neq v$, it would require the atom $Q(u, u)$ to be present. So J' is not a universal solution for I under $\mathcal{M}$.

This state of affairs is clearly unsatisfactory if we look at a conjunctive query like $\exists x, y\, Q(x, y)$. If we want the certain answer to this query, a "strange" universal solution like J' will not affect the result. The tuple $Q(u, v)$ from J' will not be contained in the certain answers, since it is not contained in J. Indeed, $\mathcal{M}$ and $\mathcal{M}'$ will yield the **same certain answers to conjunctive queries**. ◀

This example directly leads us to *conjunctive-query equivalence* (CQ-equivalence). It is based on the behavior of conjunctive queries, posed against the solutions of a schema mapping.

▶ **Definition 6.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings. $\mathcal{M}$ and $\mathcal{M}'$ are *conjunctive-query equivalent*, if for every source instance I and every conjunctive query q, either $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}') = \emptyset$ or

$$\mathsf{cert}(q, I, \mathcal{M}) = \mathsf{cert}(q, I, \mathcal{M}')$$

We denote conjunctive-query equivalence by $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$. ◀

By this definition, there can be two reasons for schema mappings to be CQ-equivalent. The first is that both schema mappings could have no solutions at all. In this case, we say that the schema mappings are CQ-equivalent. The other case is of course that there are solutions to both schema mappings, and the certain answers to queries against them coincide.

The original definition of CQ-equivalence given above is based on the certain answers to conjunctive queries. However, an alternative characterization is possible:

▶ **Proposition 7.** *[9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings such that the following holds.*

$$\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset \quad \textit{implies} \quad \mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$$

$\mathcal{M}$ and $\mathcal{M}'$ are conjunctive-query equivalent*, if for every source instance I, either* $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}') = \emptyset$ *or*

$$\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(I, \mathcal{M}')$$ ◀

A few points are of interest now. The first crucial question is for which classes of schema mappings it holds that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ implies $\mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$. That is, for which kinds of schema mappings is there always a universal solution whenever any solution exists. In [9], it is shown that a sufficient condition is to have schema mappings defined by s-t tgds, target egds, target tgds that have a terminating chase, as well as to SO tgds. In particular, leaving out target tgds or requiring the set of target tgds to be weakly acyclic guarantees a terminating chase and therefore the property that we require.

What is also clear now is that in this case, CQ-equivalence is in fact a relaxation of DE-equivalence and therefore of logical equivalence. This was not obvious for the characterization based on conjunctive queries. It is easy to see here though, since the core is based on some universal solution and by DE-equivalence all universal solutions coincide, so from $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ follows $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.

We have now arrived at a *hierarchy of schema-mapping equivalences* given by respective relaxation between logical equivalence, DE-equivalence and CQ-equivalence. This hierarchy itself holds for *all* classes of schema mappings. For the most important classes of schema mappings (where universal solutions exist, given that solutions exist), one can easily see this: given that *all* solutions coincide, the *universal* solutions coincide and from that follows that the *cores* of the universal solutions coincide.

▶ **Proposition 8.** *[9] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T, \Sigma')$ be two schema mappings*

$$\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}' \quad \Rightarrow \quad \mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}' \quad \Rightarrow \quad \mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$$ ◀

Also, we have seen in Examples 3 and 5 that this hierarchy is proper for schema mappings based on s-t tgds and target tgds. Later in this chapter, we will take a more detailed look at for which classes of schema mappings the hierarchy is proper, and for which it collapses.

We now have at hand two very natural relaxations of logical equivalence. Next we will explore what happens if we need to reason about schema mappings that have differing source or target schemas. They were introduced by Arenas et al. in [1].

▶ **Example 9.** Over the source schema $S = \{P\}$ and target schema $T = \{Q\}$, let the schema mapping $\mathcal{M} = (S, T, \Sigma)$ be given by the following dependencies:

- $P(x, y) \to Q(x, y)$

Now consider the slightly altered schema mapping $\mathcal{M}' = (S, T', \Sigma')$ for $T' = \{R\}$ given by

- $P(x, y) \to R(x, y, y)$

Since the target relations affected by the two schema mappings are different, it is clear that $\mathcal{M}$ and $\mathcal{M}'$ are neither logically, nor DE-, nor CQ-equivalent. But intuitively, these schema mappings are very similar.

In fact, through a schema mapping $\mathcal{N}$ based on $Q(x, y) \to R(x, y, y)$ we can "reconstruct" $\mathcal{M}'$ from $\mathcal{M}$ in the following sense: The composition of $\mathcal{M}$ and $\mathcal{N}$ yields $\mathcal{M}'$. In the same way, we can reconstruct $\mathcal{M}$ through the composition of $\mathcal{M}'$ and a schema mapping $\mathcal{N}'$ based on $R(x, y, y) \to Q(x, y)$.
In this way, we see that both schema mappings **transfer the same amount of source information**, in the sense that they are able to reconstruct the result of the other. ◀

▶ **Definition 10.** [1] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T', \Sigma')$ be two schema mappings. $\mathcal{M}'$ *transfers at least as much source information as* $\mathcal{M}$, written $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$, iff there exists a schema mapping $\mathcal{N}$ from T to T' s.t.

$$\mathcal{M} \circ \mathcal{N} \equiv_{\mathsf{log}} \mathcal{M}'$$

We say that $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent w.r.t. the source information transferred*, written $\mathcal{M} \equiv_{\mathsf{S}} \mathcal{M}'$, iff $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$ and $\mathcal{M}' \preceq_{\mathsf{S}} \mathcal{M}$. ◀

Given that we talked about schema mappings with differing target schemas, it is natural to ask about differing source schemas. The following definition mirrors the above one:

▶ **Definition 11.** [1] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S', T, \Sigma')$ be two schema mappings. $\mathcal{M}'$ *covers at least as much target information as* $\mathcal{M}$, written $\mathcal{M} \preceq_{\mathsf{T}} \mathcal{M}'$, iff there exists a schema mapping $\mathcal{N}$ from S to S' s.t.

$$\mathcal{N} \circ \mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$$

We say that $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent w.r.t. the target information covered*, written $\mathcal{M} \equiv_{\mathsf{T}} \mathcal{M}'$, iff $\mathcal{M} \preceq_{\mathsf{T}} \mathcal{M}'$ and $\mathcal{M}' \preceq_{\mathsf{T}} \mathcal{M}$. ◀

Note that while the names of the ordering relations are given in this way in [1], the corresponding equivalence relations are originally used without reference to a specific name.

3.2 Summary

In this section, we introduced notions of equivalence for schema mappings and showed through examples how they naturally arise when working with schema mappings.

We started with *logical equivalence* and then introduced *relaxed notions of equivalence*: notions of equivalence which do not distinguish between mappings which behave the same for a given purpose. We discussed *data-exchange equivalence* and *conjunctive-query equivalence*. After that, we looked at two notions of *equivalence in terms of information transfer*.

In total, for schema mappings where the existence of solutions implies the existence of universal solutions, the definitions are for reference summarized in Figure 2.

Two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are

- **logically equivalent** ($\equiv_{\mathsf{log}}$)
  iff for all I we have $\mathsf{Sol}(I, \mathcal{M}) = \mathsf{Sol}(I, \mathcal{M}')$
- **data-exchange equivalent** ($\equiv_{\mathsf{DE}}$)
  iff for all I we have $\mathsf{UnivSol}(I, \mathcal{M}) = \mathsf{UnivSol}(I, \mathcal{M}')$
- **conjunctive-query equivalent** ($\equiv_{\mathsf{CQ}}$)
  iff* for all I we have $\mathsf{core}(I, \mathcal{M}) = \mathsf{core}(I, \mathcal{M}')$
- **equivalent w.r.t. source information transferred** ($\equiv_{\mathsf{S}}$)
  iff there exist $\mathcal{N}, \mathcal{N}'$ s.t. $\mathcal{M} \circ \mathcal{N} \equiv_{\mathsf{log}} \mathcal{M}'$ and $\mathcal{M}' \circ \mathcal{N}' \equiv_{\mathsf{log}} \mathcal{M}$
- **equivalent w.r.t. target information covered** ($\equiv_{\mathsf{T}}$)
  iff there exist $\mathcal{N}, \mathcal{N}'$ s.t. $\mathcal{N} \circ \mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ and $\mathcal{N}' \circ \mathcal{M}' \equiv_{\mathsf{log}} \mathcal{M}$

Figure 2 Notions of equivalence (*assuming universal solutions exist in case solutions exist).

4 Optimality

Given that we can reason about schema mappings that are equivalent according to a variety of notions as introduced in the previous section, there is a natural next task at hand: Finding a schema mapping that is "best" among those equivalent mappings.

In this section, we therefore discuss a number of optimality criteria. Mostly, we will talk about notions of "minimality" or "redundancy". These give rise to decision problems, i.e. identifying if a given schema mapping is minimal or non-redundant among equivalent schema mappings. They of course also induce optimization problems in the sense of actually finding such minimal or non-redundant schema mappings.

This section starts with the most basic optimality criteria, like subset- and cardinality-minimality. There, our main concern will be understanding how they affect schema mappings. But there are also quite intricate optimality criteria that we will talk about.

We thus start with some of the most basic optimality criteria (formal definitions follow later):

- **σ-redundancy**: By that we mean detecting if some specific dependency σ is redundant, i.e. could be left out while still yielding an equivalent schema mapping. Closely related is
- **subset-minimality**: That is, finding a minimal subset of dependencies. In other words, that set should contain no dependency that is redundant. A natural next step is
- **cardinality-minimality**: Finding a schema mapping that uses the minimum number of dependencies possible.

The previous three notions were concerned with schema mappings at the level of dependencies, but did not look inside of those dependencies. In [14], further criteria were presented that are concerned with internal characteristics of the given dependencies. There is, given in slightly generalized form:

- **antecedent-minimality**: It is concerned with the total number of atoms in antecedents. Together with cardinality-minimality, this aims at reducing the computational cost of the joins computed by the chase. The complementary notion is
- **conclusion-minimality**: Minimizing the total number of atoms in the conclusions. Besides the number of atoms, we can also consider
- **variable-minimality**: It is based on minimizing the total number of existentially quantified variables. This is of course related to the number of labeled nulls introduced during the chase.

The previously mentioned optimality criteria were all syntactically defined, which is important for the computational cost of the chase or similar procedures. Still, there are interesting semantic optimality criteria that are not based on the dependencies, but on the mapping seen as a binary relation between source and target instances. The following semantic criteria were introduced in [1] w.r.t. specific notions of equivalence:

- **target-redundancy**: Is there an equivalent schema mapping that "uses fewer target instances" (in the sense that the range of the optimized schema mapping is a subset of the range of the original one). The complementary notions is
- **source-redundancy**: Is there an equivalent schema mapping with a subset of source instances. We will give formal definitions of all notions later in this section.

For all of the criteria, two things need to be fixed: First, what is the *notion of equivalence* we are talking about? Secondly, what is the class of schema mappings that we allow for the desired optimized mapping? The interplay between notions of optimality, notions of equivalence and desired classes of schema mappings will turn out to be interesting.

4.1 Notions of optimality

We will now motivate and define these notions of optimality. Like in the previous section, we will always start with an example, seeing why the respective notion naturally arises, and after that formally define that notion. Let us start by looking at such an example.

▶ **Example 12.** Consider the schema mapping $\mathcal{M}$ given by the following dependencies:

- $P(x,y) \to Q(x,y)$ (σ_1)
- $P(x,x) \to Q(x,x)$ (σ_2)
- $P(x,y) \to R(y)$ (σ_3)
- $P(u,v) \to R(v)$ (σ_4)

Looking at dependency σ_2, it is clear that whenever a tuple is produced through σ_2, the same tuple is also produced by σ_1. That is, σ_2 is clearly redundant in $\mathcal{M}$ (w.r.t. logical equivalence). ◀

▶ **Definition 13.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\sigma \in \Sigma$. We say that $\mathcal{M}$ *is σ-redundant w.r.t. e-equivalence*, iff $\Sigma \setminus \{\sigma\} \equiv_e \Sigma$. ◀

▶ **Example 12 (ctd).** We have seen that $\mathcal{M}$ is σ_2-redundant, and we can remove it while retaining logical equivalence. It is also easy to see that $\mathcal{M}$ is both σ_3 and σ_4-redundant, since they are isomorphic "copies" of each other. Still, simply removing all σ-redundant dependencies will not yield a logically equivalent schema mapping: either σ_3 or σ_4 needs to be retained. Thus, the following schema mapping $\mathcal{M}'$ given by Σ' as

- $P(x, y) \to Q(x, y)$ (σ_1)
- $P(x, y) \to R(y)$ (σ_3)

is a minimal subset of Σ s.t. $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$. ◀

▶ **Definition 14.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\sigma \in \Sigma$. We say that $\mathcal{M}$ *is subset-minimal w.r.t. e-equivalence*, iff there is no schema mapping $\mathcal{M}' = (S, T, \Sigma')$ s.t. $\Sigma' \subset \Sigma$ and $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

▶ **Example 12 (ctd).** Somehow, the schema mapping $\mathcal{M}'$ is still not completely satisfactory regarding the number of dependencies. If we talk about *subsets*, $\mathcal{M}'$ is clearly the best we can do, but if we allow *arbitrary* dependencies, we can define $\mathcal{M}''$ based on

- $P(x, y) \to Q(x, y) \land R(y)$ (σ_5)

This clearly has the minimum cardinality among all logically equivalent schema mappings. ◀

In the previous example, we have seen that $\mathcal{M}''$, is cardinality minimal among schema mappings based on arbitrary dependencies. If we only look at schema mappings based on GAV dependencies (which restrict tgds by allowing only a single atom in the conclusion), we see that $\mathcal{M}'$ is cardinality minimal. This motivates the following definition relative to the class of schema mappings:

▶ **Definition 15.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. We say that $\mathcal{M}$ *is cardinality-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $|\Sigma'| < |\Sigma|$ and $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

Up to now, we have looked only at the dependencies themselves. We will now look inside of them to find additional ways to optimize these schema mappings:

▶ **Example 16.** Consider the schema mapping $\mathcal{M}$ given by the following dependency:

- $P(x, y) \land P(u, v) \to Q(x, y)$

It is clear that we could just as well leave out the atom $P(u, v)$ in the antecedent, thus getting the schema mapping $\mathcal{M}'$ based on

- $P(x, y) \to Q(x, y)$

which has the minimum total number of atoms in the antecedent (based on all schema mappings that are logically equivalent). ◀

This motivates the following definition. Note that we are talking about the *total* number of atoms over all dependencies here, not the *maximum* number over all dependencies.

▶ **Definition 17.** [14] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of sch. mappings. We say that $\mathcal{M}$ *is antecedent-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{AntSize}(\Sigma') < \mathsf{AntSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{AntSize}$ denotes the total number of atoms in antecedents. ◀

▶ **Example 18.** Consider the schema mapping $\mathcal{M}$ given by the following dependency:

- $P(x, y) \to \exists z (Q(x, y) \land Q(x, z))$

There are two dimensions in the conclusion that we can measure: the number of *atoms*, and the number of *existential variables* that we use. The following mapping $\mathcal{M}'$

- $P(x, y) \to Q(x, y)$

uses both the minimum total number of atoms in the conclusion, as well as the minimum total number of existentially quantified variables. ◀

▶ **Definition 19.** [14] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of sch. mappings:

- $\mathcal{M}$ *is conclusion-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no schema mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{ConSize}(\Sigma') < \mathsf{ConSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{ConSize}$ denotes the total number of atoms in conclusions.
- $\mathcal{M}$ *is variable-minimal w.r.t. e-equivalence among $\mathcal{C}$-schema mappings*, iff there is no schema mapping $\mathcal{M}' = (S', T', \Sigma')$ in $\mathcal{C}$ s.t. $\mathsf{VarSize}(\Sigma') < \mathsf{VarSize}(\Sigma)$ and $\mathcal{M} \equiv_e \mathcal{M}'$, where $\mathsf{VarSize}$ denotes the total number of existentially quantified variables. ◀

We can now talk about schema mappings based on their dependencies opaquely, and we can look inside of those dependencies based on the number of atoms and existentially quantified variables. For many tasks in data exchange and data integration, above all for the chase procedure, it can be argued that it is desirable to find schema mappings which are minimal under some, if not all of those criteria.

Still there is another, semantical, point of view in which such a schema mapping can still be redundant, and it will have important applications later on:

▶ **Example 20.** Consider the schema mapping $\mathcal{M}$ given by the dependency:

- $P(x, y) \to Q(x, x)$

Intuitively, this schema mapping "wastes space" compared to a schema mapping based on e.g. $P(x, y) \to R(x)$ by storing each source value x twice in the target.

In a more precise way, $\mathcal{M}$ is redundant in the following sense: Given source instance $I = \{P(\mathsf{a}, \mathsf{b})\}$, the canonical universal solution is $J = \{Q(\mathsf{a}, \mathsf{a}))\}$. But there is also another possible solution $J' = \{Q(\mathsf{a}, \mathsf{a})), Q(\mathsf{a}, \mathsf{b}))\}$ with $J \subset J'$. And indeed, we can find another schema mapping $\mathcal{M}'$ which has J as a solution, but not J':

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to x = y$

Clearly $\mathcal{M} \equiv_{\mathsf{S}} \mathcal{M}'$, that is, they transfer the same amount of source information (incidentally, they are also CQ-equivalent). In total, we have two mappings $\mathcal{M}$ and $\mathcal{M}'$, both are equivalent w.r.t. source information transferred, but one has a strict subset of solutions for I. ◀

▶ **Definition 21.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. $\mathcal{M}$ is *target-redundant w.r.t. e-equivalence among $\mathcal{C}$ schema mappings*, iff there is a target instance $J' \in \{J \mid (I, J) \in \mathcal{M}\}$ s.t. for $\mathcal{M}' = \{(I, J) \in \mathcal{M} \mid J \neq J'\}$ holds $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

The preceding definition was originally given in [1] w.r.t. S-equivalence. As we will later see, this is also the way it is commonly used and if no other notion of equivalence is explicitly mentioned, we assume this notion as default.

As an important remark, note that this definition does not necessarily talk about schema mappings "wasting space" *inside* the target instances. In particular, both schema mappings $\mathcal{M}$ and $\mathcal{M}'$ in the previous example store the value x twice in the target. So in this sense, they both waste space, even though one is target redundant and the other is not (w.r.t. S-equivalence among all schema mappings). The point is that $\mathcal{M}$ wastes target *instances*, since a subset of those would suffice.

Let us conclude this section by defining the natural counterpart to target-redundancy: source-redundancy. It is commonly used w.r.t. T-equivalence.

▶ **Definition 22.** Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping and $\mathcal{C}$ a class of schema mappings. $\mathcal{M}$ is *source-redundant w.r.t. e-equivalence among $\mathcal{C}$ schema mappings*, iff there is a source instance $I' \in \{I \mid (I, J) \in \mathcal{M}\}$ s.t. for $\mathcal{M}' = \{(I, J) \in \mathcal{M} \mid I \neq I'\}$ holds $\mathcal{M} \equiv_e \mathcal{M}'$. ◀

4.2 Summary

We now have means to talk about non-redundant and minimal schema mappings in a variety of manners: We can have schema mappings that are non-redundant in their syntactic representation (dependencies, atoms, variables) and their semantic extension (source- and target instances). For reference, we informally summarize the definitions in Figure 3.

The following criteria are given w.r.t. a notion of equivalence and a class of mappings:

- σ**-redundant**: a *specific* dependency σ could be left out
- **subset-minimal**: no dependency could be left out
- **cardinality-minimal**: the number of dependencies is minimal

- **antecedent-minimal**: the *total* number of atoms in antecedents is minimal
- **conclusion-minimal**: the *total* number of atoms in conclusions is minimal
- **variable-minimal** the *total* number of *existentially* quantified variables is minimal

- **target-redundant**: a target instance could be left out
- **source-redundant**: a source instance could be left out

All criteria are usually w.r.t. logical equivalence, the exceptions are that usually target-redundancy is w.r.t. S-equivalence, source-redundancy w.r.t. T-equivalence.

Figure 3 Notions of optimality for a schema mapping (informal summary).

5 Normalization and optimization for logical equivalence

In the previous sections, our main goal was to develop the relevant notions of equivalence and optimality for reasoning about schema mappings. What was left open was *how* to use these notions for actual reasoning, that is, the question of algorithms and complexity.

In this section, we will talk about reasoning under logical equivalence. The major result we will cover here, presented by Gottlob et al. in [14], is that there is an algorithm which transforms schema mappings based on s-t tgds into an optimal form in the following sense:
- it is a *unique normal form* (up to variable renaming), and
- the mapping is cardinality-, antecedent-, conclusion- and variable-minimal among all *split-reduced* schema mappings.

This form can be computed in polynomial time if the length of each dependency is bounded by a constant. What exactly split-reduced schema mappings are will be our next topic:

5.1 Finding optimal split-reduced schema mappings

Let us see why optimality among *all* mappings based on s-t tgds is not always desirable:

▶ **Example 23.** Over the source schema $S = \{L\}$ and the target schema $T = \{C, E\}$, let the schema mapping $\mathcal{M}$ be given by the following dependencies:
- $L(x_1, x_2, x_3) \rightarrow \exists y\, C(x_1, y)$ (σ_1)
- $L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \rightarrow E(x_1, x_4)$ (σ_2)

It is easy to see that the antecedents of σ_1 is fulfilled whenever the antecedent of σ_2 is fulfilled. Therefore, clearly $\mathcal{M}$ is logically equivalent to $\mathcal{M}'$ based on the following dependency
- $L(x_1, x_2, x_3) \wedge L(x_4, x_2, x_5) \rightarrow \exists y(C(x_1, y) \wedge E(x_1, x_4))$ (σ_3)

which contains both conclusions in a single dependency.

Now let us compute the canonical universal solutions to both of these schema mappings for source instance $I = \{L(\mathsf{a}, \mathsf{b}, \mathsf{c}), L(\mathsf{d}, \mathsf{b}, \mathsf{f})\}$ (which is just the antecedent of σ_3 with variables replaced by distinct constants). Let $J = \mathsf{chase}(I, \mathcal{M})$ and $J' = \mathsf{chase}(I, \mathcal{M}')$.

Considering the C-atoms, for J we have $C(\mathsf{a}, y_1)$ and $C(\mathsf{d}, y_2)$, since there are two possible ways to instantiate the antecedent of σ_1. But for J', since for σ_3 there are actually four possible ways to instantiate the antecedent, we additionally have $C(\mathsf{a}, y_3)$ and $C(\mathsf{d}, y_4)$. ◀

In the preceding example, we saw that, while $\mathcal{M}'$ is the cardinality-, antecedent- and conclusion-minimal mapping among *all* schema mappings, this leads to a quadratic blowup of the size of the canonical universal solution compared to $\mathcal{M}$.

However, σ_3 of $\mathcal{M}'$ has a problematic property: The atoms in the conclusion are actually not related to each other, they could very well be formulated in separate dependencies. This was the reason for the quadratic blowup. Therefore the following was defined:

▶ **Definition 24.** [14] A schema mapping $\mathcal{M} = (S, T, \Sigma)$ consisting of s-t tgds is *split-reduced*, if there is no logically equivalent mapping $\mathcal{M}' = (S, T, \Sigma')$ with $|\Sigma| > |\Sigma'|$ but $\mathsf{ConSize}(\Sigma) = \mathsf{ConSize}(\Sigma')$. ◀

Let us look at what this definition means: If we can, through "splitting up" a dependency – thus raising the number of dependencies – still have the same total size of the conclusions, then some conclusion atoms where not related to each other in a significant way. In other words, the dependencies are decomposed without raising the total size of the conclusions.
If we find a schema mapping among the class of *split-reduced* schema mappings that is minimal according to our chosen criteria, we get both a schema mapping that has good properties in terms of the dependencies (minimality) and good properties in the solution produced by the chase (some unnecessary blowup is avoided).

Note that one can view split-reducedness also as a derived optimality criterion like the ones discussed in the previous section (based on cardinality- and conclusion minimality). Let us now find, through an example, rules to rewrite a mapping into the optimal form we promised. For ease of reference, the rewrite rule numbers will match those in [14]:

▶ **Example 25** (based on [14]). Over the source schema $S = \{L\}$ and $T = \{P, Q, R\}$, let the mapping $\mathcal{M}$ be given by the following dependencies. For readability, all variables x_i are universally quantified and all variables y_i are existentially quantified. For the same reason, we use constant a within dependencies (avoidable by e.g. adding $A(\mathsf{a})$ to all antecedents).

- $L(x_1, x_2, x_3) \to P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a}) \wedge R(y_1, x_2, y_2)$ (σ_1)
- $L(x_1, x_1, x_1) \to P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \to P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2) \wedge R(x_2, y_4, x_3)$ (σ_3)

In this state, it is very hard for a human to make much sense out of this schema mapping without significant analysis. Let us therefore try to simplify it before trying to understand it.

A simple first rewriting is for σ_1: The last atom $R(y_1, x_2, y_2)$ is actually a more specific form of the second one $R(y_1, x_2, \mathsf{a})$ in the conclusion. Thus we can do the following:

Rule 1: Simplify the conclusion to its core
applied to: $L(x_1, x_2, x_3) \to P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a}) \wedge R(y_1, x_2, y_2)$ (σ_1)

So through the homomorphism $[y_2 \mapsto \mathsf{a}]$, we can drop the last atom arriving at the core of the conclusion. Before making things easier by dropping further atoms, let us try to split up the quite long dependency σ_3, to get a better overview (and reach a split-reduced form):

Rule 3: Split the dependency if possible

applied to: $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow$ (σ_3)
$P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2) \wedge R(x_2, y_4, x_3)$

We see that in the first three atoms of the conclusion, there are existentially quantified variables y_1 to y_3 intermingled and in the last one, there is only y_4. Indeed, we can split σ_4 in this way, yielding two dependencies, one with the conclusion $R(x_2, y_4, x_3)$ and one with the other three atoms. Let us look at the current state of the schema mapping after having applied those two rules:

- $L(x_1, x_2, x_3) \rightarrow P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a})$ (σ_1')
- $L(x_1, x_1, x_1) \rightarrow P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ_3')
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow R(x_2, y_4, x_3)$ (σ_4')

Now we look a bit further at the new rule σ_3'. Following the split, in the antecedent there occurs variable x_3, but it is never used in the conclusion. Therefore we can:

Rule 2: Simplify the antecedent to its core

applied to: $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ_3')

Easily, through the homomorphism $[x_3 \mapsto x_2]$, we can thereby drop the second atom of the antecedent.

Let us stay with the conclusion of this dependency. Take the first atom $P(x_1, y_2, y_1)$. It is clearly not implied by any of the other conclusion atoms, so Rule 1 – simplifying the conclusion to its core – will not help. But maybe it is produced by another dependency:

Rule 5: Remove atoms from the conclusion, if it they are implied

applied to: $L(x_1, x_2, x_2) \rightarrow P(x_1, y_2, y_1) \wedge Q(y_1, y_3, x_2) \wedge Q(\mathsf{a}, y_3, x_2)$ (σ_3'')

More formally, by *implied*, the following is meant: If dependency τ' is produced from τ by removing atoms from the conclusion, then the removed atoms are implied, if $(\Sigma \setminus \{\tau\}) \cup \{\tau'\}$ is logically equivalent to Σ. Indeed, looking at σ_1', we see that it produces the atom $P(x_1, y_1, \mathsf{a})$ under more general antecedents. This clearly implies our first atom $P(x_1, y_2, y_1)$, so we can drop that. Furthermore, we see that the second conclusion atom $Q(y_1, y_3, x_2)$ is implied by the last one $Q(\mathsf{a}, y_3, x_2)$ and we can also drop it.

Let us look at the one dependency we did not rewrite up to now, σ_2: The first conclusion atom $P(x_1, y_1, y_2)$ is a more specific case of $P(x_1, y_1, \mathsf{a})$ from σ_1', and the other atoms are also not very different from those implied by some dependencies. Let us check whether σ_2 is needed at all:

Rule 4: Remove the following dependency, if it is implied by other dependencies

applied to: $L(x_1, x_1, x_1) \rightarrow P(x_1, y_1, y_2) \wedge Q(y_2, y_3, x_1) \wedge R(y_1, x_1, y_2)$ (σ_2)

We see that the last atom $R(y_1, x_1, y_2)$ is implied by $R(y_1, x_2, \mathsf{a})$ of σ_1, and the remaining one $Q(y_2, y_3, x_1)$ by σ_3'''. So indeed, we can remove σ_2. Our mapping is now given as follows:

- $L(x_1, x_2, x_3) \rightarrow P(x_1, y_1, \mathsf{a}) \wedge R(y_1, x_2, \mathsf{a})$ (σ_1')
- $L(x_1, x_2, x_2) \rightarrow Q(\mathsf{a}, y_3, x_2)$ (σ_3''')
- $L(x_1, x_2, x_2) \wedge L(x_1, x_2, x_3) \rightarrow R(x_2, y_4, x_3)$ (σ_4')

It can be checked that this mapping is now minimal according to our optimality criteria. ◀

1. **Simplify the conclusion** to its core
2. **Simplify the antecedent** to its core
3. **Split** the dependency if possible
4. **Remove** the dependency, if it is implied by other dependencies
5. **Remove atoms** from the conclusion, if it they are implied by other dependencies

Figure 4 Rewrite rules for optimization and normalization (informal formulation).

Indeed, these rules are sufficient for achieving all four optimality criteria for mappings based on s-t tgds among split-reduced mappings. Even more interesting, they actually yield a unique normal form. Let us now summarize the rewriting system as defined by [14]:
Note that the first and second rules come down to core computations. If the length of each dependency is bounded by a constant, this can be done efficiently (cf. [13]). The last two items are implication tests, which are well known to be efficiently computable [2]. Splitting can also be efficiently performed. In total, the normal form can be computed in polynomial time if the length of each dependency is bounded by a constant [14].

5.2 Summary

In this section, we have seen how to compute an optimized unique normal form of a schema mapping based on s-t tgds. The resulting schema mapping is cardinality-, antecedent-, subset- and variable-minimal among all split-reduced schema mappings.

Through a quite complex extension, and a reformulation of what "split-reduced" should mean in the presence of egds, an optimized but not normalized normal form can be obtained for schema mappings based on s-t tgds and target egds [14].

6 Decidability of reasoning with relaxed notions of equivalence

In the previous section we have seen how to obtain optimized schema mappings under logical equivalence. However, as we have talked about in the beginning, logical equivalence is quite restrictive in the optimization potential it admits. In this section, we will therefore discuss the question of the computational properties of optimality under relaxed notions of equivalence.

We have seen that logical-, DE- and CQ-equivalence form a hierarchy where each notion might offer additional optimization potential compared to the one before. That is, while logical equivalence is the most restrictive, CQ-equivalence is the least restrictive.

A number of questions remained. The first one is:

- For which classes of schema mappings is this hierarchy proper, that is, for which schema mappings can we really gain **additional optimization potential**? The counterpart of this question is one about computability and complexity:
- If there is additional optimization potential, can we find algorithms for reasoning about them? In particular, can we construct **algorithms for finding optimal** schema mappings given various optimality criteria?

These are the questions that will guide this section. They will lead us to the boundaries of computability as explored by Fagin et al. [9] and Pichler et al. [19].

Before looking at questions of computability, let us first find out for which classes of schema mappings there is additional optimization possible. In other words, when is the hierarchy of logical-, DE- and CQ-equivalence strict, and when does it collapse?

6.1 Hierarchy or collapse

Let us first summarize what we have seen earlier in Section 3 when we introduced logical equivalence, DE-equivalence and CQ-equivalence:

- Logical equivalence, DE-equivalence and CQ-equivalence form a hierarchy (Proposition 8): $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}' \Rightarrow \mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}' \Rightarrow \mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.
- For schema mappings based on s-t tgds and target tgds, all three notions are distinct (through Examples 3 and 5 in Section 3).

Similar examples show that the three notions are different also for s-t tgds with target tgds [9] or target egds [14] as well as for SO tgds [9]. By contrast, we will now see that for schema mappings based on s-t tgds, the three notions actually coincide. We thus return to an example we have seen in a similar form before (Example 5). This time, it will allow us to find out something quite different:

▶ **Example 26.** For the source schema $S = \{P\}$ and the target schema $T = \{Q\}$ let the schema mapping $\mathcal{M}$ be defined by the following dependencies:

- $P(x, y) \to Q(x, x)$
- $Q(x, y) \to x = y$

Let us compare this to the schema mapping based on just the s-t tgd. That is, we define the schema mapping $\mathcal{M}'$ based on

- $P(x, y) \to Q(x, x)$

We have that the schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are CQ-equivalent, but not DE-equivalent. As an intuition for the CQ-equivalence, observe that the egd has no effect on the cores of the universal solutions, since the atoms contained in it already have the form $Q(x, x)$.

Now let $I = P(\mathsf{a}, \mathsf{a})$. The canonical universal solution, which in this case is also the core of the universal solutions, will be $J = \{Q(\mathsf{a}, \mathsf{a})\}$ for both $\mathcal{M}$ and $\mathcal{M}'$. But now consider $J' = \{Q(\mathsf{a}, \mathsf{a}), Q(u, v)\}$, for variables u and v. This instance is universal for both $\mathcal{M}$ and $\mathcal{M}'$, evidenced by the homomorphism $[u \mapsto a, v \mapsto a]$. Still, while J' is a universal solution for $\mathcal{M}'$, it is not even a solution for $\mathcal{M}$. ◀

Fagin et al. [9] identify this property as the key reason for CQ-equivalence and DE-equivalence to be distinct for a class of schema mappings: For two DE-equivalent schema mappings, there is a universal solution for one mapping that is not even a solution for the other mapping.

▶ **Definition 27.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping. $\mathcal{M}$ has *all the universal solutions*, if whenever

$$J \in \mathsf{UnivSol}(I, \mathcal{M}) \text{ and } J \leftrightarrow J', \text{ then } J' \in \mathsf{Sol}(I, \mathcal{M})$$ ◀

Recall that $\leftrightarrow$ denotes homomorphic equivalence in this context. That is, for this property, every instance homomorphically equivalent to a universal solution must also be a universal solution. Note that from $J' \in \mathsf{Sol}(I, \mathcal{M}')$ through homomorphic equivalence to J follows that $J' \in \mathsf{UnivSol}(I, \mathcal{M}')$. Given this definition, we then know that:

▶ **Proposition 28.** *[9] If $\mathcal{M}$ and $\mathcal{M}'$ have all the universal solutions, then $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ implies $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$.* ◀

That is, if all mappings in a class of schema mappings have all the universal solutions, then for this class DE- and CQ-equivalence coincide. Also, the following sufficient condition for this property is attained:

▶ **Definition 29.** [9] Let $\mathcal{M} = (S, T, \Sigma)$ be a schema mapping. $\mathcal{M}$ is *preserved under target homomorphisms*, if whenever

$$J \in \mathsf{Sol}(I, \mathcal{M}) \text{ and } J \rightarrow J', \text{ then } J' \in \mathsf{Sol}(I, \mathcal{M})$$ ◀

This property holds for schema mappings based on s-t tgds. Knowing that preservation under target homomorphism holds has the following important consequence:

▶ **Proposition 30.** *[9] If $\mathcal{M}$ and $\mathcal{M}'$ are*
- *both preserved under target homomorphisms, and*
- *both have that $\mathsf{Sol}(I, \mathcal{M}) \neq \emptyset$ implies $\mathsf{UnivSol}(I, \mathcal{M}) \neq \emptyset$ for all I*

then $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ iff $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ iff $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$. ◀

From this, since the properties hold for s-t tgds, we have that the three notions of logical-, DE- and CQ-equivalence coincide for mappings based on s-t tgds. Also, since the three notions are distinct for the classes of schema mappings
- based on s-t tgds and target tgds
- based on s-t tgds and target egds
- based on SO tgds

we know through the preceding theorem that they are *not* generally preserved under target homomorphisms for these classes. In total, if we allow target egds, target tgds or SO dependencies, then the relaxed notions of equivalence offer additional optimization potential.

6.2 Decidability of equivalence and optimization

First, let us note that for mappings based on s-t tgds (where logical, DE-, and CQ-equivalence coincide), all three notions are decidable. The decidability proof comes down to checking implication of dependencies by the chase (cf. e.g. [7]).

▶ **Proposition 31.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be schema mappings based on s-t tgds. Then it is decidable whether $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$, $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ and $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$.* ◀

So in this case, the problems are decidable, but there is no additional optimization power, since the notions coincide. Given that for a variety of schema mapping classes there is clearly additional optimization power using DE- and CQ- equivalence, we would like to use this power by appropriate algorithms for reasoning about them. In [9], the following general bounds are shown:

▶ **Theorem 32.** *[9] Given two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ based on* s-t tgds and a weakly acyclic set of target tgds
- *it is* decidable *whether $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$*
- *it is* undecidable *whether $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$*

The undecidability results holds even for copy s-t tgds and full target tgds. ◀

The decidability proof again comes down to checking implication of dependencies by the chase. The undecidability result is based on a reduction from Datalog equivalence (which is undecidable as shown in [20]), where target tgds are used to mirror recursive Datalog rules.

Further exploration of these bounds is done by Pichler et al. in [19], where schema mappings based on target egds and target tgds are considered under DE-equivalence in addition to CQ-equivalence. Also, various optimality criteria are discussed there.

▶ **Theorem 33.** *[19] For schema mappings $\mathcal{M}$ and $\mathcal{M}'$ based on* full s-t tgds and full target tgds*, or* s-t tgds and target egds*, the following problems are* undecidable

- $\mathcal{M} \equiv_{\mathsf{DE}} \mathcal{M}'$ *and* $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$
- *σ-redundancy of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence*
- *subset-minimality of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence*
- *cardinality-minimality of $\mathcal{M}$ w.r.t. DE- and CQ-equivalence* ◀

Thus leaving out target dependencies leads to a collapse of the hierarchy, hence no additional optimization power. Adding target dependencies leads to undecidability. These bounds leave one possibility open: If we require the target dependencies to be fixed, can we then optimize the s-t tgds further?

Towards this goal, the following connection between normalization of Section 5 and relaxed notions of equivalence was shown for s-t tgds and target tgds or egds. Here we consider source egds, which are simply egds defined over the source schema.

▶ **Theorem 34.** *[19] Let $\mathcal{M}$ and $\mathcal{M}'$ be schema mappings based on* s-t tgds and target tgds or egds*. Let $\Sigma = \Sigma_{st} \cup \Sigma_t$ the sets of s-t tgds resp. target dependencies of $\mathcal{M}$. Assume the same for Σ' and $\mathcal{M}'$.*
If $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$, then there exists a common set *Σ^*_s and Σ^*_{st} of* source egds resp. s-t tgds*, s.t.*

$$\Sigma \equiv_{\mathsf{log}} \Sigma^*_s \cup \Sigma^*_{st} \cup \Sigma_t \text{ and } \Sigma' \equiv_{\mathsf{log}} \Sigma^*_s \cup \Sigma^*_{st} \cup \Sigma'_t$$ ◀

So why is this theorem interesting? Assume that we have CQ-equivalent schema mappings whose target dependencies are logically equivalent. Given the previous theorem, we can normalize the source and s-t tgds upholding logical equivalence. But given that also the target dependencies are logically equivalent, the two schema mappings are altogether logically equivalent. This has the following consequence:

▶ **Theorem 35.** *[19] In the same setting as in Theorem 34. If $\Sigma_t \equiv_{\mathsf{log}} \Sigma'_t$, then*

$$\Sigma \equiv_{\mathsf{log}} \Sigma' \text{ iff } \Sigma \equiv_{\mathsf{DE}} \Sigma' \text{ iff } \Sigma \equiv_{\mathsf{CQ}} \Sigma'$$ ◀

This settles the question of whether optimization is possible if the target dependencies are fixed: There is no additional optimization power.
The third road to finding interesting decidable fragments is to look at special cases, e.g. mappings based on functional dependencies or inclusion dependencies. However, the following result weakens this hope for CQ-equivalence:

▶ **Theorem 36.** *[19] CQ-equivalence is* undecidable *for schema mappings based on s-t tgds and* at most one key dependency per target relation*.* ◀

Interestingly, the situation for DE-equivalence looks quite different:

▶ **Theorem 37.** *[19] DE-equivalence is* decidable *for schema mappings based on s-t tgds and weakly acyclic sets of* functional- and inclusion dependencies *as target dependencies.* ◀

These two results show our first disparity between the computational properties of DE- and CQ-equivalence.

We have now talked about various optimization tasks under both DE- and CQ-equivalence. From the point of schema mappings, we looked at those based on s-t tgds only, and at those with target egds or target tgds in addition to s-t tgds. What we still have not discussed are SO tgds. Here the following results are known:

▶ **Theorem 38.** *[12] Let $\mathcal{M}$ and $\mathcal{M}'$ be given by SO tgds. It is undecidable whether*

- $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$
- $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ *even if it is known that* $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$
- $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ *for mappings based on SO tgds and source key deps.* ◀

The proof of the first bullet is based on results about the existence of inverses from [1], about which we will talk a bit later in this chapter.

The previous theorem talks about equivalence between schema mappings based on SO tgds (showing undecidability) and before we talked about equivalence between schema mappings based on s-t tgds (yielding decidability). Recently, Fagin and Kolaitis [7] looked at equivalence where one mapping is based on SO tgds and the other one is based on s-t tgds:

▶ **Theorem 39.** *[7] Let $\mathcal{M}$ be given by SO tgds and $\mathcal{M}'$ be given by s-t tgds*

- *it is* undecidable *whether* $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$
- *it is* decidable *whether* $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ ◀

Interestingly, CQ-equivalence becomes decidable in this case, while it is undecidable for schema mappings based on SO tgds plus source key dependencies.

6.3 Equivalence to classes of schema mappings

In this final subsection, we will talk about the computational properties of deciding whether a mapping from some class of schema mappings is equivalent to a mapping in a more restricted class of schema mappings.

Results in this area were shown in [9] for the following problem: Given a schema mapping, under which conditions is it CQ-equivalent to a mapping consisting of s-t tgds?

We start with mappings based on full s-t tgds and full target tgds. For this, we need the following concept: A mapping has *bounded parallel chase* if there is a constant, such that for every source instance, the parallel chase needs at most that constant number of steps. Using this, we have that

▶ **Theorem 40.** *[9] Let $\mathcal{M}$ be a schema mapping based on* full s-t tgds and full target tgds. *There exists a schema mapping $\mathcal{M}'$ based on* full s-t tgds *with $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ iff $\mathcal{M}$ has bounded parallel chase.* ◀

The problem of finding such a schema mapping is undecidable [9]. We now look at mappings based on SO tgds. Again, we need a characterizing concept, but this time it is more complex:

▶ **Definition 41.** [9] The *Gaifman graph of facts G* of a target instance K is the graph whose nodes are the facts of K and there is an edge between two facts if they have a null in common. A *fact block* (f-block) of K is a connected component of G.

Mapping $\mathcal{M}$ has *bounded f-block size* if there is a constant such that $\mathsf{core}(I, \mathcal{M})$ has f-block size bounded by this constant for every source instance I. ◀

In [7], it was shown that deciding whether the f-block size of an SO tgd is bounded by a given number is equivalent to a problem we have already seen in the previous section: is a schema mapping based on s-t tgds equivalent to one based on SO tgds. Importantly, the notion of bounded f-block size characterizes CQ-equivalence of an SO tgd to s-t tgds:

▶ **Theorem 42.** *[9] Let $\mathcal{M}$ be a schema mapping based on an SO tgd. There exists a mapping $\mathcal{M}'$ based on s-t tgds with $\mathcal{M} \equiv_{\mathsf{CQ}} \mathcal{M}'$ iff $\mathcal{M}$ has bounded f-block size.* ◀

Note that in [9] a characterization is also given for schema mappings based on s-t tgds and target tgds with terminating chase.

Concerning schema mapping languages that can express only a subset of s-t tgds, results where shown by ten Cate and Kolaitis [21]. Let a LAV tgd be a tgd with a single atom on the left-hand side (this corresponds to what is called "extended LAV" in [7], compared to "strict LAV" which requires that all variables on the left-hand side occur just once).

▶ **Theorem 43.** *For a given schema mapping $\mathcal{M}$, deciding whether there exists a schema mapping $\mathcal{M}'$ such that $\mathcal{M} \equiv_{\mathsf{log}} \mathcal{M}'$ is* NP-complete *if*

- *$\mathcal{M}$ is given by* s-t tgds *and $\mathcal{M}'$ shall be definable by* full s-t tgds
- *$\mathcal{M}$ is given by* s-t tgds *and $\mathcal{M}'$ shall be definable by* LAV s-t tgds
- *$\mathcal{M}$ is given by* LAV s-t tgds *and $\mathcal{M}'$ shall be definable by* full s-t tgds

it is decidable in polynomial time *if*

- *$\mathcal{M}$ is given by* full s-t tgds *and $\mathcal{M}'$ shall be definable by* LAV s-t tgds ◀

6.4 Summary

In this section, we have looked into the power of logical-, DE- and CQ-equivalence for optimizing schema mappings. We have seen that for schema mappings based on s-t tgds only, all three notions of equivalence coincide and therefore admit no additional potential for optimization. We then looked at schema mappings based on target egds or target tgds in addition to s-t tgds, as well as SO tgds. We have seen that there clearly is additional potential for optimization using relaxed notions of equivalence.

However, unfortunately, most tasks are undecidable in general apart from logical equivalence for s-t tgds and weakly acyclic sets of target tgds. In particular, DE- and CQ-equivalence are undecidable for schema mappings based on s-t tgds and target tgds or target egds. For SO tgds, even logical equivalence is undecidable. We also looked at how to find mappings which are CQ-equivalent to more restricted classes of schema mappings.

Altogether, many of the boundaries are known, but decidable cases are still sparse. Still, the decidable special case for DE-equivalence and schema mappings with functional and inclusion dependencies shows an interesting disparity between DE- and CQ-equivalence. We summarize the results in Figure 5.

7 Equivalence and optimality for schema mapping management

In the previous two sections, we first discussed logical equivalence and its application to normalization. After that, we talked about DE- and CQ-equivalence and the boundaries of computability. In this section, we will talk about the notions of equivalence we have not covered up to this point: Schema mappings which are equivalent w.r.t. the source information transferred (S-equivalence) or equivalent w.r.t. the target information covered (T-equivalence). They were introduced and applied by Arenas et al. [1].

These notions have a slightly different character compared to logical-, DE- and CQ-equivalence. In particular, we can use them to compare schema mappings that have different source or target schemas. On the other hand, mappings which are e.g. S-equivalent may produce completely different target instances. Still, they guarantee that we can reconstruct the original target instance using some other schema mapping.

Our focus in this section will be on applications of S-equivalence and T-equivalence, in particular combined with the corresponding redundancy notions of source-redundancy and

Optimization potential

Logical-, DE- and CQ-equivalence **coincide** for
- mappings based on s-t tgds

They are **distinct** for
- mappings based on s-t tgds and target tgds or target egds
- mappings based on SO tgds

Boundaries of decidability

Logical equivalence is
- decidable for mappings based on s-t tgds and sets of weakly acyclic target tgds
- undecidable for mappings based on SO tgds

Data-exchange equivalence is
- undecidable for mappings based on s-t tgds, target tgds or target egds
- decidable for mappings with weakly acyclic sets of functional- and inclusion deps.

Conjunctive-query equivalence is
- undecidable for mappings based on s-t tgds, target tgds or target egds (even if restricted to a single key dependency per relation)
- undecidable for mappings based on SO tgds and source key dependencies
- decidable if one mapping is given by SO tgds, the other one by s-t tgds

Further results

Optimality is undecidable for mappings based on s-t tgds, target tgds or egds for
- σ-redundancy, subset-minimality, cardinality-minimality

CQ-equivalence to mappings based on s-t tgds is characterized for
- mappings based on target tgds with terminating chase
- mappings based on SO tgds

Figure 5 Optimization potential and boundaries of decidability.

target-redundancy. We will see that for the important area of characterizing the operators for schema mapping management, these notions of equivalence and optimality find natural applications. After that, we will briefly discuss some algorithmic properties.

7.1 Application to schema mapping management

We begin by talking about the *extract* operator of schema mapping management [18]. Though there are a number of possible characterizations, the intended meaning of the extract operator is the following: Given a schema mapping, find a new source schema that captures exactly the information that participates in it. Let us start by looking at an example.

▶ **Example 44** (based on [1]). Over the source schema $S = \{P, Q, R\}$ and the target schema $T = \{U, V, W\}$ let the schema mapping $\mathcal{M}$ be given by the following dependencies:
- $P(x, y) \to \exists u\, W(x, u) \wedge U(x, x)$ (σ_1)
- $P(x, y) \wedge R(y, z) \to \exists v\, V(x, y, v)$ (σ_2)

Before taking an in-depth look into what the dependencies of schema mapping $\mathcal{M}$ do, let us look at the source relation Q. It actually never occurs in the dependencies of $\mathcal{M}$.

We could *extract* a new source schema S' that does not include the information of Q at all and then use two new schema mappings: $\mathcal{M}_1$ from S to S' migrates from the old source schema to the new one. $\mathcal{M}_2$ from S' to T uses this new source schema to map to the target schema. In this example, we could take $\mathcal{M}_1$ to be just copy tgds from $S = \{P, Q, R\}$ to $S' = \{P, Q\}$ i.e. based on

- $P(x, y) \to P'(x, y)$
- $R(x, y) \to R'(x, y)$

and $\mathcal{M}_2$ to consist exactly of our two original dependencies σ_1 and σ_2 adapted to the new schema. Here, we would trivially have that the two mappings $\mathcal{M}_1$ and $\mathcal{M}_2$ composed do the same thing as the original $\mathcal{M}$, or more precisely $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$. ◀

In the preceding example, we have extracted a new source schema S' and $\mathcal{M}_1$ and $\mathcal{M}_2$ so that the composition $\mathcal{M}_1 \circ \mathcal{M}_2$ yields the original mapping $\mathcal{M}$. This feels like a natural condition for an extract operator, but as we have seen, this condition alone yields rather unimpressive results for S', $\mathcal{M}_1$ and $\mathcal{M}_2$. Let us continue our example.

▶ **Example 44 (ctd).** In our previous example, we saw that we may find unsatisfying $\mathcal{M}_1$ and $\mathcal{M}_2$ s.t. $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$, if we impose no further restrictions. Yet indeed, we want to extract *exactly* the information that participates in $\mathcal{M}$. Let us look at the source information that participates in $\mathcal{M}$, and should therefore transferred by $\mathcal{M}_1$.

In our first dependency σ_1, only the first variable x is actually used in the conclusion. We must not transfer y if we want to capture exactly the needed information. Similarly for σ_2, only the result after the join between P and R is relevant. In particular, we do not need the variable z in the conclusion. Let us express this as a schema mapping $\mathcal{M}_1$ based on

- $P(x, y) \to P_1(x)$
- $P(x, y) \wedge R(y, z) \to P_2(x, y)$

To make our argument about "transferring the needed amount of source information" precise, we have a tool at our hand: S-equivalence expresses that two schema mappings transfer the same amount of source information. That is, we have found an $\mathcal{M}_1$ with $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$.

Now having constructed $\mathcal{M}_1$, let us talk about $\mathcal{M}_2$, which should be able to do two things. First, it should be able to yield $\mathcal{M}$ in the sense that $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$. Secondly, we should require that $\mathcal{M}_2$ really covers exactly the amount of target information needed, or in other words $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$. Let us construct such an $\mathcal{M}_2$ based on

- $P_1(x) \to \exists u\, W(x, u) \wedge U(x, x)$
- $P_2(x, y) \to \exists v\, V(x, y, v)$

Altogether, we now have found schema mappings that capture exactly the information participating in $\mathcal{M}$, by requiring $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$ and $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$. ◀

Our characterization of how we expect the *extract* operator to behave is now reasonably complete. However while $\mathcal{M}_1$ transfers exactly the information needed, and $\mathcal{M}_2$ covers the information needed, the in-between schema S' has no condition imposed on it so far. In particular, we could still store every x from $P(x, y)$ as $P_1(x, x, x, x)$, which intuitively is not a good result of the extract operator. Let us continue our example.

▶ **Example 44 (ctd).** The problem we still have is possible redundancy in the new source schema S'. While using $P_1(x, x, x, x)$ as an intermediate atom seems quite drastic, actually we also have a somewhat surprising redundancy in $\mathcal{M}_1$, S' and $\mathcal{M}_2$ from the previous example, though it is a bit subtle:

Let us look at the source instance $I = \{P(\mathsf{a}, \mathsf{a}), P(\mathsf{b}, \mathsf{b}), R(\mathsf{b}, \mathsf{b})\}$. The canonical universal solution of I using our schema mapping $\mathcal{M}_1$ is $J = \{P_1(\mathsf{a}), P_1(\mathsf{b}), P_2(\mathsf{b}, \mathsf{b})\}$. But actually we do not need to store $P(\mathsf{b})$, since it occurs as $P_2(\mathsf{b},\mathsf{b})$ anyway. That is, any x from $P(x,y)$ that has a join-partner in R need not necessarily be stored in the intermediate instance. So the kind of redundancy we are talking about here is redundancy of possible *source instances or target instances.*

Of course, our current schema mapping $\mathcal{M}_2$ would not be sufficient to make use of this information, it would require a mapping with an additional dependency of the form $P_2(x,y) \rightarrow \exists u\, W(x,u) \wedge U(x,x)$. Yet this modified schema mapping is clearly T-equivalent, it covers the same target information. Let us avoid this redundancy in another way. Add to *both* $\mathcal{M}_1$ and $\mathcal{M}_2$ the following dependency:

- $P_2(x,y) \rightarrow P_1(x)$

yielding $\mathcal{M}_1'$ and $\mathcal{M}_2'$. This is a target tgd for $\mathcal{M}_1'$ and a source tgd for $\mathcal{M}_2'$.

Let us sum up and make precise our argument about redundancy. We are talking about possibly redundant instances here, and the optimality notions appropriate for this case have been already introduced in Section 4: source-redundancy and target-redundancy. We would like $\mathcal{M}_1$ to be target non-redundant among the S-equivalent mappings, and $\mathcal{M}_2$ to be source non-redundant among the T-equivalent mappings. ◀

Through a progression of examples, we have now identified a characterization for the *extract* operator that meets natural requirements. Let us make this definition explicit:

▶ **Definition 45.** [1] Let $\mathcal{M} = (S, T, \Sigma)$ be a mapping. $(\mathcal{M}_1, \mathcal{M}_2)$ is an extract of $\mathcal{M}$ if

- $\mathcal{M}_1 \circ \mathcal{M}_2 \equiv_{\mathsf{log}} \mathcal{M}$
- $\mathcal{M}_1 \equiv_{\mathsf{S}} \mathcal{M}$ and $\mathcal{M}_1$ is target non-redundant w.r.t. $\equiv_{\mathsf{S}}$
- $\mathcal{M}_2 \equiv_{\mathsf{T}} \mathcal{M}$ and $\mathcal{M}_2$ is source non-redundant w.r.t. $\equiv_{\mathsf{T}}$ ◀

We finish this subsection on the extract-operator by a few notes: Apart from the characterization, there also exists an algorithm for computing extracts for mappings based on s-t tgds with FO formulas in the antecedent. It is based on rewriting and composition [1]. The exact language needed to express these extracts is still open.

Apart from the extract operator, the merge operator was analyzed in [1] as well as the setting of schema evolution. Also, a characterization of the inverse operator [6] is given. The inverse operator is discussed in detail in Chapter 3 of this book.

7.2 Decidability and complexity

For the following results about the properties of our notions, we will be mainly talking about the ordering relations $\preceq_{\mathsf{S}}$ and $\preceq_{\mathsf{T}}$ instead of the equivalence relations $\equiv_{\mathsf{S}}$ and $\equiv_{\mathsf{T}}$. Our first goal will be to find algorithms for deciding these ordering relation.

An alternative characterization based on the following notions brings us one step closer to this goal: A query Q over source schema S is called *target rewritable* under $\mathcal{M}$, if there is a query Q' over T such that $Q(I) = \mathsf{certain}(Q, I, \mathcal{M})$ for all I. Then we have:

▶ **Theorem 46.** *[1] Let $\mathcal{M} = (S, T, \Sigma)$ and $\mathcal{M}' = (S, T', \Sigma')$ be a schema mapping based on s-t tgds. Then $\mathcal{M} \preceq_{\mathsf{S}} \mathcal{M}'$ iff for every query Q, if Q is target rewritable in $\mathcal{M}$ then Q is target rewritable in $\mathcal{M}'$. This result even holds if FO formulas are allowed as antecedents.* ◀

Backed up by this result we see that, equivalence and ordering w.r.t. source information transferred (which are based on transferring enough source information to be able to recover the original target information) are both intuitively and provably close to target rewritability.

For schema mappings based on s-t tgds, even with inequivalence in the antecedent, deciding $\mathcal{M} \preceq_S \mathcal{M}'$ is in coNEXPTIME. However, for schema mappings based on s-t tgds that allow FO formulas as antecedents, it is undecidable whether $\mathcal{M} \preceq_S \mathcal{M}'$ [1].

7.3 Summary

In this section, we looked at S-equivalence and T-equivalence. We applied them to characterize the operator *extract*, one of the central operators of schema mapping management. To characterize this operator, we saw that notions of equivalence (S- and T-equivalence) were needed, as well as notions of optimality (source- and target-redundancy).

We hinted at other operators *invert*, *merge*, and the setting of schema evolution that can be characterized using these concepts. For all of these, characterizations and algorithms can be found in [1]. We finished this section by a quick look at questions of decidability and complexity.

8 Reasoning in the broader sense

In the previous sections, we discussed reasoning about schema mappings in a strict sense. Our topics were primarily equivalence and optimality. But of course the term "reasoning" can be applied to a broad range of important tasks associated with schema mappings. While we cannot cover all of them, we want to finish this chapter by at least talking about one of them, in particular one that fits very well into what we discussed up to now.

Reasoning about schema mappings as a task humans have to do poses a number of challenges. Of course, using optimization techniques beforehand might help create schema mappings that are easier to handle as humans. Yet at some point, we have to deal with the actual schema mappings we have at that moment.

Given such a schema mapping, one of the first challenges is to find out what this schema mapping actually *does*, or rather what its intended meaning is. The other question that usually enters our reasoning process earlier than we might like is what a certain schema mapping *does wrong*. This topic of finding errors, that is debugging schema mappings, will be our topic in this section.

8.1 Analyzing and debugging with routes

Since we want to actually understand and debug some arbitrary schema mapping that we are given, let us start with such an example.

▶ **Example 47** (based on [5]). In this example, to ease debugging, we leave the names of the relation symbols intact. Also, universally quantified variables are denoted by words starting in lower case (sal), existentially quantified variables are given starting with upper case (M) and constants as usual in sans-serif font (Smith or $\mathsf{6689}$).

Let the mapping $\mathcal{M}$ over the source schema $\mathsf{ManhattenCredit} = \{\mathsf{Cards}, \mathsf{SuppCards}\}$ and the target schema $\mathsf{FargoFinance} = \{\mathsf{Accounts}, \mathsf{Clients}\}$ be given by the following dependencies:

- $\mathsf{Cards}(cn, l, s, n, m, sal, loc) \rightarrow \exists A\, (\mathsf{Accounts}(cn, l, s) \wedge \mathsf{Clients}(s, m, m, sal, A))$ (σ_1)
- $\mathsf{SuppCards}(an, s, n, a) \rightarrow \exists M, I\, \mathsf{Clients}(s, n, M, I, a)$ (σ_2)
- $\mathsf{Accounts}(a, l, s) \rightarrow \exists N, M, I, A\, \mathsf{Clients}(s, N, M, I, A)$ (σ_3)
- $\mathsf{Clients}(s, n, m, i, a) \rightarrow \exists N, L\, \mathsf{Accounts}(N, L, s)$ (σ_4)
- $\mathsf{Accounts}(a, I, s) \wedge \mathsf{Accounts}(a', I', s) \rightarrow I = I'$ (σ_5)

The schemas and the s-t tgds σ_1 and σ_2 are illustrated in Figure 6. Seemingly, it is a schema mapping describing how data about credit cards is transferred to some financial organization. Without worrying too much for now how this schema mapping actually works in detail, let us try debugging it with a test instance. Let I be given by the following ground atoms:

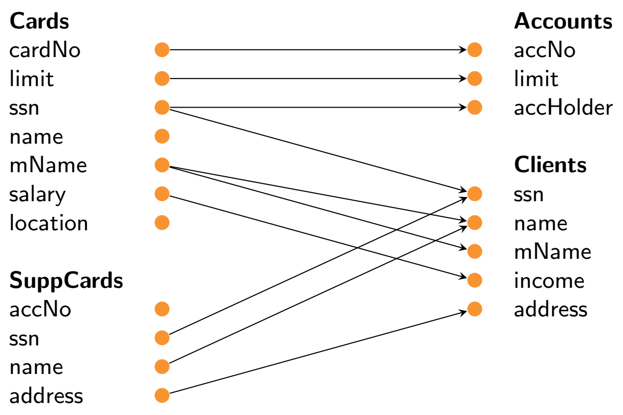


Figure 6 Schemas and s-t tgds of mapping $\mathcal{M}$.

- Cards(6689, 15K, 434, J.Long, Smith, 50K, Seattle) (s_1)
- SuppCards(6689, 234, A.Long, California) (s_2)

In the target database, we use the following instance J given by:

- Accounts(6689, 15K, 434) (t_1)
- Accounts(N_1, 50K, 234) (t_2)
- Clients(434, Smith, Smith, 50K, A_1) (t_3)
- Clients(234, A.Long, M_1, I_1, California) (t_4)

Let us take a closer look on t_3, which is slightly strange: In Clients(434, Smith, Smith, 50K, A_1), why is there a labeled null A_1 introduced, and why does the constant Smith occur twice?

Let us try to answer this question by tracing tuple t_3 back to the atoms that are directly responsible for creating it. We get the following atom s_1 being responsible for creating both t_1 and t_3 through dependency σ_1 using homomorphism h:

- Cards(6689, 15K, 434, J.Long, Smith, 50K, Seattle) (s_1)

  σ_1: Cards$(cn, l, s, n, m, sal, loc) \rightarrow \exists A\,($Accounts$(cn, l, s) \wedge$ Clients$(s, m, m, sal, A))$

  h: $\{cn \mapsto$ 6689$, l \mapsto$ 15K$, s \mapsto$ 434$, n \mapsto$ J.Long$, m \mapsto$ Smith$, sal \mapsto$ 50K$, loc \mapsto$ Seattle$, A \mapsto A_1\}$
- Accounts(6689, 15K, 434) (t_1) Clients(434, Smith, Smith, 50K, A_1) (t_3) ◀

Before using this chase-like step for debugging our schema mapping, let us formally define it:

▶ **Definition 48.** [5] A **satisfaction step** is given as $K_1 \xrightarrow{\sigma,h} K_2$ where

- K_1 is an instance such that $K_1 \subseteq K$ and K satisfies σ
- σ is a tgd $\varphi(\vec{x}) \rightarrow \exists \vec{y}\,\psi(\vec{x}, \vec{y})$
- h is a homomorphism from $\varphi(\vec{x}) \wedge \psi(\vec{x}, \vec{y})$ to K such that
  h is also a homomorphism from $\varphi(\vec{x})$ to K_1
- K_2 is the *result* of satisfying σ on K_1 with h, where $K_2 = K_1 \cup h(\psi(\vec{x}, \vec{y}))$ ◀

Note that this differs from the definition of a chase step, in particular because the applicability condition is far broader. This is not surprising since such a satisfaction step shall be able to help debug arbitrary solutions, whether they were created through the chase or not. We now continue our debugging:

▶ **Example 47 (ctd).** Let us look at what we can find out using this satisfaction step with the result including t_3. As we can see, the location Seattle is indeed contained in s_1, it just is not copied by σ_1, instead being replaced by a labeled null. This is most probably not the intended meaning, we will correct it. For the constant Smith occurring twice, we see that σ_1 uses m, m twice in the conclusion, instead of n, m from the antecedent. We can correct this typo as well, modifying σ_1 to

- $\mathsf{Cards}(cn, l, s, n, m, sal, loc) \to \mathsf{Accounts}(cn, l, s) \wedge \mathsf{Clients}(s, n, m, sal, loc)$ (σ_1')

So we corrected an error, but we might have spotted this error without any help of a debugging system.
Let us look at a more complex case, debugging t_2: $\mathsf{Accounts}(N_1, \mathsf{50K}, 234)$. This atom cannot be traced via a single satisfaction step to source atoms. But it can be traced back to two satisfaction steps (here, we omit the homomorphisms):

- $\mathsf{SuppCards}(6689, 234, \mathsf{A.Long}, \mathsf{California})$ (s_2)
  σ_2: $\mathsf{SuppCards}(an, s, n, a) \to \exists M, I\ \mathsf{Clients}(s, n, M, I, a)$
- $\mathsf{Clients}(234, \mathsf{A.Long}, M_1, I_1, \mathsf{California})$ (t_4)
  σ_4: $\mathsf{Clients}(s, n, m, i, a) \to \exists N, L\ \mathsf{Accounts}(N, L, s)$
- $\mathsf{Accounts}(N_1, \mathsf{50K}, 234)$ (t_2) ◀

Before using this sequence of satisfaction-steps for debugging, we again formally describe the notion first:

▶ **Definition 49.** [5] A **route** for J_s with $\mathcal{M}$, I and J is a sequence of satisfaction steps $(I, \emptyset) \xrightarrow{\sigma_1, h_1} (I, J_1) \ldots \xrightarrow{\sigma_n, h_n} (I, J_n)$ where

- J is a solution of I under $\mathcal{M}$
- $J_i \subseteq J$ and σ_i are from $\mathcal{M}$
- $J_s \subseteq J_n$ ◀

Having now defined what we mean by a route, let us use the one we have found in our continuing example for debugging our schema mapping:

▶ **Example 44 (ctd).** The route shows some strange things: The value 50K suddenly appears in t_2, without being required by the dependency. This also shows that J is actually not a *universal* solution, thus we witness the difference between satisfaction step and chase step.

Also, the account contains a labeled null N_1 as the account number, even though in the source tuple, we have the concrete value 6689. The reason is clear looking at this trace: The intermediate atom t_4 simply cannot store this account number. We can correct this by a more complex modification of σ_2 based on a join:

- $\mathsf{Cards}(cn, l, s_1, n_1, m, sal, loc) \wedge \mathsf{SuppCards}(cn, s_2, n_2, a) \to$
  $\exists M, I\ \mathsf{Clients}(s_2, n_2, M, I, a) \wedge \mathsf{Accounts}(cn, l, s_2)$ (σ_2')

In total, we might not have found all errors through this debugging, but a few obvious ones, and also some non-obvious errors have now been corrected. ◀

To conclude this section, we note a quite important fact for actual debugging with such routes: There is an algorithm for computing a minimal route, essentially in polynomial time w.r.t. the size of the atoms to-be-debugged. Also, there are situations where one route is not enough, but computing all routes is required. There is an algorithm for that as well in [5].

8.2 Summary

In this subsection, we scratched the surface of reasoning about schema mappings in the broader sense. We looked at a particular application of debugging schema mappings using the concept of routes. While we briefly noted algorithmic properties, we had no chance to explore further connections to e.g. the topic of provenance.

This was of course only an exemplified excursion into the broad topic of what "reasoning" about schema mappings may mean. Each of those meanings might fill a chapter of its own.

9 Conclusion

The topic of *reasoning about schema mappings* is a broad one. In this chapter, we focused on some of the central concepts of reasoning tasks: *equivalence* (Section 3) and *optimality* (Section 4). But we also talked about applications and reasoning in the broader sense.

So before summarizing what we discussed in this chapter, let us look at how those topics fit together in how we *actually* reason about schema mappings:

Given a **schema mapping**:
- What does it **do**, and
- Can we find **errors** in it? (*Debugging*, Section 8)
- Can we **optimize** it?
  - automatically using **logical** equivalence? (*Optimizing and Normalizing*, Section 5)
  - is there hope using **relaxed notions**? (*Boundaries of Decidability*, Section 6)
  - or at least preserving the **information** involved? (*Information Transfer*, Section 7)

This process could of course be augmented with any number of other reasoning tasks about schema mappings, both in the strict or in a broader sense. Let us now summarize what we discussed in this chapter:

9.1 Summary

In the first part of this chapter, focused on **concepts**, we introduced notions of *equivalence* and notions of *optimality*. We saw how they naturally arise when working with schema mappings and discussed their relationship to each other. As a quick reference of all involved notions, see Figure 2 and 3 at the ends of Section 3 and 4.

In the second part, we looked at **applications** and **computational properties** of these concepts. We first saw that under *logical equivalence*, one can optimize schema mappings based on s-t tgds under a broad range of optimality criteria, achieving a unique normal form.

We then explored the boundaries of decidability under *data-exchange equivalence* and *conjunctive-query equivalence*. While many of the general problems there are undecidable, we saw that there is both additional potential opened by these relaxations of logical equivalence, as well as some decidable cases that may exploit this additional potential.

We then continued to apply *equivalence in terms of information transfer* to characterizing important operators of schema mapping management. We saw that one can achieve quite concise characterizations in that way.

In the final part, we also gave a glimpse at the **broader sense** of reasoning about schema mappings. There, we briefly looked at analyzing and *debugging schema mappings* by introducing the concept of routes.

9.2 Outlook

While many of the general computational boundaries of reasoning about *equivalence* and *optimality* of schema mappings have been explored, there are a number of theoretical and practical problems open. For practical utilization, the search for useful decidable fragments is paramount. In particular, current algorithms like those illustrated in Section 5 might be extendable to cover an even broader range of schema mappings. Still, new approaches might be needed to cope with relaxed notions of equivalence.

In the broader sense of *reasoning about schema mappings*, we touched only the surface of available material. It is a topic that could fill many chapters of this size.

Acknowledgements. The author wants to thank the editors and reviewers for their exceptionally detailed and helpful comments.

References

1 Marcelo Arenas, Jorge Pérez, Juan L. Reutter, and Cristian Riveros. Foundations of schema mapping management. In *PODS*, pages 227–238, 2010.

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

3 Philip A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

4 Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

5 Laura Chiticariu and Wang Chiew Tan. Debugging schema mappings with routes. In *VLDB*, pages 79–90, 2006.

6 Ronald Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(4), 2007.

7 Ronald Fagin and Phokion G. Kolaitis. Local transformations and conjunctive-query equivalence. In *PODS*, pages 179–190, 2012.

8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

9 Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. Towards a theory of schema-mapping optimization. In *PODS*, pages 33–42, 2008.

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

11 Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.

12 Ingo Feinerer, Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. On the undecidability of the equivalence of second-order tuple generating dependencies. In *AMW*, 2011.

13 Georg Gottlob and Alan Nash. Data exchange: computing cores in polynomial time. In *PODS*, pages 40–49, 2006.

14 Georg Gottlob, Reinhard Pichler, and Vadim Savenkov. Normalization and optimization of schema mappings. *VLDB J.*, 20(2):277–302, 2011.

15 Alon Y. Halevy, Anand Rajaraman, and Joann J. Ordille. Data integration: The teenage years. In *VLDB*, pages 9–16, 2006.

16 Phokion G. Kolaitis. Schema mappings, data exchange, and metadata management. In *PODS*, pages 61–75, 2005.

17 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

18 Sergey Melnik. *Generic Model Management: Concepts and Algorithms*, volume 2967 of *Lecture Notes in Computer Science*. Springer, 2004.

19 Reinhard Pichler, Emanuel Sallinger, and Vadim Savenkov. Relaxed notions of schema mapping equivalence revisited. In *ICDT*, pages 90–101, 2011.

20 Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.

21 Balder ten Cate and Phokion G. Kolaitis. Structural characterizations of schema-mapping languages. In *ICDT*, pages 63–72, 2009.

Query Processing in Data Integration

Paolo Guagliardo[1] and Piotr Wieczorek[2]

1 **KRDB Research Centre, Free University of Bozen-Bolzano, Italy**
guagliardo@inf.unibz.it
2 **University of Wrocław, Poland**
piotr.wieczorek@cs.uni.wroc.pl

Abstract

In this chapter we illustrate the main techniques for processing queries in data integration. The first part of the chapter focuses on the problem of query answering in the relational setting, and describes approaches based on variants of the *chase*, along with how to deal with integrity constraints and access patterns. The second part of the chapter investigates query processing in the context of semistructured data, which is best described by graph-based data models, where the expressiveness of query languages not common in traditional database systems allows to point out the subtle differences between query answering and query rewriting. The chapter is closed by a very brief discussion of query processing in data integration with XML and ontologies.

1998 ACM Subject Classification H.2.4 [Systems]: Query processing, A.1 Introduction and survey

Keywords and phrases Query processing, data integration, relational data, semistructured data

Digital Object Identifier 10.4230/DFU.Vol5.10452.129

1 Introduction

The present chapter deals with the broad area of query processing in data integration, by illustrating the existing query answering techniques both for relational data and for semi-structured data.

The focus of the first part is on query answering in the relational case. We start with a brief description of possible results for various combinations of parameters of the problem. Then we describe the approaches that insist on the reconstruction of a representation of global database(s). The main algorithmic techniques are variants of the *chase* of the sources. We explain the concept of universal solution and show that it can be useful even if we would like to compute a query rewriting only (*inverse-rules method*). Next idea discussed is to rewrite the query such that it could be evaluated directly on the sources without materialisation (even if temporary) of the sources. We illustrate approaches based on the analysis of the relationship between the atoms of the user query and in the views, like in the MiniCon algorithm. We illustrate also the ways of dealing with integrity constraints and access patterns. We also discuss a technique of chasing the queries that leads to complete but unsound rewritings. The first part ends with a study of the information-theoretic notion of determinacy and its relation to rewriting.

Then, we investigate query processing in the context of semistructured data, capturing data that does not fit into the predefined and strict schemas of the relational setting, but is best described by graph-based data models. The main mechanism used for querying such kind of data consists in *regular path queries*, which are binary queries specifying the pairs of objects connected in a semistructured database by a path that conforms to a regular expression. Regular path queries can traverse the edges of a semistructured database in

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 129–160
Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

the forward direction only, but they can be extended with the ability of navigating edges also backward by means of an inverse operator, in which case we speak of *two-way regular path queries*. These query languages, not common in traditional database systems, possess a peculiar expressive richness that allows to uncover the subtle differences existing between query answering and query rewriting and which are completely blurred when focusing on conjunctive queries.

In the second part of the chapter, we first present a technique for rewriting regular expressions that can be readily applied for regular path queries, and we then discuss the relationship between query answering and query rewriting, also in relation to the relevant notion of *losslessness*. Indeed, we examine the relationship between various notions that assess different aspects of losslessness, w.r.t. answering and w.r.t. rewriting, in order to understand whether there is loss of information when processing a query based on a set of views and, in such a case, what is the cause.

We conclude the chapter by briefly discussing query processing in other data integration scenarios, namely XML data integration and the newly emerging area of ontology-based data integration. Due to space limitations, this part has no pretence of exhaustiveness, but it merely mentions current trends in data integration research.

2 Preliminaries

In this section, we introduce the necessary notation and give some basic definitions that will be used throughout the chapter.

An n-ary *relation* on a set A, where $n \in \mathbb{N}$ is called the *arity* of the relation, is a subset of the Cartesian product A^n, that is, a set of n-tuples of elements of A. A *signature* (or *alphabet*) is a finite set of relation symbols, each of which has an associated arity. A *relational structure* (or *instance*) over a signature σ is a pair $\mathbf{I} = \langle \Delta^{\mathbf{I}}, \cdot^{\mathbf{I}} \rangle$, where $\Delta^{\mathbf{I}}$ is a *domain* of objects and $\cdot^{\mathbf{I}}$ is a function associating each relation symbol $r \in \sigma$ with a relation $r^{\mathbf{I}}$ of appropriate arity (i.e., if r is an n-placed relation symbol, then $r^{\mathbf{I}}$ is an n-ary relation). A relational structure over a signature σ is also called a *σ-extension*. Sometimes we treat relational structures as sets of facts, that is, an instance $\mathbf{I}$ is a set containing exactly one fact $r(t)$ for each relation symbol r and each tuple $t \in r^{\mathbf{I}}$.

We call *database signature* a signature $\mathcal{R} = \{R_1, \ldots, R_n\}$ of *database symbols* and *view signature* a signature $\mathcal{V} = \{V_1, \ldots, V_k\}$ of *view symbols* not occurring in $\mathcal{R}$. Each view symbol $V \in \mathcal{V}$ has an associated *view definition* $V^{\mathcal{R}}$, which is a formula in some language $\mathcal{L}$ over $\mathcal{R}$ expressing V in terms of the database symbols. A $\mathcal{V}$-extension (i.e., a view instance) is denoted by $\mathbf{E}$; an $\mathcal{R}$-extension (i.e., a database instance) is denoted by $\mathbf{D}$ and simply called a *database*.

A *query* Q is a function from relational structures over a given signature $\mathcal{S}$ to relations, associating each relational structure $\mathbf{I}$ over $\mathcal{S}$ with a relation $Q(\mathbf{I})$ of a certain arity, called the *answer* to Q over $\mathbf{I}$. We refer to queries over the database signature $\mathcal{R}$ as *database queries* and to queries over the view signature $\mathcal{V}$ as *view queries*.

We consider two different assumptions on $\mathcal{V}$-extensions. Under Closed World Assumption (CWA), a $\mathcal{V}$-extension stores all the tuples that satisfy the view definitions. In this case we call the views *exact*. Alternatively, under Open World Assumption (OWA), a $\mathcal{V}$-extension may store only some of the tuples that satisfy the view definitions. In this case, we call the views be *sound*. We formally define sound and exact views as follows.

▶ **Definition 1.** Let $\mathbf{D}$ be a database and $\mathcal{V}^{\mathcal{R}}(\mathbf{D})$ be the $\mathcal{V}$-extension $\mathbf{E}$ such that $V(\mathbf{E}) = V^{\mathcal{R}}(\mathbf{D})$ for each $V \in \mathcal{V}$. A $\mathcal{V}$-extension $\mathbf{E}$ is said to be *sound* w.r.t. $\mathbf{D}$ iff $\mathbf{E} \subseteq \mathcal{V}^{\mathcal{R}}(\mathbf{D})$, and it is said to be *exact* w.r.t. $\mathbf{D}$ iff $\mathbf{E} = \mathcal{V}^{\mathcal{R}}(\mathbf{D})$.

In a traditional database setting, we answer queries by evaluating them on the database. In the context of view-based query processing, we have view extensions, and we aim at processing queries based only on the information about the views. There are two forms of view-based query processing, namely: view-based query answering (i.e., computing certain answers), and view-based query rewriting (i.e., computing query rewritings). The basic notions for the two tasks are defined as follows.

▶ **Definition 2** (Certain answers). The *certain answers* to a query Q *under sound views* $\mathcal{V}$ w.r.t. a $\mathcal{V}$-extension $\mathbf{E}$ is the set of all tuples t such that $t \in Q(\mathbf{D})$ for every database $\mathbf{D}$ w.r.t. which $\mathbf{E}$ is sound, that is:

$$\operatorname{cert}^{\text{sound}}_{Q,\mathcal{V}}(\mathbf{E}) = \bigcap \bigl\{ Q(\mathbf{D}) \mid \mathbf{D} \text{ is such that } \mathbf{E} \subseteq \mathcal{V}^{\mathcal{R}}(\mathbf{D}) \bigr\} \ . \tag{1}$$

The *certain answers* to a query Q *under exact views* $\mathcal{V}$ w.r.t. a $\mathcal{V}$-extension $\mathbf{E}$ is the set of all tuples t such that $t \in Q(\mathbf{D})$ for every database $\mathbf{D}$ w.r.t. which E is exact, that is:

$$\operatorname{cert}^{\text{exact}}_{Q,\mathcal{V}}(\mathbf{E}) = \bigcap \bigl\{ Q(\mathbf{D}) \mid \mathbf{D} \text{ is such that } \mathbf{E} = \mathcal{V}^{\mathcal{R}}(\mathbf{D}) \bigr\} \ . \tag{2}$$

▶ **Definition 3** (Rewriting). Let Q be a query over a database signature $\mathcal{R}$ and Q_{r} be a query over a view signature $\mathcal{V}$. Q_{r} is a *rewriting* of Q *under sound views* $\mathcal{V}$ iff for every database $\mathbf{D}$ and every $\mathcal{V}$-extension $\mathbf{E}$ which is sound w.r.t. $\mathbf{D}$ it holds that $Q_{\mathrm{r}}(\mathbf{E}) \subseteq Q(\mathbf{D})$. Q_{r} is a *rewriting* of Q *under exact views* $\mathcal{V}$ iff for every database $\mathbf{D}$ and every $\mathcal{V}$-extension $\mathbf{E}$ which is exact w.r.t. $\mathbf{D}$ it holds that $Q_{\mathrm{r}}\bigl(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\bigr) \subseteq Q(\mathbf{D})$. A rewriting is *exact* if the subset inclusion in the above conditions is in fact an equality.

Rewritings are view queries that, in general, are formulated in a different language than the one used for database queries. We consider languages $\mathcal{L}_{\mathrm{q}}$ and $\mathcal{L}_{\mathrm{r}}$ in which database queries and rewritings are respectively expressed, along with a language $\mathcal{L}_{\mathrm{v}}$ for expressing view definitions. When the rewriting language $\mathcal{L}_{\mathrm{r}}$ is monotonic, the definition of rewriting under sound views coincides with the one of rewriting under exact views.

▶ **Proposition 1.** *Let $Q_r \in \mathcal{L}_r$ and $Q \in \mathcal{L}_q$, and let $\mathcal{L}_r$ be monotonic. Then, Q_r is a rewriting of Q under sound views $\mathcal{V}$ iff Q_r is a rewriting of Q under exact views $\mathcal{V}$.*

Proof. A rewriting under sound views is always also a rewriting under exact views, thus we only need to show the opposite direction (under the assumption that the rewriting language $\mathcal{L}_{\mathrm{r}}$ is monotonic). Assume that Q_{r} is a rewriting of Q under exact views $\mathcal{V}$, hence for every database $\mathbf{D}$ we have that $Q_{\mathrm{r}}\bigl(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\bigr) \subseteq Q(\mathbf{D})$. By the monotonicity of $\mathcal{L}_{\mathrm{r}}$, we then get that for every $\mathcal{V}$-extension $\mathbf{E}$ such that $\mathbf{E} \subseteq \mathcal{V}^{\mathcal{R}}(\mathbf{D})$ it holds that $Q_{\mathrm{r}}(\mathbf{E}) \subseteq Q_{\mathrm{r}}\bigl(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\bigr)$ and, in turn, $Q_{\mathrm{r}}(\mathbf{E}) \subseteq Q(\mathbf{D})$. ◀

Rewritings as defined above in Definition 3 are "contained" rewritings, that is, they provide an underestimation of the original query. Often we are interested in rewritings that are in a sense the best underestimations of the query, hence it is natural to consider rewritings that are maximal, also called *maximally-contained* rewritings, which we define below.

▶ **Definition 4** (Maximal rewriting). A rewriting $Q_{\mathrm{r}} \in \mathcal{L}_{\mathrm{r}}$ of a query $Q \in \mathcal{L}_{\mathrm{q}}$ under sound views $\mathcal{V}$ is *$\mathcal{L}_r$-maximal* iff, there is no rewriting $Q'_{\mathrm{r}} \in \mathcal{L}_{\mathrm{r}}$ of Q under $\mathcal{V}$ such that, for some $\mathcal{V}$-extension $\mathbf{E}$ which is sound w.r.t. some database $\mathbf{D}$, it is the case that $Q'_{\mathrm{r}}(\mathbf{E}) \supset Q_{\mathrm{r}}(\mathbf{E})$. A rewriting $Q_{\mathrm{r}} \in \mathcal{L}_{\mathrm{r}}$ of a query $Q \in \mathcal{L}_{\mathrm{q}}$ under exact views $\mathcal{V}$ is *$\mathcal{L}_r$-maximal* iff, there exists no rewriting $Q'_{\mathrm{r}} \in \mathcal{L}_{\mathrm{r}}$ of Q under $\mathcal{V}$ such that $Q'_{\mathrm{r}}\bigl(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\bigr) \supset Q_{\mathrm{r}}\bigl(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\bigr)$ for some database $\mathbf{D}$.

Dually, we can consider also *minimally-containing* rewritings that are the least overestimations of the query, but which are not even rewritings according to Definition 3. We study them in Section 3.

In some applications, it is of interest to consider rewritings that always provide exactly the same information provided by the original query. Such rewritings are called *exact* and are formally defined as follows:

▶ **Definition 5** (Exact rewriting). A rewriting $Q_{\mathrm{r}} \in \mathcal{L}_{\mathrm{r}}$ of a query $Q \in \mathcal{L}_{\mathrm{q}}$ under views $\mathcal{V}$ is *equivalent* to Q iff $Q_{\mathrm{r}}\big(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\big) = Q(\mathbf{D})$ for every database $\mathbf{D}$.

That is, for every database $\mathbf{D}$, the evaluation of the rewriting on the view extensions w.r.t. $\mathbf{D}$ returns exactly the same answers given by evaluating the original query on $\mathbf{D}$. Observe that an exact rewriting may not always exist, possibly because it is not expressible in $\mathcal{L}_{\mathrm{r}}$.

We are now ready to provide the formal definition of data integration system, following the approach in [41]: A *data integration system* $\mathcal{I}$ is a triple $(\mathcal{G}, \mathcal{S}, \mathcal{M})$, where $\mathcal{G}$ is the *global schema*, $\mathcal{S}$ is the *source schema*, and $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$. The source schema describes the structure of the sources, where the real data is stored, while the global schema provides a reconciled, integrated and virtual view of the underlying sources. The source and global schema may be simple definitions of a set of relations or may allow for various forms of integrity constraints. The mapping connects the elements of the global schema with those of the source schema.

Consider a *source database* $\mathbf{D}$, that is, a relational structure over the source schema $\mathcal{S}$. We say that a *global database* $\mathbf{B}$ (any relational structure over $\mathcal{G}$) is legal for $\mathcal{I}$ and $\mathbf{D}$ if $\mathbf{B}$ satisfies all constraints of $\mathcal{G}$ and $\mathbf{B}$ satisfies the mapping $\mathcal{M}$ with respect to $\mathbf{D}$.

A mapping is expressed as a set of dependencies between $\mathcal{G}$ and $\mathcal{S}$. We consider dependencies in one of the following forms: *local-as-view* (LAV), *global-as-view* (GAV) and their combination *global-and-local-as-view* (GLAV).

In the GAV setting, the mapping characterise the content of each element of the global database as a view over the sources. Typically, GAV views are assumed to be exact, in which case the mapping (in the relational setting) can be specified as a set of dependencies of the form

$$\forall \vec{x}\ \varphi_{\mathcal{S}}(\vec{x}) \leftrightarrow R(\vec{x}) \ , \tag{3}$$

one for each relation symbol R in the global schema $\mathcal{G}$, where $\varphi_{\mathcal{S}}(\vec{x})$ is a query over $\mathcal{S}$. On the other hand, when GAV views are assumed to be sound, the equivalence in (3) must be replaced by an implication as follows:

$$\forall \vec{x}\ \varphi_{\mathcal{S}}(\vec{x}) \rightarrow R(\vec{x}) \ . \tag{4}$$

In the LAV setting, the sources are characterised in terms of a view over the global schema. Conversely, LAV views are commonly assumed to be sound, in which case the mapping can be specified as a set of dependencies of the form

$$\forall \vec{x}\ P(\vec{x}) \rightarrow \exists \vec{y}\ \psi_{\mathcal{G}}(\vec{x}, \vec{y}) \ , \tag{5}$$

one for each relation P in the source schema $\mathcal{S}$, where $\psi_{\mathcal{G}}(\vec{x}, \vec{y})$ is a query over $\mathcal{G}$. On the other hand, when LAV views are assumed to be exact, the implication in (5) must be replaced by an equivalence as follows:

$$\forall \vec{x}\ P(\vec{x}) \leftrightarrow \exists \vec{y}\ \psi_{\mathcal{G}}(\vec{x}, \vec{y}) \ . \tag{6}$$

Note that the LAV setting is exactly the scenario of view-based query processing. Indeed, in LAV we have one view for each source and the source content corresponds to the view extensions. A query posed against the global schema is answered by using both the view definitions (i.e., the mapping) and the view extension (i.e., the data at the sources). In some occasions, we refer to "view-based query processing" (i.e., query answering/rewriting using views), rather than to "query processing in LAV", so as to reflect the terminology used in the relevant literature.

Finally, we can consider the most general approach, called GLAV setting, which allows to specify mappings as any source-to-target tuple generating dependencies (s-t tgds) [34]:

$$\forall \vec{x}\; \varphi_{\mathcal{S}}(\vec{x}) \to \exists \vec{y}\; \psi_{\mathcal{G}}(\vec{x}, \vec{y}) \ . \tag{7}$$

In general, given a data integration system and a source database, there could more than one legal global databases. In order to define what it means to answer a query in this case, we resort to the notion of certain answers. Similarly to what we did in the context of view-based query processing, we define the certain answers $\mathrm{cert}_{Q,\mathcal{I}}(\mathbf{D})$ to a query Q in the system $\mathcal{I}$ with respect to $\mathbf{D}$ as

$$\mathrm{cert}_{Q,\mathcal{I}}(\mathbf{D}) = \bigcap \big\{ Q(\mathbf{B}) \mid \mathbf{B} \text{ is a legal database for } \mathcal{I} \text{ and } \mathbf{D} \big\} \ . \tag{8}$$

In what follows, we will refer to the most well known query languages such as datalog, conjunctive queries (CQ) and their unions.

▶ **Definition 6.** *A datalog*$^\neg$ *rule* is an expression of the following form $P(\vec{x}) \leftarrow l_1(\vec{x}_1), \ldots, l_n(\vec{x}_n)$, where each of $l_i(\vec{x}_i)$ is a *literal* i.e. positive or negated atom $R(\vec{x})$. We mention it explicitly if we allow negated atoms in queries by placing the superscript like in datalog$^\neg$. $P(\vec{x})$ is the head of the rule and $l_1(\vec{x}_1), \ldots, l_n(\vec{x}_n)$ is the body. Each variable in the head of a rule must occur in the body of the rule. *A datalog*$^\neg$ *program* is a finite set of datalog$^\neg$ rules. If a query is given by one or more non-recursive rules and all of the rules have the same head then it is *a union of conjunctive queries (UCQ)*. A single, non-recursive, datalog rule is called *a conjunctive query (CQ)*. We say that a query is *safe* if, for each of its rules, every variable appearing in such rule (whether in the head or in the body) appears in a positive literal of the same rule.

When discussing conjunctive queries we will often use the notion of a canonical structure.

▶ **Definition 7** (Canonical structure). *A canonical structure* $\mathbf{C_Q}$ for a conjunctive query Q is the structure over the signature consisting of the relation symbols mentioned by Q such that each relation $R^{\mathbf{C_Q}}$ is the set of all tuples $\vec{a}$ of variables and constants in (positive) atoms $R(\vec{a})$ of Q.

▶ **Definition 8** (Expansion of a query). Let $\mathcal{R}$ be a database signature, let $\mathcal{V}$ be a view signature and assume that for each $V \in \mathcal{V}$ the view definition $V^{\mathcal{R}}$ is expressed as a conjunctive query. *An expansion* $\exp(Q_{\mathrm{r}})$ of a query Q_{r} over $\mathcal{V}$ is the query over $\mathcal{R}$ that is obtained by substituting every atom $V(\vec{a})$ in Q_{r} with the corresponding view definition $V^{\mathcal{R}}(\vec{a})$.

An expansion in a LAV data integration setting, where the mapping is given as a set of view definitions expressed as conjunctive queries, is defined analogously.

3 Query processing in relational data integration

In the first part of this section we will describe the main approaches that have been proposed for query answering in data integration for the case of relational data. Successively, we study the information-theoretic notion of determinacy and its relation to rewriting.

3.1 Approaches to query answering

There is a number of parameters that heavily influence query answering in data integration [41], namely: the presence of integrity constraints in the global schema, the class of allowed mappings, the classes of user queries, and the class of queries in the mappings.

Before we discuss the fundamental techniques in query answering, we review shortly the basic results for various combinations of the parameters.

GAV without constraints. This is the simplest case for query answering. We assume first-order queries in the mapping. If additionally, the views are exact then it can be proved that there is a single global database that is legal w.r.t the sources. This is the *retrieved global database* **B** over the source schema where the view extensions are computed using the view definitions. Note that since there are no existential variables on the right-hand side of view definitions, the tuples in **B** have the elements from the source database only.

Clearly, the answer to user query Q is computed by evaluating Q over the database **B**.

Similarly, it is easy to modify the user query in order to obtain an equivalent query that can be executed over the sources. This can be done following the unfolding strategy where each of the atoms over V where V is a relation symbol in the global schema is substituted with the corresponding query in the GAV mapping (i.e. the view definition).

It turns out that if we assume that views are sound then only a very limited form of incompleteness appears. Namely, each view extension can be any superset of what is computed using the view definitions (for monotone queries). Thus, given a source database **D** there exist a number of global databases that are legal w.r.t. **D**. However, it is easy to see that there exists the single minimal global database, which is the intersection of all such databases.

GAV with constraints. The presence of integrity constraints changes the situation radically. It turns out that in addition to incompleteness (i.e. there may be several global databases that are legal w.r.t. the sources) also the inconsistency can show up (i.e. there may be no global database that is legal w.r.t. sources).

Query answering in a GAV system with key and foreign key constraints has been studied in [11]. It is proved that computing certain answers corresponds to evaluating the query over a special *canonical database* that may be infinite in general. However, instead of direct evaluation of the query on the canonical database, the algorithm constructing a query rewriting in terms of a logic program is proposed. The approach results in polynomial data complexity for query answering in the case of conjunctive queries. The results were further extended in [13, 12] to deal with unions of conjunctive queries and inclusion dependencies.

(G)LAV. This is the setting that was the most intensively studied in literature, see [43, 1, 38, 33, 51, 48, 23]. Before we discuss the main approaches to (G)LAV query answering in detail we present a set of (data) complexity results in Tables 1 and 2 [1].

In the rest of the chapter we describe the main groups of query processing techniques for the case of LAV mappings. The approaches in the first group insist on the reconstruction of a representation of global database(s). The main algorithmic techniques are various variants of the *chase* of the sources. We explain the concept of universal solution and show that it can be useful even if we would like to compute a query rewriting only (*inverse-rules method*). Then, we describe approaches based on the analysis of the relationship between the atoms of the user query and in the views (e.g. the MiniCon algorithm). The main focus we put on the maximally-contained rewritings, since as proved in [1] for $\mathcal{L} \in \{\text{datalog}, \text{UCQ}\}$, $\mathcal{L}$-maximal

Table 1 Data complexity of computing certain answers assuming sound views [1].

views \ query	CQ	$CQ^{\neq}$	PQ	datalog	FO
CQ	PTIME	coNP	PTIME	PTIME	undec.
$CQ^{\neq}$	PTIME	coNP	PTIME	PTIME	undec.
PQ	coNP	coNP	coNP	coNP	undec.
datalog	coNP	undec.	coNP	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Table 2 Data complexity of computing certain answers assuming exact views [1].

views \ query	CQ	$CQ^{\neq}$	PQ	datalog	FO
CQ	coNP	coNP	coNP	coNP	undec.
$CQ^{\neq}$	coNP	coNP	coNP	coNP	undec.
PQ	coNP	coNP	coNP	coNP	undec.
datalog	undec.	undec.	undec.	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

rewriting of a query in $\mathcal{L}$ under OWA computes exactly the set of certain answers. We illustrate also the ways of dealing with integrity constraints and access patterns. Finally, we discuss a technique of chasing the queries that leads to complete but unsound rewritings.

When describing complexity bounds we always mean *data complexity*, that is the complexity w.r.t. the size of data at the sources.

3.2 Reconstruction of global data

The first approach we are going to describe reduces query answering to the two steps:

1. Materialise some representation of global data.
2. Evaluate the query on the materialisation.

This approach is closely connected to the of *data exchange* [34] where we have a source schema, a target schema (that corresponds to the global schema in data integration) and a source-to-target mapping. The main difference between data integration and data exchange is in their goals. The primary goal of data exchange is to materialise the global database (*target instance*), while such materialisation is not required in data integration. Then query answering is performed using the materialised target instance without accessing the sources.

In this context we consider source-to-target dependencies of the form $\forall\vec{x}\forall\vec{y}\,\varphi(\vec{x},\vec{y}) \rightarrow \exists\vec{z}\,\psi(\vec{x},\vec{z})$, where φ is a conjunction of atoms over $\mathcal{S}$ and ψ is a conjunction of atoms over $\mathcal{G}$. Such dependencies correspond to the mappings in the GLAV data integration system.

The first step in order to implement this strategy is to find a good representation of possible global databases. Natural candidates has been studied in the field of *incomplete databases* such as Codd tables, naive tables or conditional tables [39, 37]. Generally, a table is a database such that in the domain, in addition to constants, variables (*nulls*) are allowed. A table represents a set of complete databases, each obtained by substituting all variables with constants.

Formally, let Δ be the set of all values, called *constants*, that may occur in domains of relational structures. In addition, the domain of a table can contain values from an infinite set $\underline{\mathrm{Var}}$ of labelled nulls. We have $\Delta \cap \underline{\mathrm{Var}} = \varnothing$. In naive tables there may be different occurrences of the same variable in contrast to a Codd tables where each variable can appear only once. While querying naive tables labelled nulls are treated like constants, in particular, each of them is equal to itself only. The distinction between variables and constants needs to be kept because of the function variables play when working with homomorphisms and substitutions.

A table $\mathbf{B}$ represents (under Open World Assumption) the following set of databases:

$$\text{Rep}(\mathbf{B}) = \{v(\mathbf{B}) \quad | \quad v \text{ maps all nulls in } \mathbf{B} \text{ to elements of } \Delta\},$$

where $v(\mathbf{B})$ is the database that results from $\mathbf{B}$ after replacing each null x with $v(x)$.

Given tables $\mathbf{B}$ and $\mathbf{B}'$, a homomorphism $h : \mathbf{B} \to \mathbf{B}'$ is a mapping from $\Delta^{\mathbf{B}}$ (i.e. the domain of $\mathbf{B}$), to $\Delta^{\mathbf{B}'}$ (i.e. the domain of $\mathbf{B}'$), such that (1) $h(c) = c$, for every $c \in \Delta$ and (2) for every fact $R(\vec{a})$ in $\mathbf{B}$, we have that $R(h(\vec{a}))$ is a fact of $\mathbf{B}'$.

The answer $Q(\mathbf{B})_{\downarrow}$ to a query Q over a table $\mathbf{B}$ is computed in the following way: first Q is evaluated on $\mathbf{B}$ (recall that in the process the nulls are treated as they were constants i.e. two nulls are equal if and only if they are syntactically equal) and then all tuples containing nulls are discarded.

Now we introduce the notion of *universal solution* [34]. Assume we are given a data integration system $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ and a source database $\mathbf{D}$. We say that a naive table $\mathbf{B}$ over the global schema $\mathcal{G}$ such that $\Delta^{\mathbf{B}} \subseteq \Delta \cup \underline{\text{Var}}$ is a universal solution for $\mathcal{I}$ and $\mathbf{D}$ if and only if (1) $\mathbf{B}$ is legal with respect to $\mathbf{D}$ and $\mathcal{I}$ and (2) for each global database $\mathbf{B}'$ that is legal w.r.t. $\mathbf{D}$ and $\mathcal{I}$ there exists a homomorphism $h : \mathbf{B} \to \mathbf{B}'$.

Why universal solutions are important? Intuitively, a universal solution is a solution that contains all the essential information as required by the mappings. Assume that the user queries are preserved under homomorphisms. That is, for every query Q if $h : \mathbf{B} \to \mathbf{B}'$ and $\vec{a} \in Q(\mathbf{B})$ then $h(\vec{a}) \in Q(\mathbf{B}')$. This condition is satisfied by positive queries such as conjunctive queries, unions of conjunctive queries and datalog.

We have the following theorem.

▶ **Theorem 9** ([34]). *Let $\mathcal{I}$ be a (GLAV) data integration system, let $\mathbf{D}$ be a source database, and let $\mathbf{B}$ be a universal solution for $\mathcal{I}$ and $\mathbf{D}$. For a query Q that is preserved under homomorphisms we have* $\text{cert}_{Q,\mathcal{I}}(\mathbf{D}) = Q(\mathbf{B})_{\downarrow}$.

Indeed, $\text{cert}_{Q,\mathcal{I}}(\mathbf{D}) \subseteq Q(\mathbf{B})_{\downarrow}$ because $\mathbf{B}$ is legal w.r.t. $\mathbf{D}$ and $\mathcal{I}$. Moreover $Q(\mathbf{B})_{\downarrow} \subseteq \text{cert}_{Q,\mathcal{I}}(\mathbf{D})$ because from the definition for each global database $\mathbf{B}'$ that is legal w.r.t. $\mathbf{D}$ and $\mathcal{I}$ there exists a homomorphism $h : \mathbf{B} \to \mathbf{B}'$ and h preserves Q.

In order to compute a universal solution we can use the classical chase [44, 4]:

▶ **Algorithm 1** (Chase the sources).

Input: *a data integration system $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ and a source database $\mathbf{D}$.*

Output: *The canonical universal solution Sol($\mathbf{D}$).*

```
For every element of $\mathcal{M}$ of the form $\forall\vec{x}\forall\vec{y}\,\varphi(\vec{x},\vec{y}) \to \exists\vec{z}\,\psi(\vec{x},\vec{z})$
    for every tuple $(\vec{a},\vec{b})$ such that $\mathbf{D}$ satisfies $\varphi(\vec{a},\vec{b})$;
        insert the tuples $\psi(\vec{a},\vec{Z})$ into Sol($\mathbf{D}$), where $\vec{Z}$ is a fresh tuple of nulls
```

In the simple case where all dependencies are source-to-target and there are no constraints in $\mathcal{G}$ (i.e. no target dependencies) the chase always stops and constructs the solution in polynomial time.

The construction can be extended to the case where there are global constraints in $\mathcal{G}$, such as

- *equality-generating target dependencies (egds)* of the form

  $$\forall \vec{x}\, \varphi(\vec{x}) \rightarrow x_i = x_j,$$

  where φ is a conjunction of atoms over $\mathcal{G}$, x_i and x_j are among the variables in $\vec{x}$; and
- *tuple-generating target dependencies (tgds)* of the form

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

  where both φ and ψ are conjunctions of atoms over $\mathcal{G}$. If there are no existential variables in the right hand side then such tgds is called to be *full*.

During the chase, target egds and tgds are applied as long as $\mathrm{Sol}(\mathbf{D})$ does not satisfy all of them. However, in the presence of egds the chase may fail. This is because two distinct constants may be required to be equal. Moreover, in presence of tgds that are not full the chase does not always terminate and to make things worse checking termination of the chase is undecidable. [34] and [28] introduce a sufficient and verifiable in polynomial time condition for the termination of chase: the dependencies have to be *weakly-acyclic* (or have *stratified witnesses*). The idea is to keep track of how the values propagate during the chase, and, in particular, whether a new labelled null can determine the creation of another new labelled null, at a later chase step. For full details and proofs we refer to [34]. The research on the properties of the chase continues and better termination conditions are discovered (see e.g. [24].

Quite surprisingly, the concept of chase is useful even if we would like to compute a query rewriting only, without considering the view extensions in the first place. Consider the following strategy:

1. modify the query such that it is possible to evaluate it on the sources (i.e. compute the query rewriting),
2. evaluate the query rewriting on the sources.

Now we present *inverse-rules method* [33]. We assume the LAV setting $(\mathcal{G}, \mathcal{S}, \mathcal{M})$, in particular the mapping $\mathcal{M}$ is specified by the queries of the form $V(\vec{x}) \rightarrow \psi(\vec{x}, \vec{z})$, where V is a symbol in $\mathcal{S}$ and $\psi(\vec{x}, \vec{z})$ is a conjunction of atoms over $\mathcal{G}$. Furthermore, the language for user queries and query rewritings is datalog.

The *inverse rules* are defined as follows. For each rule r in $\mathcal{M}$ and for each of the free variables $y_1, \ldots, y_n$ in r we introduce a new function symbol f_{r,y_j} of the same arity as the head of r. Consider the rule r in $\mathcal{M}$ of the form

$$V(\vec{x}) \rightarrow \bigwedge_{i=1,\ldots,n} R_i(\vec{x_i}, \vec{z_i})$$

For $i = 1, \ldots, n$ we define the inverse rules in the following way

$$R_i(\vec{x_i}, \mathrm{s}(\vec{z_i})) \leftarrow V(\vec{x}).$$

The function s replaces each of the free variables z in $\vec{z_i}$ with the Skolem term $f_{r,z}(\vec{x})$.

▶ **Example 10.** The rule r

$$V(x, y) \rightarrow R_1(x, z) \wedge R_1(z, y)$$

generates the following inverse rules

$$R_1(x, f_{r,z}(x, y)) \leftarrow V(x, y),$$

and

$$R_1(f_{r,z}(x, y), z) \leftarrow V(x, y).$$

Intuitively, the Skolem terms play the role of nulls in the construction of a universal solution. Let $\mathcal{M}^{-1}$ be the set of inverse rules for the mapping $\mathcal{M}$. The rules of $\mathcal{M}^{-1}$ together with the rules from the original query Q form the desired query rewriting $(Q, \mathcal{M}^{-1})$ that can be evaluated over the sources. Note, however, that in the result there may be tuples containing function symbols. Clearly, in the final step all such tuples have to be discarded, similarly to the tuples with nulls in the process of naive evaluation above. We denote the resulting query as $(Q, \mathcal{M}^{-1})_{\downarrow}$.

Although the query $(Q, \mathcal{M}^{-1})_{\downarrow}$ is no longer a datalog query but rather a logic program (it contains function symbols) it can be evaluated in polynomial time w.r.t. size of a source database $\mathbf{D}$ [33]. Indeed, the evaluation has to be performed in two stages - it should start with the inverse rules (they introduce function symbols but they are not recursive) and then the original rules of Q should be applied (they could be recursive but they do not introduce function symbols).

Furthermore, it is possible to eliminate function symbols at all. This is because there are only finitely many function symbols in $(Q, \mathcal{M}^{-1})_{\downarrow}$ which makes it possible to encode them by introducing new predicate names in the datalog query. Therefore $(Q, \mathcal{M}^{-1})_{\downarrow}$ is expressible in datalog.

▶ **Theorem 11** ([33]). *Let $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a data integration setting, let $\mathbf{D}$ be a source database, and let $(Q, \mathcal{M}^{-1})_{\downarrow}$ be a query rewriting constructed as above. Then the rewriting $(Q, \mathcal{M}^{-1})_{\downarrow}$ is maximally-contained (i.e. it is a maximal rewriting in the class of datalog queries), it can be evaluated in polynomial time w.r.t. the size of the sources and it computes the certain answers, i.e.,* $\mathrm{cert}_{Q,\mathcal{I}}(\mathbf{D})$.

Note however, although $(Q, \mathcal{M}^{-1})_{\downarrow}$ is maximally-contained and it computes the certain answers it is not necessarily exact, that is, equivalent to the original query (see Definition 5). Actually, exactness is a joint property of both the rewriting and the data integration setting (see the discussion at the end of Section 4). Here, the problem of checking whether there exists an exact rewriting for a datalog query reduces to the datalog query containment [33] which is undecidable [53]. Nevertheless, $(Q, \mathcal{M}^{-1})_{\downarrow}$ is exact if an exact rewriting expressible either in datalog or as UCQ exists [1].

The inverse rules method can be extended to deal with (global) integrity constraints of the form of full dependencies. Recall that the full dependencies are dependencies of the form

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

where necessarily $\vec{y} \subseteq \vec{x}$, φ is a conjunction of relational and equality atoms and ψ is a relation atom or equality atom. The key point is that there are no existentially quantified variables in the right hand side of a full dependency which guarantees the termination of chase. We construct a set chase(Γ) of datalog rules, one rule for each global full dependency in Γ. The idea is that the application of the rules in chase(Γ) will simulate the chase with (full) tgds and egds.

In presence of egds we need a way of enforcing equalities between variables and functional terms derived during the evaluation of the resulting query. We introduce a new relation E with an aim to capture equality. In order to be able to use E, the query and left hand sides of all dependencies have to be rewritten to their *rectified* version.

We limit ourselves to the following example.

▶ **Example 12.** Consider a query Q

$$\mathrm{Q}(x) \quad :- \quad P(c, x, y, y)$$

In order to use the equalities captured by E, Q is rewritten to its rectified version $\bar{\mathrm{Q}}$, where

- a constant c has to be replaced with fresh variable z and E has to include the pair (c, z),
- the variable x appears in the head, so it is replaced in the body with a fresh variable x' and E has to include (x, x'),
- the variable y appears more than once in Q, the second occurrence is replaced with a fresh variable y' and E has to include (y, y').

$$\bar{\mathrm{Q}}(x) \quad :- \quad P(z, x', y, y') \wedge E(x, x') \wedge E(c, z) \wedge E(y, y')$$

Consider a full dependency with the left hand side rectified:

$$P(x, y) \wedge P(y', z) \wedge E(y, y') \rightarrow P(x, z)$$

where P is a relation symbol in $\mathcal{G}$.

For each such dependency the following, new datalog rule is introduced in chase(Γ).

$$P(x, z) : -P(x, y) \wedge P(y', z) \wedge E(y, y')$$

Finally, the rewriting of a query Q is the union $Q \cup \mathcal{M}^{-1} \cup \text{chase}(\Gamma) \cup \text{equiv}(E)$, where equiv($E$) is the transitivity datalog rule for E: $E(x, y) \leftarrow E(x, z) \wedge E(z, y)$.

▶ **Theorem 13** ([33]). *Let $\mathcal{I} = (\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a data integration setting, let $\mathbf{D}$ be a source database. We assume that the global schema $\mathcal{G}$ includes a set of full dependencies Γ. Consider a (rectified) query Q. Then $(Q, \mathcal{M}^{-1} \cup \text{chase}(\Gamma) \cup \text{equiv}(E))_{\downarrow}$ is a maximally-contained rewriting of Q.*

The restriction to the class of full dependencies guarantees evaluation of the query rewriting in polynomial time however it is more limiting than e.g. if we restrict dependencies to be weakly-acyclic. Note that we could introduce datalog rules for arbitrary tgds but then we have to deal with Skolem terms in recursive rules. Such logic programs need not terminate. Of course, this is nothing unexpected in the light of the results on the chase termination. Note, however, that there are examples of dependencies such that the chase terminates but the bottom-up evaluation of the logic program generated by them does not. E.g. consider the weakly-acyclic dependency $P(x_1, x_2) \rightarrow \exists y P(x_1, y)$ and the corresponding chase rule $P(x_1, f(x_1, x_2)) : -P(x_1, x_2)$.

We can also include special *domain enumeration* rules [33] to deal with access patterns on the views, however we defer it for a while.

3.3 Building the rewriting directly

The approaches to query answering that insist on the materialisation of global data, even if temporary as in the inverse rules method, share a serious disadvantage. The materialisation is not required in data integration and potentially it results in a lot of recomputation. In this section our goal is to modify the query in such a way that it is possible to evaluate it on the sources without materialising the global data.

First, recall that *an expansion* $\exp(Q_{\mathrm{r}})$ of a query Q_{r} over $\mathcal{V}$ is the query over $\mathcal{R}$ that is obtained by substituting every atom $V(\vec{a})$ in Q_{r} with the corresponding view definition $V^{\mathcal{R}}(\vec{a})$.

We present methods for computing maximally-contained query rewritings that are based on the analysis of how the atoms of a query can be mapped to the atoms of the expansion of the rewriting. Such mapping is called the *containment mapping* [21] and it necessarily exists since it witnesses the containment of the rewriting in the query.

We assume a LAV data integration system $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ and that the user query Q and the queries in $\mathcal{M}$ are conjunctive queries (CQs). We say that an atom $R(\vec{x})$ of Q *is covered* by an atom $R(\vec{y})$ of a query $\mathcal{V} \in \mathcal{M}$ if there is a mapping $\theta : \vec{x} \to \vec{y}$ such that $\theta(\vec{x}) = \vec{y}$.[1] Intuitively, such partial mappings θ can be used to build the containment mapping.

We start with the following result

▶ **Theorem 14** ([42]). *Let $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a LAV data integration system and let the user query Q and the queries in $\mathcal{M}$ be conjunctive queries (CQs). If the body of Q has n atoms, and Q' is a CQ-maximal rewriting (it contains all other rewritings that are CQ), then Q' has at most n atoms.*

The result follows from the fact that the atoms of the expansion $\exp(Q')$ must cover the atoms of Q. Since $\exp(Q') \subseteq Q$ then there is a homomorphism h from (the canonical structure for) Q to (the canonical structure for) $\exp(Q')$. Clearly, each atom in Q is mapped by h to at most one atom in $\exp(Q')$. Thus, if Q' contains more than n atoms then the expansion of at least one atom of Q' is disjoint from the image of h. Hence, the atom can be deleted and this contradicts the maximality of Q'.

This gives us the *brute-force* algorithm for the construction of maximally-contained rewriting in the class of unions of conjunctive queries. It is enough to consider all possible conjunctions of n or fewer atoms.

▶ **Theorem 15** ([42], [1]). *Let $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a LAV data integration system, let Q and the queries in $\mathcal{M}$ be conjunctive queries (CQs) and let Q_r be the union of all CQ-maximal rewritings for Q. Then*
1. *Q_r is the UCQ-maximal rewriting,*
2. *Q_r can be evaluated in polynomial time w.r.t. the size of the sources,*
3. *Q_r is perfect (i.e. it computes exactly the certain answers),*
4. *Q_r is exact if an exact rewriting (in the class of UCQs) exists.*

The *bucket algorithm* [43] and its improved versions *MiniCon algorithm* [51], *SVB algorithm* [48] provide better ways of constructing maximally-contained rewritings. Let $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a LAV data integration system and let Q and the queries (views) $\mathcal{V} = V_1, \ldots, V_n$ in $\mathcal{M}$ be conjunctive queries (CQs). The key idea is to analyse how the atoms of the expansion of a conjunctive rewriting can cover the atoms of the user query. We say that a variable is *distinguished* if it appears in the query head or in the head of a view definition. Existential variables are called *nondistinguished* (or *local*). We say that a variable in Q is *shared* if it it appears more than once in Q. There are a few simple conditions that a contained rewriting has to preserve.

C0 All atoms of Q have to be covered by atoms of the expansion of a rewriting.
C1 A distinguished variable in Q has to be mapped to a distinguished variable in some V_i in the expansion.
C2 A shared variable x of Q has to be mapped either to a distinguished variable or all atoms of Q involving x have to be covered by atoms in the expansion of a single V_i.

The example below illustrates the motivation for C2.

▶ **Example 16.** Consider a query $Q(x, z) \leftarrow P(x, y) \wedge P(y, z)$ and a mapping consisting of the two queries:

$$V_1(v) \to P(v, w), \qquad V_2(t) \to P(u, t)$$

[1] Such mapping does not exist only if some x appears twice in $\vec{x}$ (or because of constants if they are present).

Then the rewriting $V_1(x) \wedge V_2(z)$ is not contained in Q. This is because its expansion is $P(x,w) \wedge P(u,z)$, where w and u are fresh variables. Clearly, there is no way to equate the local variables w and u at the level of rewriting. Note that the variable y of Q is shared by two atoms in Q but since it is not in the head of Q it gets mapped to local variables of V_1 and V_2. Unfortunately, the bucket algorithm is not aware of the condition C2.

The bucket algorithm creates a bucket for each of the atoms of Q. The bucket for an atom $R(\vec{s})$ in Q contains the heads of the queries in $\mathcal{M}$ that include atoms to which $R(\vec{s})$ can be mapped. A query V_k can appear in the bucket for $R(\vec{s})$ multiple times if $R(\vec{s})$ can be mapped to more then one atom in V_k. Let $\vec{s} = s_1, \dots, s_n$ and $\vec{t} = t_1, \dots, t_n$.[2] An atom $R(\vec{t})$ in V_k is placed in the bucket for an atom $R(\vec{s})$ in Q if and only if:

- $R(\vec{s})$ and $R(\vec{t})$ unify (C0). [3]
- if x_i is in the head of Q then y_i has to be in the head of V_k (C1).

If both conditions are satisfied we insert into the bucket for $R(\vec{x})$ the new atom $\theta(\text{head}(V_k))$, where θ is the unifier and all variables of V_k not in the domain of θ (i.e. not mentioned in $R(\vec{y})$) are fresh.

After constructing the buckets the algorithm starts generating candidate rewritings. It considers every rewriting formed by a conjunction of atoms, one atom from each of the buckets. Effectively, it means that all elements of the Cartesian product of the buckets are enumerated. Note that because C0 and C2 are not enforced the algorithm has to check each of the rewritings whether either it is contained in Q or can be made to be contained by equating some of its variables. Finally, the resulting rewriting is the union of all contained conjunctive rewritings.

The bucket algorithm has two drawbacks - it generates a lot of candidate rewritings and then it performs an expensive containment test for each of them. This is because each of the atoms in Q is considered in isolation. On the other hand the MiniCon algorithm focuses on the interactions between the variables in the query and the mapping and enforces each of the conditions C0-C2. Again, let $(\mathcal{G}, \mathcal{S}, \mathcal{M})$ be a LAV data integration system and let Q and the views $V_1, \dots, V_n$ in $\mathcal{M}$ be conjunctive queries (CQs). The algorithm constructs a kind of generalised buckets, called *MiniCon Descriptions (MCDs)*, however once it discovers that an atom of some V_i can cover an atom of Q it finds the minimal, additional set of query atoms that have to be covered together. Recall, that in the bucket algorithm we have a single bucket for each atom of Q and such buckets consist of atoms over V_i. Here, a MCD C is formed for a particular atom $V(h(\vec{x}))$, where h equates some variables of $\vec{x}$ (i.e. we may use $V(x,y,x)$ in the rewriting instead of $V(x,y,z)$). Formally, a MCD C for Q and V consists of

- a head homomorphism h,
- the atom $V(h(\vec{x}))$,
- a partial mapping θ from the variables of Q to the variables of V.
- a subset G_C of atoms of Q that are covered by some atom in the query $V(h(\vec{x}))$ using θ.

Note that the problem of bucket algorithm with enforcing the condition C0 is avoided by splitting each of the considered unifiers into a separate head homomorphism h and a mapping θ.

[2] Both $\vec{s}$ and $\vec{t}$ are tuples of constants and variables

[3] Actually, note that this does not guarantee that $R(\vec{t})$ can cover $R(\vec{s})$. The problem arises when the unifier equates a variable of $\vec{t}$ (in the head of V_k or not) with another variable of $\vec{t}$ that is not in the head of V_k (i.e. local variable). Then in the second phase if we use V_k in a rewriting we cannot enforce the equality since we have no access to the local variables at the level of rewriting. We verify this with the containment tests in the second phase.

The algorithm constructs a set $\mathcal{C}$ of MCDs in such a way that for each $C \in \mathcal{C}$, where $C = (h, V(h(\vec{x})), \theta, G_C)$ the conditions below hold

- for each head variable x in Q which is in the domain of θ, $\theta(x)$ is the head variable in $V(h(\vec{x}))$ (C1),
- If an existential variable x of Q is in the domain of θ and is shared by two or more atoms in Q then each of the atoms must be covered by $V(h(\vec{x}))$ (C2).
- The set of atoms G_C has to be the minimal one (i.e. it is not possible to cover a subset of the atoms in G by $V(h(\vec{x}))$ even if θ or h are extended).

The third condition, although not essential, allows for the optimisation of the second phase. Namely, the algorithm outputs a union of all rewritings that are combinations of MCDs $C_1, \ldots C_k$ such that the sets of covered atoms $G_{C_1}, \ldots, G_{C_k}$ form a partition of the set of atoms of Q. [4] Finally, it can be proved that the expansion of each of the rewritings is contained in Q. Hence, containment tests are not necessary.

3.4 Dealing with access patterns

Now, we show how to deal with a situation where access to data sources is limited. In practice, it may happen that some sources can answer only such queries where some variables are bound. For example, consider a source relation `owns`(`person`, `residence`) storing information about owners of residences. The source accepts queries where `residence` is bound to a given value (i.e. it is not a free variable). That is, users can ask who owns a given residence but is is forbidden to ask for a list of residences owned by a given person or for a list of all person-residence pairs.

In order to model such limitations we introduce *access patterns*. Formally, *an access pattern* for a k-ary relation V in the source schema is an expression V^{α} where α is a word of length k over the alphabet $\{\mathrm{i}, \mathrm{o}\}$, where 'i' stands for *input slot* (only bound values) and 'o' stands for *output slot* (no value required).

We say that a datalog query Q is *executable* if every variable of a rule appears first (when reading from left to right) in a positive atom in an output slot in the body of the rule. E.g. query $V^{iio}(0, 0, x) \wedge V'^{i}(x)$ is executable, while query $V^{iio}(x, 0, 0)$ is not.

It turns out that, if we allow recursive rules in the rewriting, access patterns do not require special treatment in data integration system. Indeed, if recursion is allowed in the rewriting, then we can easily enumerate the whole active domain [33]. If recursion is not allowed, then we can simulate it with the chase, first transforming the domain enumeration rules into dependencies [23].

Given a source schema $\mathcal{S}$, let $\mathrm{domain}_{\mathcal{S}}$ be the unary recursive query with one rule of the form

$$\mathrm{domain}_{\mathcal{S}}(x_j) \leftarrow \mathrm{domain}_{\mathcal{S}}(x_{i_1}) \wedge \mathrm{domain}_{\mathcal{S}}(x_{i_2}) \wedge \ldots \wedge \mathrm{domain}_{\mathcal{S}}(x_{i_k}) \wedge V(\vec{x})$$

for each relation V in $\mathcal{S}$ with an access pattern α where $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ are in the input slots and x_j is in the output slot. Note that all-output access patterns give non-recursive rules and if all access patterns are input only then $\mathrm{domain}_{\mathcal{S}}$ is empty. Now a query $\mathrm{dext}(Q)$ for a query Q is given by the rules for $\mathrm{domain}_{\mathcal{S}}$ and the following rule

$$\mathrm{dext}(Q)(\vec{x}) \leftarrow \mathrm{domain}_{\mathcal{S}}(y_1), \ldots, \mathrm{domain}_{\mathcal{S}}(y_k) \wedge Q,$$

where the head of Q is $Q(\vec{x})$ and y_i are the variables in the body of Q.

Clearly, $\mathrm{dext}(Q)$ is executable and contained in Q.

[4] In particular the combinations cover mutually exclusive sets of atoms of Q.

Each rule of the query $\mathrm{domain}_{\mathcal{S}}$ as defined above can be captured with a constraint

$$\mathrm{domain}_{\mathcal{S}}(x_{i_1}) \wedge \mathrm{domain}_{\mathcal{S}}(x_{i_2}) \wedge \ldots \wedge \mathrm{domain}_{\mathcal{S}}(x_{i_k}) \wedge V(\vec{x}) \rightarrow \mathrm{domain}_{\mathcal{S}}(x_j).$$

For every relation V^α in $\mathcal{S}$ we define a constraint $\sigma_{V,\alpha}$

$$\mathrm{domain}_{\mathcal{S}}(\vec{x}) \wedge V(\vec{x}) \rightarrow V'(\vec{x}).$$

We denote all above constraints by $\mathcal{M}$. The constraints in $\mathcal{M}$ are over the source schema $\mathcal{S}$ and the global schema $\mathcal{G}$ but note that they are not source-to-target since $\mathrm{domain}_{\mathcal{S}}$ appears in both sides of some constraints.

▶ **Theorem 17** ([23]). *For a data integration setting $(\mathcal{G}, \mathcal{S}, \mathcal{M})$, where $\mathcal{S}$ is with access patterns, for any UCQ query Q and for a source database $\mathbf{D}$, the query $\mathrm{dext}(Q)(\vec{x})$ computes $\mathrm{cert}_{Q',(\mathcal{G},\mathcal{S},\mathcal{M})}(\mathbf{D})$ where Q' is obtained from Q by replacing every symbol V in $\mathcal{S}$ with the corresponding symbol V'.*

3.5 Chasing the query

Now we discuss the problem of rewriting queries using the views from a dual perspective. We follow ideas first presented in [25] (see also [26]). The approach results in query rewritings that are overestimations of the exact answers. Still, it turns out that such query rewritings can be computed with the use of the chase. This time, however, we chase the queries, and not the data as before.

Recall that in the most common setting LAV mappings consist of dependencies of the form

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

where V is a symbol in $\mathcal{S}$ and $\psi(\vec{x},\vec{y})$ is a query over $\mathcal{G}$. Such dependencies enforce the view V to be sound. Here, we essentially make use of the implication in the other direction. Namely, the dependencies in the mapping are of the form

$$\forall\vec{x},\vec{y}\, \psi(\vec{x},\vec{y}) \rightarrow V(\vec{x})$$

where V is a symbol in $\mathcal{S}$ and $\psi(\vec{x},\vec{y})$ is a query over $\mathcal{G}$. Such dependencies enforce the views V to be complete. Note that we may enforce exactness of views with the use of both kinds of dependencies at the same time. Recall also the result presented in Table 2 - computing certain answers under exact views is coNP-complete for CQ queries and views. Such a setting is called LAV with exact sources.

The following algorithm illustrates the idea of the approach limited to the case with conjunctive queries (CQs). Recall that by $\mathbf{C_Q}$ we denote the canonical structure for a CQ Q.

▶ **Algorithm 2** (ViewRewrite($(\mathcal{G},\mathcal{S},\mathcal{M}), Q(\vec{x})$)).

```
Input:
  - a LAV data integration setting $(\mathcal{G},\mathcal{S},\mathcal{M})$ with exact sources,
  - a conjunctive query $Q(\vec{x})$ over $\mathcal{G}$.
Let $Q'(\vec{x})$ be the empty query (with no atoms).
For each rule in $\mathcal{M}$ of the form $\forall\vec{x},\vec{y}\,\psi(\vec{x},\vec{y}) \rightarrow V(\vec{x})$
  for each tuple $\vec{a}$ such that
    - there is a tuple $\vec{b}$ such that $\mathbf{C_Q}$ satisfies $\psi(\vec{a},\vec{b})$; and
    - $V(\vec{a})$ is not an atom of $Q'$.
   add $V(\vec{a})$ to the conjunction $Q'$ as a new atom.
$Q^{\mathcal{M}}(\vec{x}) := Q'$
Output: the query $Q^{\mathcal{M}}(\vec{x})$ over $\mathcal{S}$.
```

What can we say about the properties of $Q^{\mathcal{M}}$? Consider $\exp(Q^{\mathcal{M}})$, the expansion of $Q^{\mathcal{M}}$. Recall that $\exp(Q^{\mathcal{M}})$ is obtained by substituting every atom in $Q^{\mathcal{M}}$ with the corresponding view definition from $\mathcal{M}$. Clearly, in $\exp(Q^{\mathcal{M}})$ the symbols in $\mathcal{S}$ are replaced with the queries over $\mathcal{G}$. Now, note that there is always a containment mapping from $\exp(Q^{\mathcal{M}})$ to Q. Therefore, $Q^{\mathcal{M}}$ is complete but may be unsound. That is, it returns all tuples in $\mathrm{cert}_{Q,\mathcal{I}}(\mathbf{D})$ for every $\mathbf{D}$ but it may also return some incorrect tuples. Such rewritings are called *containing rewritings.* Furthermore, it can be shown that $Q^{\mathcal{M}}$ is contained in any other containing rewriting, thus it is the best containing rewriting possible. We call such rewritings *minimally-containing.*

We illustrate the algorithm with the following example.

▶ **Example 18.** Consider the query $Q(x_1, x_2) : -R(x_1, z_1) \wedge R(z_1, z_2) \wedge R(z_2, z_3) \wedge R(z_3, x_2)$ and the dependencies $R(y_1, a) \wedge R(a, y_2) \to V_1(y_1, y_2) \in \mathcal{M}_1$ and $R(y_1, z_1) \wedge R(z_1, z_2) \wedge R(z_2, y_2) \to V_2(y_1, y_2) \in \mathcal{M}_2$. Q is the path of length 4, V_1 is the path of length 2 and V_2 is the path of length 3. Then $Q^{\mathcal{M}_1}(x_1, x_2) : -V_1(x_1, z_2) \wedge V_1(z_1, z_3) \wedge V_1(z_2, x_2)$ and $Q^{\mathcal{M}_2}(x_1, x_2) : -V_2(x_1, z_3) \wedge V_2(z_1, x_2)$.

Note that $\exp(Q^{\mathcal{M}_1})$ is equivalent to Q while $\exp(Q^{\mathcal{M}_2})$ is not sound.

Finally, note that $\exp(Q^{\mathcal{M}})$ can be computed by the chase with the use of the rules with implications in the other direction (i.e. of the form $\forall \vec{x}\, V(\vec{x}) \to \exists \vec{y}\, \psi(\vec{x}, \vec{y})$.

The construction can be developed further, the paper [23] extends the result to UCQ$^{\neg}$ queries both in the mapping (i.e. view definitions) and as the user queries. Moreover, a broad class of integrity constraints in the global schema is allowed as well as the access patterns on views. The extension requires a careful technical development, but the basic idea of chasing the query remains and offers a unified treatment of the three flavours of rewriting problems: using views, with access patterns and under integrity constraints.

3.6 Determinacy and rewriting

The question of whether a query Q can be answered using a set of views $\mathcal{V}$ can be formulated at several levels: A language-specific formulation is given by the notion of query rewriting, while a more general formulation is given by the information-theoretic notion of *determinacy*, which is formally defined as follows.

▶ **Definition 19** (Determinacy)**.** Given a set of views $\mathcal{V}$ and a query Q over a database signature $\mathcal{R}$, we say that $\mathcal{V}$ *determines* Q (written $\mathcal{V} \twoheadrightarrow Q$) iff, for every two database instances $\mathbf{D}_1$ and $\mathbf{D}_2$, we have that $Q(\mathbf{D}_1) = Q(\mathbf{D}_2)$ whenever $\mathcal{V}^{\mathcal{R}}(\mathbf{D}_1) = \mathcal{V}^{\mathcal{R}}(\mathbf{D}_2)$.

Intuitively, determinacy says that the views provide enough information to uniquely determine the answer to the query, but without specifying whether this can be done effectively or using a particular query language. Thus, the question of what is the relationship between determinacy and rewriting arises quite naturally. On the one hand, if a query Q has an exact rewriting Q_{r} under (exact) views $\mathcal{V}$, then $\mathcal{V} \twoheadrightarrow Q$; the converse, on the other hand, is in general not true, leading to the following definition.

▶ **Definition 20** (Complete rewriting language)**.** A rewriting language $\mathcal{L}_{\mathrm{r}}$ is *complete* for $\mathcal{L}_{\mathrm{v}}$-to-$\mathcal{L}_{\mathrm{q}}$ iff $\mathcal{L}_{\mathrm{r}}$ can be used to express an exact rewriting of a query $Q \in \mathcal{L}_{\mathrm{q}}$ using views $\mathcal{V}$ defined in $\mathcal{L}_{\mathrm{v}}$ whenever $\mathcal{V} \twoheadrightarrow Q$.

An interesting case is when the language $\mathcal{L}_{\mathrm{q}}$ is itself complete for $\mathcal{L}_{\mathrm{v}}$-to-$\mathcal{L}_{\mathrm{q}}$, because in such a situation the query language needs not to be extended in order to take advantage of the available views. A thorough investigation of determinacy and its connection to rewriting is carried out in [49] for relational data, in a setting with exact views and exact rewritings where, for view languages $\mathcal{L}_\text{v}$ and query languages $\mathcal{L}_\text{q}$ ranging from first-order logic to conjunctive queries, the following questions are studied:

1. Is it decidable whether $\mathcal{V} \twoheadrightarrow Q$ for $\mathcal{V}$ in $\mathcal{L}_\text{v}$ and Q in $\mathcal{L}_\text{q}$?
2. Is $\mathcal{L}_\text{q}$ complete for $\mathcal{L}_\text{v}$-to-$\mathcal{L}_\text{q}$ rewritings? If not, what is the minimal extension of $\mathcal{L}_\text{q}$ in which such rewritings can be expressed?

Here, we summarise the main results for the following languages: First-order logic (FO); conjunctive queries (CQ) without equality, inequality and constants; unions of conjunctive queries (UCQ). For additional languages (e.g., existential FO) and further details see [49]. We start by reporting a general result establishing that determinacy becomes undecidable as soon as the query language $\mathcal{L}_\text{q}$ is powerful enough so that satisfiability is undecidable, or as soon as the view language $\mathcal{L}_\text{v}$ is such that validity is undecidable.

▶ **Proposition 2.** *If satisfiability of sentences in $\mathcal{L}_q$ is undecidable or validity of sentences in $\mathcal{L}_v$ is undecidable, then it is undecidable whether $\mathcal{V} \twoheadrightarrow Q$ for Q in $\mathcal{L}_q$ and $\mathcal{V}$ defined in $\mathcal{L}_v$.*

Clearly, a direct consequence of Proposition 2 is that determinacy is undecidable whenever queries or view definitions are expressed in FO. For what instead concerns rewritings, it turns out that in the unrestricted case, that is, when possibly infinite instances are allowed, FO is complete for FO-to-FO rewritings, but unfortunately this does not hold anymore when considering finite instances only. Indeed, in the restricted case, any language that is complete for FO-to-FO rewritings must express all computable queries.

Determinacy remains undecidable also for much weaker languages than FO. In fact, it has been shown in [49] that determinacy is undecidable for UCQs in a quite strong way, as the undecidability result holds for a fixed database schema and a fixed set of views. Concerning rewritings, since the question of whether a UCQ query has a UCQ rewriting in terms of a set of UCQ views is decidable [42], we also have that UCQ is not complete for UCQ-to-UCQ rewritings, otherwise we would get a contradiction of the undecidability of determinacy for UCQs mentioned above. Indeed, the following theorem from [49] shows that no monotonic language can be complete even for UCQ-to-CQ rewritings.

▶ **Theorem 21.** *Any language that is complete for UCQ-to-CQ rewritings must express non-monotonic queries.*

Proof. Consider the database signature $\mathcal{R} = \{r_1, r_2\}$ with r_1 and r_2 unary and the view signature $\mathcal{V} = \{v_1, v_2\}$ with the following view definitions:

$$v_1{}^{\mathcal{R}} \colon \exists u\, r_1(u) \wedge r_2(x) \;; \qquad\qquad v_2{}^{\mathcal{R}} \colon r_1(x) \vee r_2(x) \;.$$

Let Q be the query $r_2(x)$. It is easy to see that the views $\mathcal{V}$ determine the answer to Q. In fact, for any database $\mathbf{D}$, we have the following two cases:

- If $r_1(\mathbf{D}) \neq \varnothing$, then $\exists z\, r_1(z)$ is always true and $Q(\mathbf{D}) = v_1(\mathbf{D})$, that is, the answer to Q is provided by v_1;
- If $r_1(\mathbf{D}) = \varnothing$, then $v_2(\mathbf{D}) = r_1(\mathbf{D}) \cup r_2(\mathbf{D}) = r_2(\mathbf{D}) = Q(\mathbf{D})$, that is, the answer to Q is provided by v_2.

Therefore, $\mathcal{V} \twoheadrightarrow Q$. Now, let $\mathbf{D}_1$ be a database such that $r_1(\mathbf{D}_1) = \{a, b\}$ and $r_2(\mathbf{D}_1) = \varnothing$, and let $\mathbf{D}_2$ be a database for which $r_1(\mathbf{D}_2) = \{a\}$ and $r_2(\mathbf{D}_2) = \{b\}$. Then, we have that $v_1(\mathbf{D}_1) = \varnothing \subseteq \{a\} = v_1(\mathbf{D}_2)$ and $v_2(\mathbf{D}_1) = \{a, b\} = v_2(\mathbf{D}_2)$. However, $Q(\mathbf{D}_1) = \{a, b\} \not\subseteq \{a\} = Q(\mathbf{D}_2)$. Hence, the mapping that for each database $\mathbf{D}$ associates the query answer $Q(\mathbf{D})$ to the corresponding extension $\mathcal{V}^{\mathcal{R}}(\mathbf{D})$ is non-monotonic. ◀

The above proof shows that Theorem 21 holds even if the database relations, views and queries are restricted to be unary.

Whether a CQ query can be rewritten as another CQ query in terms of a set of CQ views is decidable [42]. Hence, if CQ were complete for CQ-to-CQ rewritings, we would immediately have a decision procedure also for the determinacy for CQ queries and views. However, in both [49] and [3] it is unfortunately shown that CQ is not complete for CQ-to-CQ rewritings and, indeed, the former also show that no monotonic language is. The proof in [3] is of particular interest in that it provides an infinite set of examples of CQ views and queries for which the views determine the query but the query has no CQ rewriting in terms of the views. The examples involve path queries $Q_n(x, y)$ on a binary relation r returning the pairs $\langle x, y\rangle$ for which there is an R-path of length n from x to y (for more information on path queries see Section 4). For instance, it can be shown that $\{Q_3, Q_4\} \twoheadrightarrow Q_5$ and that Q_5 has the following FO rewriting:

$$Q_5(x, y) \equiv \exists z\ Q_4(x, z) \wedge \forall v\ Q_3(v, z) \rightarrow Q_4(v, y)\ \ ,$$

but Q_5 has no CQ rewriting in terms of Q_3 and Q_4. For what concerns the decidability of determinacy for CQ views and queries, the problem remains open.

We conclude this section by presenting a well-behaved query language with respect to determinacy and rewriting, namely the so-called *packed fragment* (PF) of FO, identified and studied by Marteen Marx in [47]. We first make a short digression to introduce another fragment of FO, called the *guarded fragment* (GF), of which PF is a useful extension. GF is formally defined as the smallest set such that:

- it contains every first order atom over a given relational signature,
- it is closed under the boolean connectives, and
- it is closed under the following rule for quantified formulas: if $\phi(\overline{x}, \overline{y})$ is in GF, then so are also $\exists\overline{y}\ \big(G \wedge \phi(\overline{x}, \overline{y})\big)$ and $\forall\overline{y}\ \big(G \rightarrow \phi(\overline{x}, \overline{y})\big)$, provided that G is an atomic formula, called the *guard*, in which all variables $\overline{x}$ and $\overline{y}$ occur free.

In other words, GF allows only for quantified formulas of the form $\forall\overline{y}\ \big(G(\overline{x}, \overline{y}) \rightarrow \phi(\overline{x}, \overline{y})\big)$,[5] where G is an atomic relation symbol and $\phi(\overline{x}, \overline{y})$ is also in GF.

The key requirement in GF is that all of the variables occurring free in the subformula ϕ must also occur in the guard. On the one hand, such a restriction ensures nice logical properties to this fragment, in particular that the validity problem is decidable in 2EXPTIME; on the other hand, however, it strongly limits its expressivity. Intuitively, guarding corresponds to restricting the tuples that can be generated by queries to tuples whose elements form a sub-tuple[6] in some atomic relation, therefore views defined in GF always consist of sub-tuples of a relation in the database. For instance, we can define the view ICT-Employee(name) from the tables Employee(emp, id) and Department(id, dep) in the guarded fragment, as shown in the example below, but we cannot define the binary relation WorksFor(emp, dep) from these two tables.

▶ **Example 22.** The exact view definition

$$\forall x\ \big[\,\mathsf{ICT\text{-}Employee}(x) \leftrightarrow \exists y\ \big(\mathsf{Employee}(x, y) \wedge \mathsf{Department}(y, \text{“ICT”})\big)\,\big]$$

[5] The variables in $\overline{x}$ are free, while the ones in $\overline{y}$ are universally quantified: when $\overline{x}$ is empty we have a GF closed formula; when $\overline{y}$ is empty we have an open formula without outermost quantification.
[6] That is, a projection over some of the elements in tuple.

is equivalent to (the conjunction of) the following two formulae:

$$\begin{aligned} &\forall x, y \ \big[\big(\mathsf{Employee}(x,y) \land \mathsf{Department}(y, \text{“ICT”})\big) \to \mathsf{ICT\text{-}Employee}(x) \big] \ , \\ &\forall x \quad \big[\mathsf{ICT\text{-}Employee}(x) \to \exists y \, \big(\mathsf{Employee}(x,y) \land \mathsf{Department}(y, \text{“ICT”})\big) \big] \ , \end{aligned}$$

which are both in GF since the first one can be rewritten as

$$\forall y \, \big[\mathsf{Department}(y, \text{“ICT”}) \to \forall x \, \big(\mathsf{Employee}(x,y) \to \mathsf{ICT\text{-}Employee}(x)\big) \big] \ .$$

From a database perspective, GF coincides with the *semijoin algebra*, which is obtained from Codd's relational algebra by replacing the product operator with the semijoin operator $\ltimes$ [40]. Indeed, differently from the natural join, the semijoin $R \ltimes S$ always returns a subset of the tuples in R. As the most common type of queries occurring in practise are CQs, it is interesting to remark that the guarded conjunctive queries (i.e., the CQs that are expressible in GF) are precisely the *acyclic conjunctive queries* [36]. A conjunctive query Q is acyclic iff so is its hypergraph, having as vertices all the variables occurring (free or bound) in Q and, for each atom $A(\overline{x})$ in Q, an hyperedge consisting of all the variables in $\overline{x}$. In turn, the hypergraph of a query is acyclic iff it can be reduced to the empty one by repeatedly deleting vertices that occur in exactly one hyperedge and deleting hyperedges that are contained in some other one [2].

The packed fragment extends GF by allowing guards of the form:

$$G(\overline{x}) = \bigwedge_k \exists \overline{y} \ A_k(\overline{x}, \overline{y}) \ , \tag{9}$$

where each A_k is an atom and, for every pair of distinct variables x_i and x_j, $G(\overline{x})$ contains an atom A_k in which x_i and x_j both occur free. A guard of this kind is a “safe product” whose hypergraph is such that any two of its vertices belong together to an hyperedge. When all the relation symbols are binary, the hypergraph is in fact a complete graph, that is, a clique. For example, the query $\mathsf{Employee}(x,z) \land \mathsf{Department}(z,y) \land \mathsf{WorksFor}(x,y)$ is a safe product and its hypergraph is indeed a clique, whereas $\exists z \, \big(\mathsf{Employee}(x,z) \land \mathsf{Department}(z,y)\big)$ is not because there is no atom connecting the variables x and y. The former query is expressible in PF but not in GF, as its hypergraph is cyclic. Indeed, PF is strictly more expressive than GF: the “until” operator of temporal logic is another example of a FO formula that can be expressed in PF but not in GF.

As for the positive properties, the validity problem in PF is 2EXPTIME-complete and every satisfiable PF formula is satisfiable on a finite model. The packed fragment enjoys good properties also w.r.t. determinacy and rewriting. In fact, as shown in [47], determinacy for PF queries and views is 2EXPTIME-complete and PF is complete for PF-to-PF rewritings. Moreover, the completeness result holds in addition for (unions of) packed conjunctive queries (PCQ), that is, formulas obtained from relation symbols and equality using only conjunction and existential quantification.[7] Indeed, we also have that PCQ is complete for PCQ-to-PCQ rewritings and UPCQ is complete for UPCQ-to-UPCQ rewritings. Other well-behaved classes of CQ queries and views, but orthogonal to PF, are reported in [49].

4 Query processing in semistructured data integration

Semistructured databases capture data that do not fit into rigid, predefined schemas, and are best described by means of graph-based data models. Indeed, a *semistructured database* is a

[7] Note that, although called packed conjunctive queries for simplicity, such formulas cannot always be written in prenex form due to syntactic restrictions.

Table 3 Summary of the major results on determinacy and completeness for rewritings.

LANGUAGE $\mathcal{L}$	DETERMINACY	COMPLETE FOR $\mathcal{L}$-TO-$\mathcal{L}$ REWRITINGS?	
		FINITE	UNRESTRICTED
FO	undecidable	✗	✓
UCQ	undecidable	✗	✗
CQ	open	✗	✗
PF	2EXPTIME-complete	✓	✓
(U)PCQ	2EXPTIME-complete	✓	✓

finite relational structure over a signature $\mathcal{R}$ of binary symbols, that can be seen as a finite directed graph where each node is an element of the domain and each edge is labelled by a symbol $r \in \mathcal{R}$. We denote by $r(x, y)$ an edge from x to y labelled by r, representing the fact that relation r holds between objects x and y. Clearly, in order to extract information from this kind of data model, special querying mechanisms are required that are not common in traditional database systems.

A *regular-path query* (RPQ) is a binary query defined in terms of a regular language over $\mathcal{R}$. In particular, the answer to an RPQ Q over a semistructured database $\mathbf{D}$ is the set $Q(\mathbf{D})$ of pairs of nodes connected in $\mathbf{D}$ by a directed path traversing a sequence of edges that form a word in the regular language $\mathcal{L}(Q)$ defined by Q. A *two-way regular-path query* (2RPQ) is an RPQ extended with the ability of traversing edges backwards when navigating in a semistructured database. Formally, a 2RPQ over $\mathcal{R}$ is defined in terms of a regular language over the alphabet $\mathcal{R}^{\pm} = \mathcal{R} \cup \{r^- \mid r \in \mathcal{R}\}$ obtained by adding, for each $r \in \mathcal{R}$, a new relation symbol r^- that denotes the *inverse* of r. In addition, the standard notion of path in a graph is replaced by the notion of "semipath", that is, a navigation of the database from some node to another according to a sequence of edge labels, in which edges are traversed forward or backward depending on whether the corresponding edge label is direct (i.e., some $r \in \mathcal{R}$) or inverse (i.e., some r^-). Then, the answer to a 2RPQ Q over a semistructured database $\mathbf{D}$ is given by the pairs of objects connected in $\mathbf{D}$ by a semipath conforming to the regular language defined by Q. Examples of RPQ and 2RPQ are given below.

▶ **Example 23.** Let $\mathbf{D}$ be the semistructured database over the signature $\mathcal{R} = \{r_1, r_2, r_3\}$ shown in Figure 1, and consider the RPQ $Q = (r_1 + r_2) \cdot r_3$ and the 2RPQ $Q' = (r_1 + r_2) \cdot r_3{}^-$ over $\mathcal{R}$. Then, the answers Q and Q' produce when evaluated over $\mathbf{D}$ are respectively given by $Q(\mathbf{D}) = \{\langle a, d\rangle, \langle c, d\rangle\}$ and $Q'(\mathbf{D}) = \{\langle a, b\rangle, \langle c, b\rangle\}$.

In this section, we study view-based query processing in the context of semistructured data. First, we focus on the query rewriting approach by presenting a method, proposed in

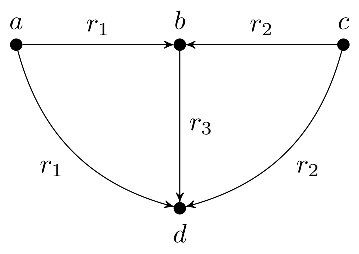


Figure 1 The semistructured database $\mathbf{D}$ of Example 23.

[17], for rewriting a query expressed as a regular expression (i.e., an RPQ) in terms of a set of views also expressed as regular expressions. Since RPQs are essentially regular expressions, the presented technique is directly applicable for them, and it is extended to 2RPQs in [20] using two-way automata to deal with inverse. Then, following [19], we explain and clarify the relationship between rewriting and answering, highlighting the distinction between them in the context of RPQs and 2RPQs. Indeed, the expressive richness of these languages allows to point out subtle differences between answering and rewriting that are blurred in the case of conjunctive queries. Lastly, we deal with the issue of trying to understand whether processing a query based on a set of views causes loss of information. We will introduce different notions for assessing "losslessness" w.r.t. answering and w.r.t. rewriting, and discuss the distinction and the relationship between them.

4.1 Rewriting of regular expressions

In this section, we present a method [17] for computing the rewriting of a regular expression (RE) in terms of other regular expressions. We consider a finite alphabet $\mathcal{R}$ of database symbols and a finite alphabet $\mathcal{V}$ of view symbols. Each view symbol $v \in \mathcal{V}$ is associated with a RE $v^{\mathcal{R}}$ over $\mathcal{R}$ that defines v in terms of the symbols in $\mathcal{R}$. Given a query Q expressed as a RE over $\mathcal{R}$, we want to reformulate it (if possible) by a suitable combination of the view symbols. The *expansion* of a language $\mathcal{L}$ over $\mathcal{V}$ is the language $\exp(\mathcal{L})$ over $\mathcal{R}$ consisting of all the words obtained from a word $v_1 \cdots v_n \in \mathcal{L}$ by substituting each v_i with every possible word of the regular language defined by the RE associated with v_i. In symbols:

$$\exp(\mathcal{L}) = \bigcup_{v_1 \cdots v_n \in \mathcal{L}} \{w_1 \cdots w_n \mid w_i \in \mathcal{L}\big(v_i^{\mathcal{R}}\big)\} \ . \tag{10}$$

The expansion of a $\mathcal{V}$-word w is given by $\exp(\{w\})$.

Let Q_{r} be an expression defining a language $\mathcal{L}(Q_{\mathrm{r}})$ over $\mathcal{V}$: We say that Q_{r} is a *rewriting* w.r.t. $\mathcal{V}$ of a RE Q over $\mathcal{R}$ iff $\exp\big(\mathcal{L}(Q_{\mathrm{r}})\big) \subseteq \mathcal{L}(Q)$. A rewriting Q_{r} of Q w.r.t. $\mathcal{V}$ is said to be *exact* when $\exp\big(\mathcal{L}(Q_{\mathrm{r}})\big) = \mathcal{L}(Q)$. Obviously, we are interested in capturing the language defined by the original RE at best, that is, in finding rewritings that are maximal. Formally, a rewriting Q_{r} of Q w.r.t. $\mathcal{V}$ is $\mathcal{R}$*-maximal* iff every rewriting Q'_{r} of Q w.r.t. $\mathcal{V}$ is such that $\exp\big(\mathcal{L}(Q'_{\mathrm{r}})\big) \subseteq \exp\big(\mathcal{L}(Q_{\mathrm{r}})\big)$ and it is $\mathcal{V}$*-maximal* iff every rewriting Q'_{r} of Q w.r.t. $\mathcal{V}$ is such that $\mathcal{L}(Q'_{\mathrm{r}}) \subseteq \mathcal{L}(Q_{\mathrm{r}})$. Intuitively, for $\mathcal{V}$-maximality we compare the languages over $\mathcal{V}$ defined by the rewritings, while for $\mathcal{R}$-maximality we compare the corresponding expansions over $\mathcal{R}$ of such languages. Note that all of the $\mathcal{R}$-maximal (resp., $\mathcal{V}$-maximal) rewritings define the same language, and there exist $\mathcal{R}$-maximal rewritings which are not $\mathcal{V}$-maximal, as shown in the following example.

▶ **Example 24.** Let $\mathcal{R} = \{r\}$, $Q = r^*$ and $\mathcal{V} = \{v\}$ with $v^{\mathcal{R}} = r^*$. Then, we have that $Q_{\mathrm{r}} = v^*$ and $Q'_{\mathrm{r}} = v$ are both $\mathcal{R}$-maximal rewritings of Q w.r.t. $\mathcal{V}$, but the former is also $\mathcal{V}$-maximal while the latter is not.

As it turns out, $\mathcal{V}$-maximality is a sufficient condition for $\mathcal{R}$-maximality, i.e., every $\mathcal{V}$-maximal rewriting is also $\mathcal{R}$-maximal, hence we can search for a $\mathcal{V}$-maximal rewriting in order to find an $\mathcal{R}$-maximal one. This approach is suitable for (common) cases in which any $\mathcal{R}$-maximal rewriting would do, whereas it is not when one is interested in finding a specific $\mathcal{R}$-maximal rewriting (e.g., one maximising the use of some view symbol because it is less expensive), which might not indeed be $\mathcal{V}$-maximal.

We present a method that constructs a $\mathcal{V}$-maximal (hence also $\mathcal{R}$-maximal) rewriting of Q. Such a maximal rewriting always exists, even though it may be empty. The idea on which the proposed method is based consists in characterising, by means of an automaton, exactly those $\mathcal{V}$-words that do not belong to any of the languages defined by every possible rewriting of Q w.r.t. $\mathcal{V}$. Observe that a $\mathcal{V}$-word is such if its expansion contains an $\mathcal{R}$-word that is not in $\mathcal{L}(Q)$. Then, the complement of this automaton represents the maximal rewriting we are seeking, since it accepts exactly the $\mathcal{V}$-words whose expansions belong to $\mathcal{L}(Q)$.

▶ **Algorithm 3** (Compute maximal rewriting).

Input: *A regular expression Q over $\mathcal{R}$; a regular expression $v^{\mathcal{R}}$ over $\mathcal{R}$ for each $v \in \mathcal{V}$.*

```
1. Construct a deterministic automaton $A$ over $\mathcal{R}$ such that $\mathcal{L}(A) = \mathcal{L}(Q)$.
2. Define the automaton $A'$ over $\mathcal{V}$ such that:
   - $A'$ has the same set of states and the same initial state as $A$;
   - all states that are not final in $A$ are the final states of $A'$;
   - $A'$ has a transition from state $s_i$ to state $s_j$ labelled by $v$ iff there is a word in $\mathcal{L}(v^{\mathcal{R}})$
     that leads (through a sequence of transitions) from $s_i$ to $s_j$ in $A$.
3. Construct the complement $\overline{A'}$ of $A'$.[8]
```

Output: *The automaton $\overline{A'}$.*

The automaton A' built in the second step of the algorithm accepts only those $\mathcal{V}$-words leading from the initial state (which is the same as A's) to a state that is non-final for A (A's final states are non-final in A' and vice versa). The reason why the automaton A built in the first step is required to be deterministic is that we need to make sure that no $\mathcal{R}$-word in the expansion of each $\mathcal{V}$-word accepted by A' leads to a final state of A, thus belonging to $\mathcal{L}(Q)$.

▶ **Example 25.** Let $\mathcal{R} = \{r_1, r_2, r_3\}$ and $\mathcal{V} = \{v_1, v_2, v_3\}$, with the following definitions:

$$v_1{}^{\mathcal{R}} = r_1 \ ; \qquad v_2{}^{\mathcal{R}} = r_1 \cdot r_3{}^* \cdot r_2 \ ; \qquad v_3{}^{\mathcal{R}} = r_3 \ .$$

We want to rewrite the RE $Q = r_1 \cdot (r_2 \cdot r_1 + r_3)$ in terms of $\mathcal{V}$. The result of applying each step of Algorithm 3 is shown in Figure 2: first, we construct the deterministic automaton A of Figure 2a that accepts the language defined by Q; then, we build the automaton A' of Figure 2b that, for instance, has a v_2-labelled loop in s_0 because there exists a word (e.g., $r_1 \cdot r_2$) in the language defined by $v_2{}^{\mathcal{R}}$ leading from s_0 (through s_1) back to s_0 in A;[9] finally, since in this specific example A' is deterministic,[10] its complement $\overline{A'}$ is obtained by simply swapping final and non-final states of A' as in Figure 2c.

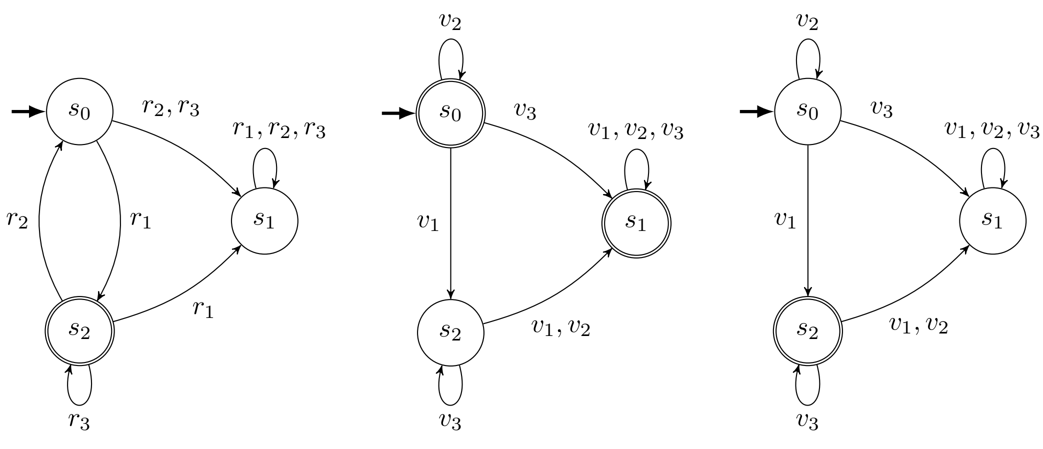
It can be proved (see [17] for details) that the automaton $\overline{A'}$ in the output of Algorithm 3 is indeed a $\mathcal{V}$-maximal rewriting of Q w.r.t. $\mathcal{V}$. This also shows that the language over $\mathcal{V}$ defined by the $\mathcal{V}$-maximal rewritings is in fact regular, although the form of the rewritings was not constrained a priori (the notion of rewriting was introduced as any expression defining a language over $\mathcal{V}$). The complexity of Algorithm 3 can be analysed as follows by considering the cost of each step:

1. Generating the deterministic automaton A from the regular expression Q is exponential.
2. Building A' is polynomial. In particular, it is required to check whether for each pair of states s_i and s_j there exists a word in the language defined by the RE associated with $v \in \mathcal{V}$ leading from s_i to s_j in A. This can be done by considering the automaton $A^{i,j}$,

[8] This is done by transforming A' into a deterministic automaton and swapping its final and non-final states. Obviously, the language $\mathcal{L}(\overline{A'})$ accepted by $\overline{A'}$ is the complement of the language $\mathcal{L}(A')$ accepted by A', that is, $\mathcal{L}(\overline{A'}) = \overline{\mathcal{L}(A')}$.

[9] In fact, in this particular example, **every** word in $\mathcal{L}(v_2{}^{\mathcal{R}})$ leads from s_0 back to s_0 in A.

[10] In general, the automaton A' constructed in the second step of Algorithm 3 is non-deterministic.



(a) A **(b)** A' **(c)** $\overline{A'}$

Figure 2 Construction of the rewriting of the regular expression $r_1 \cdot (r_2 \cdot r_1 + r_3)^*$ over $\mathcal{R} = \{r_1, r_2, r_3\}$ with respect to $\mathcal{V} = \{v_1, v_2, v_3\}$ where ${v_1}^{\mathcal{R}} = r_1$, ${v_2}^{\mathcal{R}} = r_1 \cdot {r_3}^* \cdot r_2$ and ${v_3}^{\mathcal{R}} = r_3$.

obtained from A by changing the initial state to s_i and the set of final states to $\{s_j\}$, and then checking for the non-emptiness of the product automaton between $A^{i,j}$ and an automaton for $\mathcal{L}(V^{\mathcal{R}})$.

3. Complementing A' (which is in general non-deterministic) is exponential.

Hence, generating the $\mathcal{V}$-maximal rewriting of a regular expression w.r.t. a set of regular expressions is in 2EXPTIME [17]. Deciding the existence of a non-empty rewriting can be done by first generating the maximal rewriting (2EXPTIME) and then checking for its non-emptiness (NLOGSPACE), which together give an EXPSPACE bound [17]. By means of a reduction from a suitable EXPSPACE-complete tiling problem, it has been shown that such a bound is tight (see [17] for the details of the reduction).

▶ **Theorem 26.** *The problem of verifying the existence of a non-empty rewriting of a regular expression w.r.t. a set of regular expressions is EXPSPACE-complete.*

Next, we present a method for checking whether a rewriting, in the form of the automaton $\overline{A'}$ obtained from Algorithm 3, is exact, that is, whether it captures the language defined by Q in its entirety.

▶ **Algorithm 4** (Verify exactness of rewriting).

Input: *The automatons A and $\overline{A'}$ built by Algorithm 3.*

1. *Construct an automaton B such that $\mathcal{L}(B) = \exp\big(\mathcal{L}(\overline{A'})\big)$ as follows:*

  a. *For each $v_i \in \mathcal{V}$, construct an automaton A_i such that $\mathcal{L}(A_i) = \mathcal{L}({v_i}^{\mathcal{R}})$. We assume w.l.o.g. that each A_i has a unique initial state with no incoming edges and a unique final state with no outgoing edges.*

  b. *B is obtained from $\overline{A'}$ by replacing each edge labelled by v_i with a new copy of A_i, identifying its initial state with the source of the edge and its final state with the target of the edge.*

2. *Check for the emptiness of $A \cap \overline{B}$.*

Output: *Yes, if $\mathcal{L}(A \cap \overline{B}) = \varnothing$; No, otherwise.*

It can be proved that the automaton $\overline{A'}$ is an exact rewriting of Q w.r.t. $\mathcal{V}$ iff $\mathcal{L}(A \cap \overline{B}) = \varnothing$. Thus, deciding the existence of an exact rewriting requires first the generation of $\overline{A'}$ (doubly

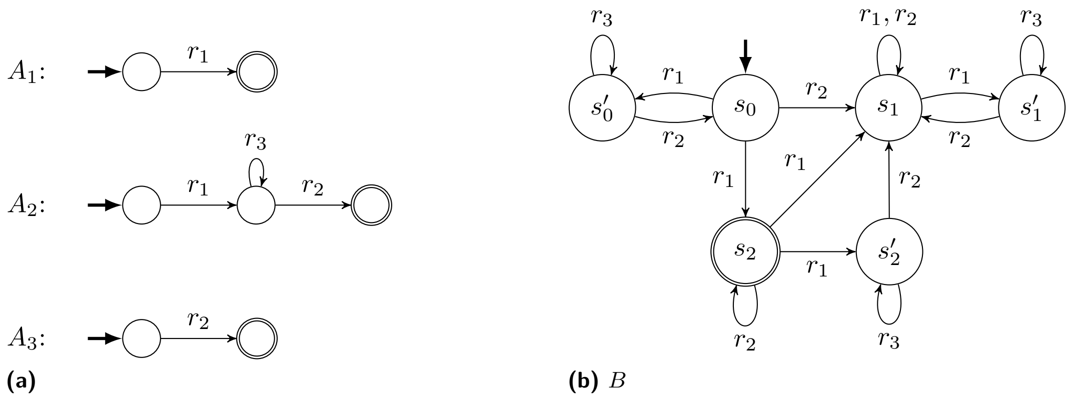


Figure 3 The construction of automaton B of Algorithm 4 in the setting of Example 25.

exponential), then the construction of B (polynomial) and its complementation (exponential), and finally checking whether $\mathcal{L}(A \cap \overline{B}) = \varnothing$ (NLOGSPACE). The explicit construction of $\overline{B}$ (and the further exponential blow-up it causes) can be fortunately avoided by building $\overline{B}$ "on-the-fly" as follows: Whenever the non-emptiness test for $A \cap \overline{B}$ requires to move from a state s_1 to a state s_2, the algorithm guesses s_2 and checks that it is directly connected to s_1; after guessing a suitable state, s_1 can be discarded; therefore, at each step, at most two states need to be kept in memory. By avoiding the generation of the whole $\overline{B}$, we obtain a 2EXPSPACE bound, while hardness can be shown with a reduction from a suitable tiling problem (see [17] for details), thus giving the following result.

▶ **Theorem 27.** *The problem of verifying the existence of an exact rewriting of a regular expression w.r.t. a set of regular expressions is 2EXPSPACE-complete.*

Referring back to Example 25, Figure 3 shows the construction of the automaton B, as in the first step of Algorithm 4. The automata A_1, A_2 and A_3 of Figure 3a accept the languages defined by the REs ${v_1}^{\mathcal{R}}$, ${v_2}^{\mathcal{R}}$ and ${v_3}^{\mathcal{R}}$, respectively. By substituting each v_i-labelled edge in $\overline{A'}$ (see Figure 2c) with a fresh copy of the corresponding A_i from Figure 3a, we obtain the automaton B depicted in Figure 3b. For instance, the loop labelled by v_2 in state s_0 of $\overline{A'}$ is replaced with a fresh copy of A_2, which requires the creation of a new state s'_0 in B and whose initial and final states are both identified with s_0. Similarly, replacing the v_2-labelled edge from s_2 to s_1 in $\overline{A'}$ requires the creation of a new state s'_2 in B, but this time the initial state of A_2 is identified with s_2 while the final one is identified with s_1. By applying the second step of Algorithm 4 it can be verified that the RE $Q_{\mathrm{r}} = {v_2}^* \cdot v_1 \cdot {v_3}^*$ represented by $\overline{A'}$ is indeed an exact rewriting of Q.

4.2 Answering, rewriting, and losslessness

Most of the work in the area of view-based query processing has focused on a setting based on conjunctive queries, in which it turns out that query answering and query rewriting coincide. Indeed, if the target queries are allowed to be written as UCQs, then the UCQ-maximal rewriting computes exactly the certain answers. For this reason, the relationship between answering and rewriting in view-based query processing is not always well understood. Calvanese et al. [19] provide a clean explanation of the distinction between the two notions in the context of semi-structured data, where the expressiveness of RPQs and 2RPQs allows to point out the subtle differences between answering and rewriting. Let us illustrate how

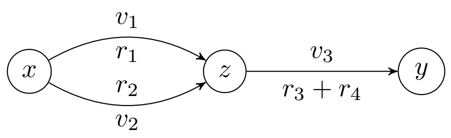


Figure 4 The structure of $\mathrm{cert}_{Q,\mathcal{V}}$ in Example 28.

things get more complicated when going beyond conjunctive queries by means of an example involving RPQs.

▶ **Example 28.** Let $\mathcal{R} = \{r_1, r_2, r_3, r_4\}$, $Q = r_1 \cdot r_3 + r_2 \cdot r_4$ and $\mathcal{V} = \{v_1, v_2, v_3\}$ with the following definitions:

$$v_1{}^{\mathcal{R}} = r_1 \ , \qquad v_2{}^{\mathcal{R}} = r_2 \ , \qquad v_3{}^{\mathcal{R}} = r_3 + r_4 \ .$$

By applying the techniques presented in the previous section, it can be checked that the RPQ-maximal rewriting of Q w.r.t. $\mathcal{V}$ is empty. On the other hand, the certain answers can be expressed as follows:

$$\mathrm{cert}_{Q,\mathcal{V}} = \big\{ \langle x, y \rangle \mid \exists z \; v_1(x,z) \wedge v_2(x,z) \wedge v_3(z,y) \big\} \ . \tag{11}$$

Observe that, as shown in Figure 4, (11) matches non-linear patterns in a database.

By characterising both answering and rewriting in terms of constraint satisfaction problems (see [19] for the technical details), Calvanese et al. show that the former is more precise than the latter. Indeed, while rewriting ignores that the same pair of objects might be connected by different edges, answering takes this into account, thus being able to recognise non-linear patterns as in the case of Example 28. Since it is defined directly in terms of the information content of the views, query answering is a more robust notion than query rewriting. In fact, whether a tuple is among the certain answers logically follows from the view extension. On the other hand, the motivation behind query rewriting is of a practical nature, that is, the need to access the view extensions by means of a specific query language.

A related and relevant issue in view-based query processing concerns *losslessness*, that is, the ability of being able to "completely" answer a query by relying only on the content of the views. Such an ability depends on the chosen approach to view-based query processing and can thus be considered from the query answering perspective, on the one hand, and from the query rewriting perspective, on the other. In the context of RPQs and 2RPQs, as we shall see, there is indeed a distinction between different notions of losslessness, while this is not the case for conjunctive queries, as answering and rewriting coincide.

A set of views $\mathcal{V}$ is *lossless* w.r.t. query Q iff for every database $\mathbf{D}$ we have that $Q(\mathbf{D}) = \mathrm{cert}_{Q,\mathcal{V}}\big(\mathcal{V}^{\mathcal{R}}(Q)\big)$. In other words, a set of views $\mathcal{V}$ is lossless when the certain answers over the $\mathcal{V}$-extension obtained from the view definitions always coincide with the answers to the query. This equivalence between the query and the certain answers determines whether the information content of a set of views is sufficient to answer completely a given query and constitutes the notion of losslessness w.r.t. answering.

In order to determine whether there is loss w.r.t. rewriting, we can compare rewritings to the original queries, on one side, and to the certain answers on the other, with the aim of checking for equivalence in either case. Depending on which comparison we are performing, we assess different aspects of losslessness w.r.t. query rewriting, which are represented by the two notions of *exactness* and *perfectness*. The former consists in the equivalence of a rewriting to the original query, modulo the view definitions; the latter is the equivalence to the certain answers. In other words, a rewriting is *exact* if it gives the same answers as the original query and *perfect* if it computes exactly the certain answers. In the case of conjunctive queries, where answering and rewriting coincide, maximal rewritings are always perfect and, as a consequence, also losslessness w.r.t. answering and losslessness w.r.t. rewriting coincide (that is, a set of views is lossless w.r.t. a query Q iff the maximal rewriting of Q is exact).

Let $\mathcal{V}$ be a set of 2RPQ views and Q a 2RPQ query, and denote by $Q_{\mathrm{r}}^{\max}$ the 2RPQ-maximal rewriting of Q with respect to $\mathcal{V}$. Then, using results from [18] and exploiting the fact that 2RPQs are monotone, we have that for every database $\mathbf{D}$ the following holds:

$$Q_{\mathrm{r}}^{\max}\big(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\big) \subseteq \mathrm{cert}_{Q,\mathcal{V}}\big(\mathcal{V}^{\mathcal{R}}(\mathbf{D})\big) \subseteq Q(\mathbf{D}) \ . \tag{12}$$

Observe that in (12) we evaluate the certain answers and the maximal rewriting over the $\mathcal{V}$-extension $\mathcal{V}^{\mathcal{R}}(\mathbf{D})$ obtained from the view definitions, instead of an arbitrary $\mathcal{V}$-extension that is sound w.r.t. $\mathbf{D}$. The reason is that, when considering $\mathcal{V}$-extensions that are strict subsets of $\mathbf{D}$, there might clearly be loss of information, but such a loss would be unrelated to the "quality" of the views, which is what we are interested in assessing. The richness of 2RPQs allows us to discern and appreciate the differences between the various notions of losslessness, which correspond to the cases in which some or all of the inclusions in (12) are actually equalities. Each case is discussed below.

- When $\mathcal{V}$ is lossless w.r.t. Q, we have equivalence between the query and the certain answers, therefore the rightmost inclusion in (12) is an equality. In this case, there is no loss of information caused by the fact that we are answering the query based on a set of views (but there might still be loss due to rewriting).
- When $Q_{\mathrm{r}}^{\max}$ is perfect, that is, equivalent to the certain answers, we have that the leftmost inclusion in (12) is an equality. In this case, we do not lose answering power by resorting to rewriting (but there might still be loss due to the fact that we are answering a query based on a set of views).
- When $Q_{\mathrm{r}}^{\max}$ is exact, that is, equivalent to the query, we have that both inclusions in (12) are actually equalities. Therefore, the maximal rewriting is not only exact but also perfect and, in addition, the views are lossless w.r.t. the query. This means that exactness of the maximal rewriting is the strongest notion, in that it combines together both losslessness of the views and perfectness of the rewriting.

We conclude this section by briefly reporting the main complexity results obtained so far concerning losslessness, exactness and perfectness in the case of RPQs [17, 16] and 2RPQs [19, 20]. Let $\mathcal{V}$ be a set of RPQ views, Q an RPQ query and $Q_{\mathrm{r}}^{\max}$ the RPQ-maximal rewriting of Q with respect to $\mathcal{V}$. Then, the following hold:

- Checking whether $\mathcal{V}$ is lossless w.r.t. Q is EXPSPACE-complete in the size of Q and PSPACE-complete in the size of the view definitions $\mathcal{V}^{\mathcal{R}}$ [16].
- Verifying the existence of an exact rewriting of Q w.r.t. $\mathcal{V}$ is 2EXPSPACE-complete [17] (see Section 4.1).

Let $\mathcal{V}$ be a set of 2RPQ views, Q a 2RPQ query and $Q_{\mathrm{r}}^{\max}$ the 2RPQ-maximal rewriting of Q with respect to $\mathcal{V}$. Then, the following hold:

- Checking whether $\mathcal{V}$ is lossless w.r.t. Q can be done in EXPSPACE in the size of Q and the view definitions $\mathcal{V}^{\mathcal{R}}$ [19].
- Verifying the existence of an exact rewriting of Q w.r.t. $\mathcal{V}$ is 2EXPSPACE-complete [20].[11]
- Checking whether $Q_{\mathrm{r}}^{\max}$ is perfect can be done in N2EXPTIME in the size of Q and in NEXPTIME in the size of $\mathcal{V}^{\mathcal{R}}$ [19].

[11] In [20], the same complexity results reported in [17] for RPQs are indeed shown to hold also in the case of regular path queries with inverse.

5 Query processing in other data integration scenarios

We would like to conclude the chapter by briefly discussing query processing in the emerging areas of XML data integration and ontology-based data integration.

5.1 Data integration with XML

An increasing number of data integration applications uses XML for describing the global schema, which allows to hide proprietary source schemas that data owners do not want to disclose, while at the same time adhering to a newly-established standardised interface without the need of migrating existing data. Indeed, much of the work in data integration with XML is motivated by and focuses on the problem of publishing portions of relational and/or XML data stored in proprietary sources through a global XML schema against which user's queries are formulated. Compared to the case discussed in Section 4 of query processing in semistructured data, the translation of XML queries (expressed in XQuery) poses technical difficulties that do not arise with semistructured queries like RPQs and 2RPQs.

A methodology for integrating heterogeneous data sources under an XML global schema with LAV mapping is described in [45], where queries against the global schema, expressed in the XML query language XQuery, are translated into SQL queries over the local data sources. The approach followed in [45] allows for mixed relational and XML (in fact, any DOM-compliant) data sources and consists of the following three phases:

Normalisation The initial user's query (expressed in XQuery) is brought, through equivalence-preserving transformations, to a syntactical form that, whenever possible, can be directly translated to SQL. Indeed, normalisation identifies the features of XQuery that do not have SQL equivalents and filters out the queries which make use of them.

Translation The normalised query is translated into an SQL query over a generic, virtual, relational schema serving as an intermediate layer independent of the actual relationship between the XML global schema and the data sources. Indeed, the methodology described in [45] is implemented in the data integration system Agora [46], where relational and tree-structured data sources are defined by views over the global XML schema by means of such an intermediate schema that closely models the generic structure of an XML document.

Rewriting The translated SQL query on the generic schema is rewritten into a SQL query on the real data sources, by means of query rewriting algorithm [42] searching for maximally-contained rewritings. The authors of [45] argue that, even though in different scenarios a rewriting algorithm searching for exact query rewritings can be employed, as is the case in [46], in large-scale data integration applications, where there is no guarantee that all qualifying data is available, maximally-contained rather than exact rewritings are more appropriate.

The problem of finding an exact rewriting (if one exists) of an XQuery query against a global (public) XML schema into one or more queries over the source schema is studied in [28, 30] in a general setting with mixed (XML and relational) storage for the local (proprietary) data under the GLAV approach and in the presence of integrity constraints on both the global and the source schemas. The class of queries considered in [28] is a fragment of XQuery, consisting of so-called *XBind* queries whose general form is reminiscent of conjunctive queries; the constraints on the relational part are *disjunctive embedded dependencies* (DEDs) [2, 27] and on the XML part are *XML Integrity Constraints* (XIC) [31] whose expressive power captures a considerable part of XML Schema [9] including keys and "keyrefs" and more.

The approach followed in [28] consists in "compiling" an instance of the XML query rewriting problem into a relational one, and then solving the latter by using the Chase & Backchase (C&B) algorithm [25, 27]. The C&B enumerates the exact rewritings of a query that are "minimal" w.r.t. a set of constraints, in the sense that no atom can be removed from the rewriting without compromising equivalence to the original query (under the given constraints). Whenever the Chase is guaranteed to terminate, which is the case for sets of dependencies with *stratified witness* [28] (a.k.a. *weakly acyclic* sets of dependencies [34]), the C&B is complete, in the sense that outputs all (up to equivalence) and only the minimal rewritings of the input query under the given constraints.

The solutions devised in [28] are implemented in the MARS system [29], which can handle all of the cases handled by other LAV-based integration systems, such as Agora [46] and STORED [22] (for XML publishing), and Information Manifold [43] (for purely relational integration). Moreover, for what concerns XML publishing in the pure GAV approach, and when the storage schema is purely relational, MARS also subsumes the expressive power of the systems XPeranto [52] and SilkRoute [35].

Even though the settings of [28] and [45] differ in many aspects, as the former is GLAV-based, deals with exact rewritings and constraints are allowed on both global and source schemas, whereas the latter is LAV-based, deals with maximally-contained rewritings and no constraints are allowed, the two approaches are quite similar, in that they both make use of an intermediate relational schema, onto which the XML query rewriting problem is translated and then solved by means of relational query rewriting techniques.

We close our brief discussion of XML data integration by mentioning the work [55], where the authors address the problem of obtaining maximally-contained rewritings of XQuery queries in the presence of constraints on the XML global schema with LAV mapping, by devising a complete query rewriting algorithm that, differently from [45] and [28], operates directly on nested structures. The class of queries considered in [55] is a fragment of XQuery that includes nested subqueries; the constraints on the global schema are *nested equality-generating dependencies* (NEGDs), which include functional dependencies in relational or nested schemas, XML Schema key constraints, and more general constraints stating that certain tuples/elements must satisfy certain equalities.

Although [45] and [55] both use the LAV approach and deal with maximally-contained rewritings, the former does not allow for constraints whereas the latter takes into account constraints on the global schema. However, due to the translation to the intermediate generic relational schema, views and queries in [45] can be quite complex and hard to understand by humans, whereas the techniques in [55] operate directly at the XML level, thus resulting more natural and user-friendly.

5.2 Ontology-based data integration

The use of an ontology as the global schema in a data integration system, best known in the literature as ontology-based data access (OBDA), has the benefit of providing a semantically rich conceptual view of the information gathered by the system, which users can more easily understand. However, using an ontology to mediate the access to data sources amounts to data integration under integrity constraints, which must be fully considered during query answering, as they have a deep impact on how certain answers are computed [11]. Indeed, when the global schema is expressed in terms of even a very simple conceptual data model, the problem of incomplete information implicitly arises also in the GAV approach, making query processing (which without integrity constraints reduces to query unfolding) difficult [10].

When the global schema is an ontology expressed in the description logic ALCQI, which fully captures class-based representation formalisms, query answering in data integration is decidable [14]. However, the high computational data complexity makes the use of a such an expressive description logic infeasible in practice when dealing with large amounts of data, therefore the authors of [14] propose the adoption of DL-Lite, a specifically tailored restriction of ALCQI that ensures tractability of query answering in data integration while keeping enough expressive power. The "lightweight" description logic DL-Lite and its variants constitute a family of tractable description logics that are used in several applications, most notably the OBDA system MASTRO [15], providing access to heterogeneous relational data sources through an integrated ontology specified in a logic of the DL-Lite family.

Surveys on ontology-based approaches to semantic data integration present in the literature [54, 50, 7] compare several OBDA systems w.r.t. their reusability, changeability, and scalability. Recent systems, not considered in the above surveys, are OntoGrate [32] and the previously mentioned MASTRO [15], for relational data sources, and SOBA [8], for extracting and integrating information from heterogeneous resources including plain text, tables and image captions. We also cite MOMIS [6, 5] for the integration of semistructured data sources.

References

1 Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of PODS '98*, pages 254–263, 1998.

2 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

3 Foto Afrati. Rewriting conjunctive queries determined by views. In Ludek Kucera and Antonín Kucera, editors, *Mathematical Foundations of Computer Science*, volume 4708 of *Lecture Notes in Computer Science*, pages 78–89. Springer Berlin / Heidelberg, 2007.

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

5 Sonia Bergamaschi, Silvana Castano, and Maurizio Vincini. Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1):54–59, March 1999.

6 Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 36(3):215 – 249, 2001.

7 Agustina Buccella, Alejandra Cechich, and Rodríguez. *Encyclopedia of Database Technologies and Applications*, chapter Ontology-Based Data Integration, pages 450–456. Idea Group Reference, 2006.

8 Paul Buitelaar, Philipp Cimiano, Anette Frank, Matthias Hartung, and Stefania Racioppa. Ontology-based information extraction and integration from heterogeneous data sources. *International Journal of Human-Computer Studies*, 66(11):759–788, 2008.

9 Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, and Wang-Chiew Tan. Keys for XML. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 201–210. ACM, 2001.

10 Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Accessing data integration systems through conceptual schemas. In Hideko S.K̃unii, Sushil Jajodia, and Arne Sølvberg, editors, *Conceptual Modeling — ER 2001*, volume 2224 of *Lecture Notes in Computer Science*, pages 270–284. Springer Berlin / Heidelberg, 2001.

11 Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Data integration under integrity constraints. *Inf. Syst.*, 29(2):147–163, 2004.

12 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS '03*, pages 260–271. ACM, 2003.

13 Andrea Calì, Domenico Lembo, and Riccardo Rosati. Query rewriting and answering under constraints in data integration systems. In *IJCAI*, pages 16–21, 2003.

14 Diego Calvanese and Giuseppe De Giacomo. Data integration: a logic-based perspective. *AI Magazine*, 26(1):59–70, March 2005.

15 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1):43–53, jan 2011.

16 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless regular views. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pages 247–258, New York, NY, USA, 2002. ACM.

17 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443–465, 2002.

18 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query containment. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '03, pages 56–67, New York, NY, USA, 2003. ACM.

19 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoretical Computer Science*, 371(3):169–182, 2007.

20 Diego Calvanese, Moshe Y. Vardi, Giuseppe de Giacomo, and Maurizio Lenzerini. View-based query processing for regular path queries with inverse. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '00, pages 58–66, New York, NY, USA, 2000. ACM.

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

22 Alin Deutsch, Mary Fernandez, and Dan Suciu. Storing semistructured data with STORED. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD '99, pages 431–442. ACM, 1999.

23 Alin Deutsch, Bertram Ludäscher, and Alan Nash. Rewriting queries using views with access patterns under integrity constraints. *Theoretical Computer Science*, 371(3):200–226, 2007.

24 Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

25 Alin Deutsch, Lucian Popa, and Val Tannen. Physical data independence, constraints, and optimization with universal plans. In *VLDB*, pages 459–470, 1999.

26 Alin Deutsch, Lucian Popa, and Val Tannen. Query reformulation with constraints. *SIGMOD Record*, 35(1):65–73, 2006.

27 Alin Deutsch and Val Tannen. Optimization properties for classes of conjunctive regular path queries. In *Database Programming Languages*, pages 21–39. Springer, 2002.

28 Alin Deutsch and Val Tannen. Reformulation of XML queries and constraints. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *Database Theory — ICDT 2003*, volume 2572 of *Lecture Notes in Computer Science*, pages 225–241. Springer Berlin / Heidelberg, 2002.

29 Alin Deutsch and Val Tannen. Mars: a system for publishing xml from mixed and redundant storage. In *Proceedings of the 29th international conference on Very large data bases*, volume 29 of *VLDB '03*, pages 201–212. VLDB Endowment, 2003.

30 Alin Deutsch and Val Tannen. XML queries and constraints, containment and reformulation. *Theoretical Computer Science*, 336(1):57–87, 2005.

31 Aline Deutsch and Val Tannen. Containment and integrity constraints for XPath. In *Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB2001)*, volume 45 of *CEUR Workshop Proceedings*. ceur-ws.org, 2001.

32 Dejing Dou, Han Qin, and Paea Lependu. OntoGrate: Towards automatic integration for relational databases and the semantic web through an ontology-based framework. *International Journal of Semantic Computing*, 04(01):123–151, 2010.

33 Oliver M. Duschka, Michael R. Genesereth, and Alon Y. Levy. Recursive query plans for data integration. *Journal of Logic Programming*, 43(1):49–73, 2000.

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

35 Mary Fernández, Yana Kadiyska, Dan Suciu, Atsuyuki Morishima, and Wang-Chiew Tan. SilkRoute: A framework for publishing relational data in XML. *ACM Trans. Database Syst.*, 27(4):438–493, December 2002.

36 Georg Gottlob, Nicola Leone, and Francesco Scarcello. On tractable queries and constraints. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, DEXA '99, pages 1–15. Springer-Verlag, 1999.

37 Gösta Grahne. Incomplete information. In *Encyclopedia of Database Systems*, pages 1405–1410. Springer US, 2009.

38 Gösta Grahne and Alberto O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT*, pages 332–347, 1999.

39 Tomasz Imieliński and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

40 Dirk Leinders, Maarten Marx, Jerzy Tyszkiewicz, and Jan Van den Bussche. The semijoin algebra and the guarded fragment. *Journal of Logic, Language and Information*, 14:331–343, 2005.

41 Maurizio Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

42 Alon Y. Levy, Alberto O. Mendelzon, and Yehoshua Sagiv. Answering queries using views. In *Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '95, pages 95–104, New York, NY, USA, 1995. ACM.

43 Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996.

44 David Maier, Alberto O. Mendelzon, and Yehoshua Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

45 Ioana Manolescu, Daniela Florescu, and Donald Kossmann. Answering XML queries on heterogeneous data sources. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 241–250, 2001.

46 Ioana Manolescu, Daniela Florescu, Donald Kossmann, Florian Xhumari, and Dan Olteanu. Agora: Living with XML and relational. In *Proceedings of the International Conference on Very Large Data Bases*, pages 623–626, 2000.

47 Maarten Marx. Queries determined by views: Pack your views. In *Proceedings of PODS '07*, pages 23–30, 2007.

48 Prasenjit Mitra. An algorithm for answering queries efficiently using views. In *ADC*, pages 99–106, 2001.

49 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3):1–41, 2010.

50 Natalya F. Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65, 2004.

51 Rachel Pottinger and Alon Halevy. MiniCon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2–3):182–198, 2001.

52 Jayavel Shanmugasundaram, Jerry Kiernan, Eugene J. Shekita, Catalina Fan, and John Funderburk. Querying XML views of relational data. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 261–270. Morgan Kaufmann Publishers Inc., 2001.

53 Oded Shmueli. Equivalence of datalog queries is undecidable. *J. Log. Program.*, 15(3):231–241, 1993.

54 H. Wache, T. Voegele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information – A survey of existing approaches. In *Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing*, volume 47 of *CEUR Workshop Proceedings*, pages 108–117. ceur-ws.org, 2001.

55 Cong Yu and Lucian Popa. Constraint-based XML query rewriting for data integration. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pages 371–382. ACM, 2004.

Semantics for Non-Monotone Queries in Data Exchange and Data Integration

André Hernich

Humboldt University Berlin
Germany
hernich@informatik.hu-berlin.de

Abstract

A fundamental question in data exchange and data integration is how to answer queries that are posed against the target schema, or the global schema, respectively. While the certain answers semantics has proved to be adequate for answering *monotone queries*, the question concerning an appropriate semantics for *non-monotone queries* turned out to be more difficult. This article surveys approaches and semantics for answering non-monotone queries in data exchange and data integration.

1998 ACM Subject Classification H.2.5 Heterogeneous Databases, H.2.4 Systems, H.2.8 Database Applications

Keywords and phrases certain answers, closed world assumption, deductive databases, universal solutions, core

Digital Object Identifier 10.4230/DFU.Vol5.10452.161

1 Introduction

Query answering is a fundamental task both in data exchange and data integration. Indeed, the goal of *data integration* is to combine different sources of data and to provide a unified view through which these sources can be queried [29]. The data often resides at the sources while the view is *virtual* (i.e., not materialized). Hence, if a user queries the view, the query has to be answered using the source data, for example, by evaluating suitable queries on the sources and combining their results, or by materializing the relevant part of the view that is needed to answer the query. *Data exchange* is similar to data integration insofar as its goal is to translate databases over a source schema into databases over a target schema [11, 27, 6, 4], whereby providing a view (over the target schema) on the source database. However, unlike in data integration, the view is *materialized* and queries have to be answered directly on that view. In fact, in data exchange it is generally assumed that the source database is not available at the time the target database is queried [11].

In both areas, the basic approach for modeling the relationship between source and target is based on *schema mappings* [29, 27]. Schema mappings describe target databases (over a *target schema*) in terms of source databases (over a *source schema*) by means of high-level declarative assertions (typically expressed in a suitable fragment of first-order logic or even second-order logic). A *solution* for a source database is a target database that makes all assertions hold true. Usually schema mappings are *underspecified*, which means that source databases may have more than one solution. Hence, it is not obvious at all how to answer queries that are posed against the target schema of a schema mapping.

The following approach for answering queries is common in such a setting [29, 27, 6, 4]. Instead of answering a query Q on a single solution, the set of all tuples that are answers to Q on *all* solutions is returned. This set is called the set of the *certain answers* to Q on the given source instance and schema mapping. Informally, it contains all tuples that are *certain* to be answers to Q no matter on which solution Q is evaluated. The certain answers semantics has turned out to be adequate for answering *monotone queries* like unions of conjunctive queries with inequalities in the sense that it yields the best result obtainable from the information in the source database and the schema mapping. Although the definition of the certain answers involves a potentially infinite set of solutions, in many practical settings it is possible to compute them in polynomial time (in data complexity, i.e., for fixed schema mappings and queries) from a single solution called *universal solution* [11, 5], or by evaluating a suitably rewritten query over the source schema (cf., [29] and Chapter 5 of this book).

Researchers soon realized [11, 3, 31] that for *non-monotone* queries, the certain answers semantics may yield results that intuitively seem to be not accurate. The following example illustrates the basic problem.

▶ **Example 1.** Suppose we just want to *copy* source databases to target databases. For instance, assume that source databases contain a single binary relation E and the target database is going to be a database containing a single binary relation E'. Then the schema mapping M describing the translation from source to target could be specified by the *tuple-generating dependency*

$$\theta \ := \ \forall x, y\, \big(E(x, y) \to E'(x, y)\big).$$

Informally, θ says that all tuples in E have to be in E'. Hence, the set of solutions for a source database S consists of all target databases whose relation E' contains all tuples in the relation E of S.

On the other hand, since schema mappings describe translations from source to target, it seems to be natural to expect that the result of translating a source database S according to M is the copy S' of S over the new schema $\{E'\}$. In particular, it seems to be natural to expect that a query posed against the target schema yields the same result as the same query evaluated on S'. However, this is not the case if we answer queries by the set of the certain answers to the query: If the source database S is such that E contains only the tuple (c, d), where c, d are distinct constants, then the expected set of answers to

$$Q(x, y) \ := \ \forall z\, \big(E'(x, z) \to z = y\big)$$

would be $\{(c, d)\}$, yet the set of the certain answers to Q on S and M is empty (since the instance whose relation E' consists of the tuples (c, d) and (c, e) is a solution for S). ◀

As indicated by the example, the problem is really a matter of the semantics of schema mappings. Which target databases should constitute the set of solutions for a given source database? For the schema mapping M in the example, we argued that only the copy of a source database S should be a solution for S under M. To enforce this, we could have used a constraint like $\forall x, y\, \big(E(x, y) \leftrightarrow E'(x, y)\big)$ stating that a tuple is in E' precisely if it belongs to E. Then the set of the certain answers would be as desired. However, this approach – of using constraints that are not expressible by standard constraints like *tuple-generating dependencies* or *equality-generating dependencies* considered in the literature – seems to have received almost no attention.[1]

The approach pursued in the literature is to use custom-made semantics of query answering [13, 22, 32, 2, 21]. Under each of these semantics, a query Q is answered by the set of the

[1] An exception is [32], where an extension of tgds, *annotated tgds*, is considered whose semantics cannot be captured by any set of tgds and egds.

certain answers to Q with respect to a suitably restricted set $\mathcal{S}$ of solutions (i.e., by the set of all tuples that are answers to Q on all solutions from $\mathcal{S}$). Except for the semantics in [13], different forms of *non-monotonic reasoning*, specifically, variants of the *closed world assumption* [36], are implemented to arrive at the corresponding set of solutions. Here, the basic idea is to consider target databases as solutions only if they can be *derived* in a certain way from the source database and the schema mapping. Apart from trying to remedy the shortcomings of the certain answers semantics when it comes to answering non-monotone queries, I think that non-monotonic reasoning in data exchange and data integration is appealing in its own right. In principle, it allows for more compact specifications of schema mappings, since only the data that is actually moved from the source to the target has to be specified, without saying what should not be in the target database.

This chapter is intended to give an overview on the different semantics for answering non-monotone queries, and the complexity of query answering under those semantics.

The remaining part of this chapter is organized as follows. In Section 2 we introduce basic notions and notation used throughout this chapter, and in Section 3 we recall the certain answers semantics and its behavior on non-monotone queries. Section 4 surveys the different semantics that have been proposed in the literature for answering non-monotone queries. Finally, Section 5 compiles what is known about the complexity of answering non-monotone queries under those semantics.

2 Basics

Below, we recall standard notions from database theory and data exchange used in the rest of this chapter. For a detailed account of these, see, e.g., [1, 4].

We let $[n]$ be the set of all integers m with $1 \leq m \leq n$. Mappings $f\colon A \to B$ are extended to tuples $\bar{a} = (a_1, \dots, a_k)$ over A via $f(\bar{a}) := (f(a_1), \dots, f(a_k))$, and to relations $R \subseteq A^k$ via $f(R) := \{f(\bar{a}) \mid \bar{a} \in R\}$.

2.1 Databases

A *schema* is a finite set σ of relation symbols, where each $R \in \sigma$ has a fixed arity $\mathrm{ar}(R) \geq 1$. A *σ-instance* I assigns to each $R \in \sigma$ a finite relation R^I of arity $\mathrm{ar}(R)$. The *active domain of* I, that is, the set of all values that occur in I, is denoted by $\mathrm{dom}(I)$. As usual in data exchange, we assume that $\mathrm{dom}(I) \subseteq \mathit{Dom}$, where Dom is the union of two fixed disjoint infinite sets – the set Const of all *constants*, and the set Null of all *(labeled) nulls*. Constants are denoted by letters c, d, e and variants like c', c_1; different letters denote mutually distinct constants. Nulls serve as placeholders, or variables, for unknown constants; we will denote them by $\perp$ and variants like $\perp', \perp_1$. Instances without nulls are called *ground*. Let $\mathrm{const}(I) := \mathrm{dom}(I) \cap \mathit{Const}$ and $\mathrm{nulls}(I) := \mathrm{dom}(I) \cap \mathit{Null}$.

It will often be convenient to view instances as sets of atoms, where an *atom* is an expression of the form $R(\bar{a})$ with R a relation symbol and $\bar{a} \in \mathit{Dom}^{\mathrm{ar}(R)}$. Thus, we identify σ-instances I with the set $\{R(\bar{a}) \mid R \in \sigma, \bar{a} \in R^I\}$. This enables us to apply set theoretic notation to instances. For example, we may write $R(\bar{a}) \in I$ instead of "$R \in \sigma$ and $\bar{a} \in R^I$." Furthermore, we may write $I \subseteq J$ to indicate that all atoms of I are contained in J, or $I \cup J$ for the instance consisting of all the atoms of I and all the atoms of J.

Let I and J be instances. A *homomorphism* from I to J is a mapping $h\colon \mathrm{dom}(I) \to \mathrm{dom}(J)$ such that for all constants $c \in \mathrm{dom}(I)$ we have $h(c) = c$, and for all $R(a_1, \dots, a_k) \in I$ we have $R(h(a_1), \dots, h(a_k)) \in J$. We call J a *homomorphic image* of I if there is a homomorphism h from I to J such that $J = h(I)$, where we define

$$h(I) := \{R(h(a_1), \dots, h(a_k)) \mid R(a_1, \dots, a_k) \in I\}.$$

Furthermore, we call I and J *homomorphically equivalent* if there is a homomorphism from I to J and a homomorphism from J to I. An *isomorphism* from I to J is a bijective homomorphism h from I to J such that h^{-1} is a homomorphism from J to I. If there is an isomorphism from I to J, we call I and J *isomorphic.* We say that J is a *core* of I if $J \subseteq I$ and there is a homomorphism from I to J, but no homomorphism from I to a proper subinstance of J. As shown in [19] (see also [13]), every instance has a core, and cores of homomorphically equivalent instances are isomorphic.

2.2 Queries and Constraints

The reader is assumed to be familiar with first-order logic (FO). Atomic FO-formulas over a schema σ are formulas of the form $R(u_1, \dots, u_{\mathrm{ar}(R)})$ or $u_1 = u_2$, where $R \in \sigma$ and each u_i is a variable or an element of *Const.* Formulas of the first form are called *relation atoms* (over σ). FO-formulas over σ are built from atomic FO-formulas over σ in the usual way using negation, conjunction, disjunction, implication, existential quantification, and universal quantification. We write $\varphi(x_1, \dots, x_k)$ to indicate that φ is a formula whose free variables are precisely $x_1, \dots, x_k$; if φ is a sentence, we omit the parentheses.

Let $\mathrm{dom}(\varphi)$ be the set of all constants in φ. An assignment for φ in an instance I is a mapping α from the free variables of φ to $\mathrm{dom}(I) \cup \mathrm{dom}(\varphi)$, which we extend to *Const* via $\alpha(c) := c$ for all $c \in$ *Const.* We write $(I, \alpha) \models \varphi$ to indicate that φ is satisfied in I under α. The relation $\models$ is defined as usual, the only difference being that constants in φ are interpreted by themselves, and quantifiers range over $\mathrm{dom}(I) \cup \mathrm{dom}(\varphi)$. That is, we apply the *active domain semantics.* For example, we have $(I, \alpha) \models R(u_1, \dots, u_{\mathrm{ar}(R)})$ precisely if $(\alpha(u_1), \dots, \alpha(u_{\mathrm{ar}(R)})) \in R^I$; $(I, \alpha) \models u_1 = u_2$ precisely if $\alpha(u_1) = \alpha(u_2)$; and $(I, \alpha) \models \exists x\, \varphi$ precisely if there is an $a \in \mathrm{dom}(I) \cup \mathrm{dom}(\varphi)$ with $(I, \alpha[a/x]) \models \varphi$, where $\alpha[a/x]$ is the assignment defined like α, except that x is mapped to a. For an FO-formula $\varphi(x_1, \dots, x_k)$ and a tuple $\bar{a} = (a_1, \dots, a_k) \in (\mathrm{dom}(I) \cup \mathrm{dom}(\varphi))^k$, we write $I \models \varphi(\bar{a})$ instead of $(I, \alpha) \models \varphi$, where $\alpha(x_i) = a_i$ for each $i \in [k]$.

An FO-*query* over σ is an FO-formula φ over σ together with a tuple $\bar{x} = (x_1, \dots, x_k)$ containing all the free variables in φ; we denote such queries by $\varphi(\bar{x})$. The *result* of $\varphi(\bar{x})$ on I is the set $\varphi(I) := \{\bar{a} \in (\mathrm{dom}(I) \cup \mathrm{dom}(\varphi))^k \mid I \models \varphi(\bar{a})\}$. We will often tacitly use the fact that for every FO-query Q over σ, there is a polynomial time algorithm that takes a σ-instance I as input and outputs $Q(I)$.

A *conjunctive query (CQ)* is an FO-query of the form $\varphi(\bar{x}) = \exists \bar{y}\, \psi$, where ψ is a conjunction of relation atoms. If ψ is a conjunction of relation atoms and inequalities $\neg u = v$, we call φ a *CQ with inequalities*, and if ψ is a conjunction of relation atoms and negated relation atoms, we call φ a *CQ with negation.* A *union of conjunctive queries (UCQ)* is a disjunction of CQs. A *UCQ with inequalities* is a disjunction of CQs with inequalities.

When we refer to the *atoms* of $\varphi(\bar{a})$ for some FO-formula $\varphi(\bar{x}) = R_1(\bar{y}_1) \wedge \dots \wedge R_k(\bar{y}_k)$ and an assignment $\bar{a}$ for $\bar{x}$, we mean the atoms $R_i(\bar{b}_i)$, where $\bar{b}_i$ is obtained from $\bar{y}_i$ by replacing each variable in $\bar{y}_i$ with the corresponding value assigned to that variable by $\bar{a}$.

2.3 Schema Mappings

The following definitions are standard in data exchange (cf., e.g., [11, 13, 27, 6, 4]). A *schema mapping* $M = (\sigma, \tau, \Sigma)$ consists of disjoint schemas σ and τ, called *source schema* and *target schema*, and a finite set Σ of assertions, where we distinguish between *source-to-target tuple-generating dependencies (st-tgds)*, *target tuple-generating dependencies (t-tgds)*, and *equality-generating dependencies (egds)*. *St-tgds* are FO-sentences of the form $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z}))$, where φ is a conjunction of relation atoms over σ, ψ is a conjunction of relation atoms over τ, and φ and ψ contain no constants. *T-tgds* are defined similarly; they differ from st-tgds only in that φ, like ψ, is a conjunction of relation atoms over τ. *Egds* are FO-formulas of the form $\forall\bar{x}(\varphi(\bar{x}) \to y = z)$, where φ is a conjunction of relation atoms over τ, y and z occur in $\bar{x}$, and φ contains no constants.

A *source instance* S for M is a ground σ-instance. A *solution for* S *under* M is a τ-instance T such that $S \cup T$ satisfies all the tgds and egds in Σ.[2] Note that, unlike solutions, source instances are not allowed to contain nulls, and that a source instance may have no solution or more than one solution.

Concerning the question as to which solution should be materialized for data exchange, [11] proposes *universal solutions*, and makes a good case for materializing such solutions. A *universal solution* for S under M is a solution T for S under M such that for every solution T' for S under M there is a homomorphism from T to T'. A source instance for M might not have a universal solution, even if it has solutions. However, if M is specified by st-tgds, then every source instance S has a universal solution under M. For example, such a universal solution can be constructed from an initially empty instance over M's target schema by adding, for each st-tgd $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z}))$ in M and each pair $\bar{a},\bar{a}'$ of tuples with $S \models \varphi(\bar{a},\bar{a}')$ the atoms of $\psi(\bar{a},\bar{b})$, where $\bar{b}$ is a tuple of pairwise distinct fresh nulls. The resulting universal solution is unique up to isomorphism. We call it the *canonical solution* for S under M, and denote it by $\mathrm{CanSol}(M,S)$.

Particular important universal solutions are *core solutions*, which can be thought of as smallest universal solutions. If M is specified by st-tgds, then a *core solution* for S under M is defined as a core of $\mathrm{CanSol}(M,S)$. Since every two cores of $\mathrm{CanSol}(M,S)$ are isomorphic, there is a unique core solution for S under M up to isomorphism. Hence, we may speak of *the* core solution, denoted by $\mathrm{Core}(M,S)$. It is easy to verify that $\mathrm{Core}(M,S)$ is a solution for S under M. We will not need core solutions for more general schema mappings; see [13] for their definition and properties.

▶ **Example 2.** Consider the schema mapping $M = (\sigma,\tau,\Sigma)$ with σ consisting of a binary relation symbol *Book*, τ consisting of binary relation symbols *Author* and *BookInfo*, and Σ containing the st-tgd

$$\forall x\forall y\big(\mathit{Book}(x,y) \to \exists z(\mathit{Author}(y,z) \wedge \mathit{BookInfo}(z,x))\big).$$

Furthermore, consider the source instance

$$\begin{aligned} S \;:=\; \{&\mathit{Book}(\text{Comput. Compl.}, \text{S. Arora}), \mathit{Book}(\text{Comput. Compl.}, \text{B. Barak}),\\ &\mathit{Book}(\text{Model Theory}, \text{W. Hodges})\}, \end{aligned}$$

which stores tuples of the form (book title, author) in Book^S. Then,

$$\begin{aligned} T \;:=\; \{&\mathit{Author}(\text{S. Arora}, \bot_1), \mathit{BookInfo}(\bot_1, \text{Comput. Compl.}),\\ &\mathit{Author}(\text{B. Barak}, \bot_2), \mathit{BookInfo}(\bot_2, \text{Comput. Compl.}),\\ &\mathit{Author}(\text{W. Hodges}, \bot_3), \mathit{BookInfo}(\bot_3, \text{Model Theory})\,\} \end{aligned}$$

[2] A word of caution: In data exchange, solutions are usually finite, as introduced here, whereas in data integration, solutions may also be infinite. For simplicity, we consider only finite solutions.

is a universal solution for S under M. It is both the canonical solution and the core solution for S under M. Other universal solutions can be obtained from T by adding arbitrary tuples with nulls to T. Note that the instance T' obtained from T by identifying $\bot_1$ and $\bot_2$ is not a universal solution, since there is no homomorphism from T' to T. On the other hand, if we would add the egd $\forall x_1 \forall x_2 \forall y \big(\mathit{BookInfo}(x_1, y) \wedge \mathit{BookInfo}(x_2, y) \rightarrow x_1 = x_2 \big)$ to Σ, then T' would be the core solution for S under M. ◀

For later reference, we state:

▶ **Theorem 3** ([11, 13]). *Let M be a schema mapping defined by st-tgds. Then there are polynomial-time algorithms that take a source instance S for M as input and compute* $\mathrm{CanSol}(M, S)$ *and* $\mathrm{Core}(M, S)$*, respectively.*

3 The Certain Answers Semantics and Non-Monotone Queries

The basic approach for answering a query Q over the target schema of a schema mapping $M = (\sigma, \tau, \Sigma)$ is to return its *certain answers* [29, 11, 27, 6, 4]. Given any source instance S for M, the *certain answers* to Q on M and S are defined as

$$\mathit{cert}(Q, M, S) \;:=\; \{\bar{a} \mid \bar{a} \in Q(T) \text{ for all solutions } T \text{ for } M \text{ under } S\}.$$

So, informally, a tuple $\bar{a}$ belongs to $\mathit{cert}(Q, M, S)$ whenever it is an answer to Q no matter on which of S's solutions Q is evaluated. Note that if Q is a UCQ (or any other domain independent query), this means that $Q(\bar{a})$ logically follows from S, viewed as a set of atomic formulas, and Σ.[3]

▶ **Example 4.** Let M and S be as in Example 2, and consider the UCQ

$$Q(x) \;:=\; \exists y \left(\mathit{BookInfo}(y, \text{Comput. Compl.}) \wedge \mathit{Author}(x, y) \right),$$

which asks for all authors of "Comput. Compl." It is intuitively clear that the result of evaluating Q with respect to M and S should be "S. Arora" and "B. Barak." And indeed, $\mathit{cert}(Q, M, S) = \{\text{S. Arora}, \text{B. Barak}\}$. ◀

The *certain answers* have several good properties for query answering in data exchange and data integration. The most apparent one is their simple and natural definition. Another one is that in many practical settings it is possible to compute them in polynomial time (for fixed schema mappings and queries) from a single solution, namely a universal solution [11, 5], or by evaluating a suitably rewritten query over the source schema (cf., [29] and Chapter 5 of this book). For example, to compute $\mathit{cert}(Q, M, S)$ for a UCQ Q, we only need to evaluate Q on an arbitrary universal solution for S, and remove all tuples with nulls from its result:

▶ **Theorem 5** ([11]). *Let M be a schema mapping, let S be a source instance for M, and let Q be a UCQ over M's target schema. For any universal solution T for S under M,*

$$\mathit{cert}(Q, M, S) \;=\; \{\bar{c} \in Q(T) \mid \bar{c} \textit{ contains no nulls}\}.$$

[3] Here we use the standard first-order semantics. That is, $Q(\bar{a})$ logically follows from S and Σ if for all instances I with $S \subseteq I$ and $I \models \Sigma$ we have $I \models Q(\bar{a})$.

In particular, if M and Q are fixed, and M is such that for any source instance S for M, S has a universal solution if it has a solution, and a universal solution for S can be computed in polynomial time, then $cert(Q, M, S)$ can be computed in polynomial time. By Theorem 3, schema mappings defined by st-tgds have this property. Much broader classes of schema mappings with this property are known, see, e.g., [11, 13, 16, 9, 28, 34, 15, 18] and Chapter 1 of this book.

▶ Remark. Extensions of universal solutions and Theorem 5 that are suitable for answering general monotone queries like UCQs with inequalities appeared in [9]. However, computing the certain answers to such queries (for fixed schema mappings and queries) is in co-NP, and co-NP-complete in general [11, 33]. Certain fragments of UCQs with inequalities were shown to be tractable, though [11, 5].

Despite their good properties, it has been realized that for *non-monotone* queries the certain answers may yield counter-intuitive results. We have illustrated the basic problem in Example 1. Other problems have been pointed out in [11, 3, 31], for example:

- A *copying schema mapping* is a schema mapping $M = (\sigma, \tau, \Sigma)$, where τ consists of copies R' for each $R \in \sigma$, and Σ consists of st-tgds $\forall \bar{x}\big(R(\bar{x}) \to R'(\bar{x})\big)$ for each $R \in \sigma$. For example, the schema mapping from Example 1 is a copying schema mapping. Although copying schema mappings intuitively say nothing else than to copy each relation R to the relation R', [3] showed that there is a copying schema mapping $M = (\sigma, \tau, \Sigma)$ and a simple FO-query Q (actually, a union of a CQ and a CQ with negation) that is not rewritable to an FO-query Q' over σ such that for every source instance S for M, $Q'(S) = cert(Q, M, S)$. They also proved that it is not rewritable to an FO-query Q' over τ such that for every source instance S for M, $Q'(T) = cert(Q, M, S)$ with $T \in \{\mathrm{Core}(M, S), \mathrm{CanSol}(M, S)\}$.
- As shown in [3], if M is a schema mapping defined by st-tgds, then for every Boolean FO-query Q over M's target schema, either $cert(Q, M, S) = \emptyset$ for all source instances S for M, or $cert(\neg Q, M, S) = \emptyset$ for all source instances S for M. To see this, suppose that $cert(Q, M, S) \neq \emptyset$ for some source instance S for M. Then for all solutions T for S under M we have $T \models Q$. Now, if S' is an arbitrary source instance for M, it is not hard to see that there is a solution T' for S' under M that is also a solution for S under M. Then, $T' \not\models \neg Q$, and therefore $cert(\neg Q, S', M) = \emptyset$. It follows that either Q or $\neg Q$ has a trivial answer, namely $\emptyset$, that does not depend on the source instance.

As already pointed out in the introduction, the problem is really a matter of the semantics of schema mappings, that is, a matter of which target instances of a schema mapping M are considered as solutions for a source instance under M. One way to enforce a suitable set of solutions would be to use a more expressive constraint language for specifying schema mappings. This approach is certainly worth pursuing, but to the best of my knowledge it seems to have received almost no attention in the literature to date. The approaches proposed in the literature are based on the certain answers to queries with respect to a restricted set of solutions.

4 Semantics for Non-Monotone Queries

A variety of semantics for answering non-monotone queries over the target schema of a schema mapping have been proposed in the literature [13, 22, 32, 2, 21]. These semantics are based on the following basic idea: for each schema mapping M and each source instance S, define an appropriate set $[\![M, S]\!]$ of solutions, and answer queries Q by the certain answers to Q on $[\![M, S]\!]$, that is, $\{\bar{a} \mid \bar{a} \in Q(T) \text{ for all } T \in [\![M, S]\!]\}$.

In [13], it is proposed to let $[\![M, S]\!]$ be the set of all universal solutions for S under M. However, the resulting semantics has similar problems as the certain answers semantics [3].[4] Therefore, we will not consider that semantics here.

The semantics proposed in [22, 32, 2, 21] are based on *non-monotonic reasoning*, specifically variants of the *Closed World Assumption (CWA)* [36], to arrive at the sets $[\![M, S]\!]$. This means that a solution will be in $[\![M, S]\!]$ if it can be derived in a certain way from M and S. The goal is always to define a set of solutions that intuitively captures "precisely the positive information in M and S, and nothing more." This is quite natural, since – as Libkin argued in [31] – in data exchange (but the same applies to data integration), data is moved from source to target according to the tgds and egds of a schema mapping. Therefore, answers to queries should only depend on that data, and not on data that could later be added to the target database. For instance, this seems to be natural in Example 1 as we have argued there. In the following, we review these semantics in more detail.

4.1 Libkin's CWA-Semantics

The CWA-semantics [31, 23] (see also [22]) was the first semantics explicitly designed for answering non-monotone queries in data exchange. It was introduced by Libkin [31] for schema mappings defined by st-tgds, and extended by Schweikardt and myself [23] to schema mappings as considered in this chapter. As the name suggests, it[5] is based on the CWA. As mentioned above, this means that for answering a query Q on M and S we take into account only those solutions for S under M which can be derived in a certain way from M and S. We call such solutions *CWA-solutions*.

4.1.1 CWA-Solutions

Informally, CWA-solutions for a source instance S under a schema mapping M are all those solutions T for S under M that satisfy the following properties:

1. All atoms in T are justified in a certain sense by M and S.
2. Each justification for atoms is used at most once.
3. Each "positive statement" (Boolean conjunctive query) that is true in T logically follows from S and the set of tgds and egds in M. That is, T should not "invent" new facts compared to what can be inferred from S using M.

Below, we give an idea of how to formalize these informal requirements.

In the context of schema mappings defined by st-tgds, the requirements can be formalized as follows. Assume that M is defined by st-tgds. Regarding the first requirement, an atom is *justified* if it can be obtained from S by means of "applying" an st-tgd in M to S, where st-tgds are considered as rules for deriving new atoms, similar to Datalog rules. A *justification for atoms* consists of an st-tgd θ in M, say $\theta = \forall\bar{x}\forall\bar{y}(\varphi(\bar{x}, \bar{y}) \to \exists\bar{z}\,\psi(\bar{x}, \bar{z}))$, and assignments $\bar{a}, \bar{a}'$ to $\bar{x}, \bar{y}$ such that $S \models \varphi(\bar{a}, \bar{a}')$. We denote it by $(\theta, \bar{a}, \bar{a}')$. An atom in T is justified if there is a justification $(\theta, \bar{a}, \bar{a}')$ with $\theta = \forall\bar{x}\forall\bar{y}(\varphi(\bar{x}, \bar{y}) \to \exists\bar{z}\,\psi(\bar{x}, \bar{z}))$, and an assignment $\bar{b}$ for $\bar{z}$ such that $T \models \psi(\bar{a}, \bar{b})$, and the atom is one of the relation atoms in $\psi(\bar{a}, \bar{b})$. The second requirement insists that each justification j is "used" with a unique assignment $\bar{b}_j$ for the existentially quantified variables of the st-tgd in j. An atom in T is then justified if there is a justification j such that the atom is justified by j as above, except that the assignment $\bar{b}$

[4] Example 1 is true for the universal solution-based semantics, too.
[5] As we shall see below, the CWA-semantics form a family of semantics. But for the moment, let us refer to this family as the CWA-semantics.

must be the assignment $\bar{b}_j$. It was shown in [31, 22] that a solution T for S under M satisfies the two requirements precisely if T is a homomorphic image of $\mathrm{CanSol}(M, S)$. Furthermore, the third requirement, once properly formalized, turns out to be equivalent to the property of being a *universal solution*. Hence:

▶ **Theorem 6** ([31, 22]). *Let M be a schema mapping defined by st-tgds, and let S be a source instance for M. A solution T for S under M is a CWA-solution for S under M iff*

1. *T is a homomorphic image of* $\mathrm{CanSol}(M, S)$*, and*
2. *T is a universal solution for S under M*
   (or, equivalently, there is a homomorphism from T to $\mathrm{CanSol}(M, S)$*).*

This characterization immediately implies that $\mathrm{CanSol}(M, S)$ is the “maximal” CWA-solution for S under M up to isomorphism in the sense that $\mathrm{CanSol}(M, S)$ is a CWA-solution for S, and that every CWA-solution for S is a homomorphic image of $\mathrm{CanSol}(M, S)$. Furthermore, it was shown in [31, 22] that $\mathrm{Core}(M, S)$ is the unique “smallest” CWA-solution for S under M up to isomorphism.

▶ **Example 7.** Let M be the schema mapping defined by the st-tgd θ from Example 1, and let S be the source instance exhibited in the same example. Then there is a unique justification consisting of θ, and assigning x, y the values c, d. Since θ has no existentially quantified variables, the only atom that can be justified using this justification is $E'(c, d)$. Hence, $T := \{E'(c, d)\}$ is the unique CWA-solution for S under M. Indeed, we have $T = \mathrm{Core}(M, S) = \mathrm{CanSol}(M, S)$. ◀

▶ **Example 8.** Let M, S and T be as in Example 2. In the same example, we mentioned that $T = \mathrm{CanSol}(M, S) = \mathrm{Core}(M, S)$. Hence, T is the unique CWA-solution for S under M up to isomorphism. ◀

To lift Libkin’s CWA-semantics to schema mappings defined by st-tgds, t-tgds and egds, it is necessary to formalize the first two requirements above for such schema mappings. In [23, 22], this is done using a derivation-based approach using a suitably controlled version of the *chase procedure*. In addition, [24] (see also [20]) introduces a *2-player game* and characterizes the requirements using this game. It is shown that CWA-solutions can still be characterized as particular universal solutions, and that the core solution, if it exists, is the “smallest” CWA-solution up to isomorphism. On the other hand, in general there is no “maximal” CWA-solution, that is, a CWA-solution with the same properties as the canonical solution in the context of schema mappings defined by st-tgds.

Concerning the question whether a given source instance has a CWA-solution, it is easy to see that a source instance has a CWA-solution whenever it has a universal solution. If M is a schema mapping defined by st-tgds, by Theorem 3, we can even compute CWA-solutions in polynomial time. For most of the classes of schema mappings mentioned in Section 3, for which universal solutions can be computed in polynomial time (data complexity), it is possible to compute CWA-solutions in polynomial time (data complexity). However:

▶ **Theorem 9** ([23, 22]). *There is a schema mapping $M = (\sigma, \tau, \Sigma)$ with Σ consisting only of st-tgds and t-tgds such that the following problem is undecidable: Given a source instance S for M, is there a CWA-solution for S under M?*

4.1.2 Query Answering under the CWA

Given a schema mapping M and a source instance S for M, it seems now perfectly reasonable to answer queries Q over M’s target schema by the certain answers to Q on the CWA-solutions for S under M, that is, by the set of all tuples that are answers to Q on all CWA-solution for S under M. However, we should be careful about what constitutes the result of a query on an individual CWA-solution. To explain why, we need a little background on incomplete instances.

An *incomplete σ-instance* is a set $\mathcal{I}$ of ground σ-instances [1, 37]. The idea is that $\mathcal{I}$ represents an unknown instance I, and the instances in $\mathcal{I}$ are the possibilities for I. Every σ-instance I represents an incomplete σ-instance. This is because nulls, which may occur in I, are place-holders for unknown constants, and therefore, any instance obtained from I by substituting constants for the nulls in I is a ground instance that could possibly be represented by I. Consequently, I represents the incomplete σ-instance

$$\mathit{rep}(I) := \{h(I) \mid h\colon \mathrm{dom}(I) \to \mathit{Const},\ h \text{ is the identity on } \mathrm{const}(I)\}\,.$$

Here we are more interested in incomplete instances represented by CWA-solutions. Technically, CWA-solutions are instances I together with a set Σ of integrity constraints (the set of t-tgds and egds of the schema mapping). Several ways of associating an incomplete instance with such an instance have been proposed (see, e.g., [1, 37]). We choose the one proposed in [26, 37], which is

$$\mathit{rep}_\Sigma(I) := \{J \mid J \in \mathit{rep}(I),\ J \models \Sigma\}\,.$$

Now, if I is an instance with nulls, and Q is a non-monotone query, returning $Q(I)$ as the answer to Q on I may lead to counter-intuitive results [26], mainly due to the fact that distinct nulls may represent the same constant. To circumvent this, one typically uses semantics designed for answering queries on incomplete instances. There are several such semantics, but the most common one is the *certain answers semantics* [1, 37]. The *certain answers* to a query Q on an incomplete instance $\mathcal{I}$ are defined by:

$$\mathit{cert}(Q, \mathcal{I}) := \{\bar{a} \mid \bar{a} \in Q(I) \text{ for all } I \in \mathcal{I}\}\,.$$

For an instance I and a set Σ of constraints, we let

$$\mathit{cert}(Q, I) := \mathit{cert}(Q, \mathit{rep}(I)) \quad \text{and} \quad \mathit{cert}_\Sigma(Q, I) := \mathit{cert}(Q, \mathit{rep}_\Sigma(I)).$$

▶ Remark. It is no coincidence – and will do no harm – that we use the same name both for the certain answers with respect to schema mappings and source instances, and for the certain answers with respect to incomplete instances. Indeed, the set of solutions for a source instance S under a schema mapping M is almost an incomplete instance $\mathcal{T}$, except that it may contain non-ground instances (think of each solution in $\mathcal{T}$ as a possible outcome of translating S to the target).

For answering queries over target schemas of schema mappings, [22] propose the following variant of the certain answers on CWA-solutions:[6]

▶ **Definition 10.** Let $M = (\sigma, \tau, \Sigma_{\mathrm{st}} \cup \Sigma_{\mathrm{t}})$ be a schema mapping, where Σ_{st} is a set of st-tgds and Σ_{t} is a set of t-tgds and egds, let S be a source instance for M, and let Q be a query over τ. Then the set of the *CWA-answers to Q on M and S* is defined as

$$\mathit{cert}_{\mathrm{CWA}}(Q, M, S) := \{\bar{a} \mid \bar{a} \in \mathit{cert}_{\Sigma_{\mathrm{t}}}(Q, T) \text{ for all CWA-solutions } T \text{ for } S \text{ under } M\}.$$

[6] Three more semantics have been proposed in [22]. For brevity, we consider only the most basic one.

The following example shows that for the schema mapping, source instance and query in Example 1, the CWA-answers leads to the desired result.

▶ **Example 11.** Let M and S be as in Example 7. As shown in Example 7, $T := \{E'(c,d)\}$ is the unique CWA-solution for S under M. Note that $\mathit{rep}(T) = \{T\}$, since T contains no nulls. In particular, for the query $Q(x,y)$ from Example 1, we have $\mathit{cert}(Q,T) = \{(c,d)\}$, and therefore $\mathit{cert}_{\mathrm{CWA}}(Q,M,S) = \{(c,d)\}$, as desired. ◀

More generally, if M is a copying schema mapping, then every source instance S for M has a unique CWA-solution S', namely its copy, and the CWA-answers to a query Q on M and S are precisely $Q(S')$, as desired. This implies that Q can be trivially rewritten into a query Q' over M's source schema such that for all source instances S we have $Q'(S) = \mathit{cert}_{\mathrm{CWA}}(Q,M,S)$. Hence, the CWA-semantics remedies the problems of the certain answers semantics on non-monotone queries described at the end of Section 3.

Let us finally consider an example that involves st-tgds with existential quantifiers:

▶ **Example 12.** Let M, S and T be as in Example 2. As shown in Example 8, T is the unique CWA-solution up to isomorphism. Consider the query

$$Q(t) := \exists^{=1} a\, \exists x\, \big(\mathit{BookInfo}(x,t) \wedge \mathit{Author}(a,x)\big),$$

which, intuitively, asks for all single-authored books. However, $\mathit{cert}_{\mathrm{CWA}}(Q,M,S) = \emptyset$, since "Comput. Compl." cannot be in $\mathit{cert}_{\mathrm{CWA}}(Q,M,S)$, and $\mathit{rep}(T)$ contains an instance obtained from T by replacing $\bot_1, \bot_2, \bot_3$ with the same constant. On the other hand, this is not really surprising, since M does not tell us that $\bot_1, \bot_2, \bot_3$ could not represent the same value.

Now let M' be the extension of M by the egd

$$\eta := \forall x \forall y_1 \forall y_2 \big(\mathit{BookInfo}(x,y_1) \wedge \mathit{BookInfo}(x,y_2) \rightarrow y_1 = y_2\big).$$

Then, T would still be the unique CWA-solution for S under M', but $\mathit{cert}(Q,M',S) = \{\text{Model Theory}\}$, since $\mathit{rep}_{\{\eta\}}(T)$ does not contain any instance that arises from T by mapping $\bot_3$ to the same constant as $\bot_1$ or $\bot_2$. ◀

For a number of schema mappings, including schema mappings defined by st-tgds, the task of computing the CWA-answers to a query can be reduced – as in the case of the certain answers semantics, explained in Section 3 – to the task of evaluating the query on a single incomplete instance, which is a well-studied topic, see, e.g., [1, 37]:

▶ **Theorem 13** ([22])**.** *Let $M = (\sigma, \tau, \Sigma)$ be a schema mapping where Σ is a set of st-tgds, let S be a source instance for M, and let Q be a query over τ. Then:*

$$\mathit{cert}_{\mathrm{CWA}}(Q,M,S) = \mathit{cert}(Q, \mathrm{CanSol}(M,S)).$$

The result also holds for schema mappings defined by st-tgds and egds with $\mathrm{CanSol}(M,S)$ extended appropriately.

4.2 A Relaxation of the CWA-Semantics

The CWA interprets existential quantifiers in tgds in a very restrictive way. For instance, in Example 8, each entry in Book^S introduces precisely one null which corresponds to a new value assigned to the variable z in the unique st-tgd in M. In some cases, like Example 2, this might be desirable. But in other cases, this might be too restrictive.

▶ **Example 14** ([22]). Let $M = (\sigma, \tau, \Sigma)$ be a schema mapping, with σ containing a unary relation symbol *Person*, τ containing a binary relation symbol *Child*, and Σ containing a single st-tgd

$$\theta := \forall x \big(\mathit{Person}(x) \to \exists y \, \mathit{Child}(x, y) \big).$$

Intuitively, θ states that for each person x there is a child y. Certainly, there could be more than one child. However, under the CWA, the effect of θ would be that each person x has *exactly* one child y. Indeed, if S is a source instance for M, then there is a unique CWA-solution T for S under M which assigns to each person $p \in \mathit{Person}^S$ a unique null $\perp_p$ such that $(p, \perp_p) \in \mathit{Child}^T$. Hence, the CWA-answers to the Boolean query $\forall x \exists^{=1} y \, \mathit{Child}(x, y)$ on M and S would yield true (i.e., a non-empty result), even though this was not intended. ◀

Libkin and Sirangelo [32] propose a relaxation of the CWA, which admits finer control over the degree of "closedness." The basic idea is to control for each position in a solution whether it should be *open* for adding new values at this position, or *closed*. In the following, we illustrate the basic idea with an example.

▶ **Example 15** (Example 14, continued). Let us annotate each occurrence of a variable in the head[7] of θ as *closed* (cl) or *open* (op) as follows:

$$\forall x \big(\mathit{Person}(x) \to \exists y \, \mathit{Child}(x^{\mathrm{cl}}, y^{\mathrm{op}}) \big).$$

Then the annotated version of θ induces the following annotated version of $\mathrm{CanSol}(M, S)$ for the source instance S with $\mathit{Person}^S = \{p_1, p_2\}$:

$$T = \{ \mathit{Child}(p_1^{\mathrm{cl}}, \perp_1^{\mathrm{op}}), \mathit{Child}(p_2^{\mathrm{cl}}, \perp_2^{\mathrm{op}}) \}.$$

The basic idea is that at a position annotated with *op* we may "insert" arbitrary many values, while at a position annotated with *cl*, the value is fixed. That is, the atom $\mathit{Child}(p_i^{\mathrm{cl}}, \perp_i^{\mathrm{op}})$ corresponds to "there exist one or more c with $\mathit{Child}(p_i, c)$," as desired. ◀

More generally, let $M = (\sigma, \tau, \Sigma)$ be a schema mapping, where Σ is a set of st-tgds. The starting point is always an annotation α of the positions in the heads of the st-tgds in M, which must be provided by the user. To be precise, a *position* in the head of an st-tgd

$$\theta := \forall \bar{x} \forall \bar{y} \left(\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \bigwedge_{i=1}^{k} R_i(\bar{u}_i) \right)$$

is represented by a pair (i, j), where $i \in [k]$ and $j \in [\mathrm{ar}(R_i)]$. Such a pair corresponds to the variable at position j in $\bar{u}_i$. Then for each st-tgd $\theta \in \Sigma$ and each position (i, j) in θ's head, we have an annotation $\alpha(\theta, i, j) \in \{\mathrm{cl}, \mathrm{op}\}$. For instance, the annotation of the st-tgd in Example 15 corresponds to $\alpha(\theta, 1, 1) = \mathrm{cl}$ and $\alpha(\theta, 1, 2) = \mathrm{op}$.

Given a source instance S, we now define the annotated canonical solution $\mathrm{CanSol}_\alpha(M, S)$ for S under M and α, which is a set of pairs $(R(u_1, \dots, u_k), \alpha')$ consisting of an atom $R(u_1, \dots, u_k)$ from $\mathrm{CanSol}(M, S)$, and an annotation $\alpha' \colon [k] \to \{\mathrm{cl}, \mathrm{op}\}$. The construction is as indicated in Example 15: starting from an empty set, for each st-tgd $\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \, \psi(\bar{x}, \bar{z}))$ and each assignment $\bar{a}, \bar{a}'$ for $\bar{x}, \bar{y}$ with $S \models \varphi(\bar{a}, \bar{a}')$, we pick a tuple $\bar{b}$ of pairwise distinct fresh nulls and add all pairs (A, α') to the set with A an atom of $\psi(\bar{a}, \bar{b})$, and α' the annotation induced by α on A.

[7] Given a tgd θ of the form $\varphi(\bar{x}, \bar{y}) \to \exists \bar{z} \, \psi(\bar{x}, \bar{z})$, we refer to the formula $\psi(\bar{x}, \bar{z})$ as the head of θ.

As in the case of $\mathrm{CanSol}(M,S)$, the annotated canonical solution $\mathrm{CanSol}_\alpha(M,S)$ represents an incomplete instance, denoted by $\mathit{rep}(\mathrm{CanSol}_\alpha(M,S))$, which is then used for query answering. Before we give its definition, let us continue with our example.

▶ **Example 16** (Example 14, continued). Let α be the annotation of θ's head as shown in Example 15. Recall $T = \mathrm{CanSol}_\alpha(M,S)$ from the same example, and that the meaning of each atom $\mathit{Child}(p_i^{\mathrm{cl}}, \bot_i^{\mathrm{op}})$ in T is "there exist one or more c with $\mathit{Child}(p_i,c)$." Hence, T represents the incomplete instance $\mathit{rep}(T)$ which consists of all τ-instances T' such that $\mathit{Child}^{T'}$ contains tuples (p_1,a) and (p_2,b) with possibly identical constants a,b, and all tuples in $\mathit{Child}^{T'}$ have the form (p_i,c) with $i \in [2]$ and $c \in \mathit{Const}$. In particular, each $T' \in \mathit{rep}(T)$ contains tuples (p_1,a) and (p_2,b), which represents the information that for each $i \in [2]$ there is at least one c with (p_i,c). Furthermore, for any such tuple we can add a new tuple by replacing the values at the positions annotated as "open" with new values. The resulting instance would be in $\mathit{rep}(T)$, too. ◀

In general, $\mathit{rep}(\mathrm{CanSol}_\alpha(M,S))$ consists of all ground instances T such that there is a homomorphism h from $\mathrm{CanSol}(M,S)$ to T with $h(\mathrm{CanSol}(M,S)) \subseteq T$, and for each atom $R(a_1,\dots,a_k) \in T \setminus h(\mathrm{CanSol}(M,S))$ there is a pair $(R(b_1,\dots,b_k),\alpha') \in \mathrm{CanSol}_\alpha(M,S)$ such that $a_i = h(b_i)$ for all positions $i \in [k]$ with $\alpha'(i) = \mathrm{cl}$. That is, each atom A in $T \setminus h(\mathrm{CanSol}(M,S))$ coincides with an atom from $h(\mathrm{CanSol}(M,S))$ on all positions that are annotated as "closed"; A may have arbitrary values at positions annotated as "open".

Analogous to Theorem 13, which characterizes the certain answers to Q on M and S by $\mathit{cert}(Q, \mathit{rep}(\mathrm{CanSol}(M,S)))$, Libkin and Sirangelo propose to answer Q on (M,α) and S by[8]

$$\mathit{cert}(Q,M,\alpha,S) \ := \ \mathit{cert}\big(Q, \mathit{rep}(\mathrm{CanSol}_\alpha(M,S))\big).$$

Note that under this semantics, the answers to a query depend on the annotation α. Note also that for the annotation that assigns to each position in the head of a st-tgd the label *cl*, we obtain the CWA-semantics. The other extreme is to assign to each position the label *op*. In this case, we arrive at the certain answers semantics as introduced in Section 3. For details, the reader is referred to [32].

▶ **Example 17** (Example 14, continued). Let α be the annotation of θ's head as shown in Example 15. For the Boolean query

$$Q \ := \ \forall x \exists^{=1} y \ \mathit{Child}(x,y)$$

from Example 14 we have $\mathit{cert}(Q,M,\alpha,S) = \emptyset$, as desired. ◀

Libkin and Sirangelo introduced their "mixed world" semantics for schema mappings defined by st-tgds only, and left open the task of extending it to more general schema mappings.

▶ Remark. Afrati and Kolaitis [2] proposed a much stricter version of the CWA, and argued that it leads to an interesting semantics for *aggregate queries* under schema mappings defined by st-tgds. Recall that for such schema mappings M, if S is a source instance and Q is a query over M's target schema, the CWA-answers to Q on M and S can be characterized as $\mathit{cert}(Q, \mathrm{CanSol}(M,S))$. That is, a tuple belongs to the set of CWA-answers to Q on M and S iff it belongs to $Q(T)$ for all $T \in \mathit{rep}(\mathrm{CanSol}(M,S))$. Afrati and Kolaitis argue that

[8] In fact, this is a characterization of their semantics, in the same way as Theorem 13 is a characterization of the CWA-semantics.

the CWA-semantics is too weak in the context of aggregate queries, since $\mathit{rep}(\mathrm{CanSol}(M,S))$ may contain instances with values that do not occur in S, and propose a new semantics based on the set of all *endomorphic images* of $\mathrm{CanSol}(M,S)$. Here, an instance I is an *endomorphic image* of an instance J if there is a homomorphism h from J to J such that $h(J) = I$. Given a query Q over M's target schema, we could now answer Q on M and S under this "endomorphic images semantics" by

$$\mathit{cert}_{\mathrm{endo}}(Q,M,S) \ := \ \{\bar{a} \mid \bar{a} \in Q(T) \text{ for all endomorphic images } T \text{ of } \mathrm{CanSol}(M,S)\}.$$

4.3 The GCWA*-Semantics

Consider two schema mappings $M_1 = (\sigma,\tau,\Sigma_1)$ and $M_2 = (\sigma,\tau,\Sigma_2)$ over the same source schema σ and the same target schema τ. We say that M_1 and M_2 are *logically equivalent* if Σ_1 and Σ_2 are logically equivalent under the standard FO-semantics[9] (i.e., for every $\sigma \cup \tau$-instance I we have $I \models \Sigma_1$ if and only if $I \models \Sigma_2$). If M_1 and M_2 are logically equivalent, it seems desirable that for each source instance S for M_1 (resp., M_2) and each query Q over τ, the answer to Q on M_1 and S is the same as the answer to Q on M_2 and S, since intuitively M_1 and M_2 specify the same translation of source data to the target. Yet, for the semantics introduced above, this is not necessarily true:

▶ **Example 18.** Let $M_1 = (\sigma,\tau,\Sigma_1)$ and $M_2 = (\sigma,\tau,\Sigma_2)$ with $\sigma = \{P\}$, $\tau = \{E\}$, and

$$\begin{aligned}\Sigma_1 &= \bigl\{\forall x\bigl(P(x) \to E(x,x)\bigr)\bigr\},\\ \Sigma_2 &= \Sigma_1 \cup \bigl\{\forall x\bigl(P(x) \to \exists y\, E(x,y)\bigr)\bigr\}.\end{aligned}$$

Clearly, M_1 and M_2 are logically equivalent. Now, for $S := \{P(c)\}$ we have

$$\begin{aligned}T_1 &:= \mathrm{CanSol}(M_1,S) = \{E(c,c)\},\\ T_2 &:= \mathrm{CanSol}(M_2,S) = \{E(c,c), E(c,\perp)\}.\end{aligned}$$

Hence, if

$$Q(x) \ := \ \exists^{=1} y\, E(x,y),$$

then $\mathit{cert}_{\mathrm{CWA}}(Q,M_1,S) = \{c\}$, since T_1 is the unique CWA-solution for S under M_1, while $\mathit{cert}_{\mathrm{CWA}}(Q,M_2,S) = \emptyset$, since T_2 is a CWA-solution for S under M_2. Analogously, we have $\mathit{cert}_{\mathrm{endo}}(Q,M_1,S) = \{c\}$ and $\mathit{cert}_{\mathrm{endo}}(Q,M_2,S) = \emptyset$.

Next we turn to the "mixed world" semantics. Fix an annotation α for the st-tgds in Σ_2. In particular, α yields an annotation for the st-tgd in Σ_1. As long as the second position in the head of the st-tgd in Σ_1 is annotated as *closed* by α, we have $\mathit{cert}(Q,M_1,\alpha,S) = \{c\}$ and $\mathit{cert}(Q,M_2,\alpha,S) = \emptyset$. Note that if the second position is annotated as *open*, the resulting semantics is very close to the certain answers semantics. ◀

There is a second issue related to the interpretation of tgds under the CWA-semantics, which is best illustrated with an example:

[9] Different notions of equivalence between schema mappings have been considered in [12]. Logical equivalence is the strongest such notion. Instead of invariance under logical equivalence one could also require invariance under any of the other notions of schema mapping equivalence.

▶ **Example 19.** Let $M = (\sigma, \tau, \Sigma)$ be defined by $\sigma = \{R\}$, $\tau = \{E, F\}$, and $\Sigma = \{\theta\}$, where

$$\theta \;=\; \forall x, y\Big(R(x, y) \to \exists z\,\big(E(x, z) \wedge F(z, y)\big)\Big).$$

Intuitively, θ states that "if $R(x, y)$, then there is at least one z such that $E(x, z)$ and $F(z, y)$ hold." There could be exactly one such z, but there could also be more than one such z. In particular, the possibility that there are precisely two such z, or precisely three such z etc. is perfectly consistent with θ, and should not be denied when answering queries. Hence, given a source instance S for M, we should expect that the answer to

$$Q(x, y) \;:=\; \exists^{=1} z\,\big(E(x, z) \wedge F(z, y)\big)$$

on M and S is empty.

However, if we consider the source instance $S = \{R(c, d)\}$, we have $\text{CanSol}(M, S) = \{E(c, \bot), F(\bot, d)\}$, so that $\mathit{cert}_{\text{CWA}}(Q, M, S) = \mathit{cert}_{\text{endo}}(Q, M, S) = \{(c, d)\}$. Hence, both the CWA-semantics and the endomorphic images semantics exclude the possibility that there is more than one z satisfying $E(x, z)$ and $F(z, y)$, although θ explicitly states that it is possible that more than one such z exists.

Note that the existential quantifier in θ can be expressed via an infinite disjunction over all possible choices of values for z (recall that nulls are just place-holders for unknown constants, so we do not have to consider nulls here):

$$\theta' \;:=\; \forall x, y\Big(R(x, y) \to \bigvee_{c \in \mathit{Const}} \big(E(x, c) \wedge F(c, y)\big)\Big).$$

Thus, we argued above that we need a semantics that interprets this disjunction *inclusively* rather than exclusively. ◀

Under the "mixed world" semantics, the query Q in Example 19 is answered as expected as long as the occurrences of z in θ are annotated as *open*. On the other hand, if the occurrences of z in θ are annotated as *open*, then, in a way, the "mixed world" semantics is "too open" in that it allows atoms to appear in solutions that intuitively cannot be justified by the source instance and the st-tgds.

▶ **Example 20** (Example 19, continued)**.** In light of the rewriting θ' of θ in Example 19 the only reasonable solutions for $S = \{R(c, d)\}$ under M seem to be those solutions T for which there is a finite set $X \subseteq \mathit{Dom}$ such that $T = \{E(c, x) \mid x \in X\} \cup \{F(x, d) \mid x \in X\}$. For example,

$$T^* \;:=\; \{E(c, e), E(c, e'), F(e, d)\}$$

should not be a valid solution, since the occurrence of $E(c, e')$ in T can intuitively not be explained in terms of S and θ (for this, $F(e', d)$ should be in T). Therefore, we should expect the answer to

$$Q'(x) \;:=\; \forall z\big(E(x, z) \to \exists y\, F(z, y)\big)$$

on M and S to be $\{c\}$. But let α be an annotation for θ that annotates the second position of $E(x, z)$ in θ as *open*. Then, $\mathit{cert}(Q', M, \alpha, S) = \emptyset$ since $T^* \in \mathit{rep}(\text{CanSol}_\alpha(M, S))$. Intuitively, the "mixed world" semantics is "too open" in that it allows $E(c, e')$ to occur in T^* without enforcing that the corresponding atom $F(e', d)$ is present in T^*. ◀

Motivated by the above examples, [21] proposes a new semantics, called *GCWA*-semantics*. This semantics is invariant under logically equivalent schema mappings, and interprets existential quantifiers in a natural way. A detailed discussion of the latter property can be found in [21]. The starting point for the development of the GCWA*-semantics was the observation that query answering with respect to schema mappings is very similar to query answering on *deductive databases* [14], and that non-monotone query answering on deductive databases is a well-studied topic in this area (see, e.g., [14, 36, 35, 38, 25, 8]). Therefore, it seemed obvious to use these semantics in the context of data exchange. For data exchange, the semantics based on Reiter's formalization of the CWA [36], and variants of the CWA like Minker's *generalized CWA (GCWA)* [35] seemed to be particularly interesting. It turns out, though, that these semantics are too strong, too weak, or do not have the desired properties. Nevertheless, their analysis provided a good starting point for developing the GCWA*-semantics. For details, see [21].

For schema mappings defined by st-tgds and egds, the GCWA*-semantics has a very simple definition in terms of *minimal solutions*. Here, a solution T for S under M is *minimal* if there is no solution T' for S under M with $T' \subsetneq T$.

▶ **Definition 21.** Let M be a schema mapping defined by st-tgds and egds, and let S be a source instance for M.

1. A *GCWA*-solution for S under M* is a ground solution that is the union of minimal solutions for S under M.
2. Given a query Q over M's target schema, the set of all *GCWA*-answers to Q on M and S* is defined by

$$\mathit{cert}_{\mathrm{GCWA}^*}(Q, M, S) \;:=\; \{\bar{a} \mid \bar{a} \in Q(T) \text{ for all GCWA}^*\text{-solutions for } S \text{ under } M\}\,.$$

▶ Remark. For schema mappings whose specification additionally contains t-tgds, an extended definition of GCWA*-solutions is necessary. See [21] for details.

It should be clear that the GCWA*-semantics is invariant under logical equivalent schema mappings. Therefore, the problem described at the beginning of this section does not appear for the GCWA*-semantics.

▶ **Example 22.** Recall Example 19. The minimal solutions for the source instance $S = \{R(c, d)\}$ under M are all solutions for S under M of the form $T_a := \{E(c, a), F(a, d)\}$ for some $a \in \mathit{Dom}$. Now, the GCWA*-solutions for S under M are precisely those instances T for which there is a finite set $C \subseteq \mathit{Const}$ with $T = \bigcup_{a \in C} T_a$. Intuitively, this reflects "precisely the positive information in M and S, and nothing more." It is easy to see that for the query Q from Example 19 and the query Q' from Example 20 we have $\mathit{cert}_{\mathrm{GCWA}^*}(Q, M, S) = \emptyset$ and $\mathit{cert}_{\mathrm{GCWA}^*}(Q', M, S) = \{c\}$, as desired.

▶ Remark. Gottlob et al. [17] propose a different approach of enforcing unique answers on logically equivalent schema mappings under the CWA-semantics (and its relatives). The idea is to first *normalize* the schema mapping as described in their paper, and then answer queries under the desired semantics.

5 The Complexity of Answering Non-Monotone Queries

This section's goal is to compile what is known about the complexity of answering queries under the different semantics introduced in Section 4, which we henceforth call *non-monotone semantics*. To the best of our knowledge, only the *data complexity* of this problem (i.e., its complexity as a function of the size of the source instance only) has been considered in the literature. For each $s \in \{\mathrm{CWA}, \mathrm{GCWA}^*\}$, each schema mapping M for which $cert_s$ is defined, and each FO-query Q over M's target schema, we will therefore consider the complexity of

$\textsc{Eval}_s(M, Q)$

Input: a source instance S for M, and a tuple $\bar{a}$ over $\mathrm{dom}(S) \cup \mathrm{dom}(Q)$

Question: Is $\bar{a} \in cert_s(Q, M, S)$?

Concerning the "mixed world semantics" from Section 4.2, an important parameter is the maximum number of *open positions per atom* in the head of an st-tgd. For an annotation α for the st-tgds in M, let us denote this number by $\#_{\mathrm{op}}(\alpha)$. Then, for each $k \geq 1$ we consider

$\textsc{Eval}_k(M, Q)$

Input: a source instance S for M, an annotation α for the st-tgds in M such that $\#_{\mathrm{op}}(\alpha) = k$, and a tuple $\bar{a}$ over $\mathrm{dom}(S) \cup \mathrm{dom}(Q)$

Question: Is $\bar{a} \in cert(Q, M, \alpha, S)$?

Note that for $k = 0$, the problem would correspond to $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$.

Let me point out that for monotone queries, most of the non-monotone semantics coincide with the certain answers semantics [22, 32, 2, 21]. The only exception is the CWA-semantics, but only in the context of schema mappings defined by st-tgds, t-tgds, and possibly also egds. This, however, seems to be only due to the choice we made for the incomplete instances represented by instances under integrity constraints. It might well be the case that this changes when we represent such instances in any of the other ways proposed in the literature. Anyway, the collapse to the certain answers semantics indicates once more that the certain answers semantics is well-suited for monotone queries. It also shows that all results concerning the certain answers semantics directly carry over to the non-monotone semantics. So, for example, we know from [33] that there are schema mappings M defined by st-tgds and CQs Q with only two inequalities such that $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$ is co-NP-hard. But we also know from [11] that the problem is in PTIME if Q is a UCQ with at most one inequality per disjunct. For the GCWA*-semantics, the latter is true even for the much broader class of *weakly acyclic* schema mappings considered in [11]. This does not hold for the CWA-semantics in general, since it does not coincide with the certain answers semantics on such schema mappings.

5.1 General FO-Queries

Not much is known about the complexity of answering *non-monotone* queries under the non-monotone semantics. In many cases, it is hard to evaluate such queries, which is not surprising given that the non-monotone semantics introduce implicit negation. Concerning the complexity of evaluating general FO-queries under the CWA-semantics and the mixed world semantics, we know:

▶ **Theorem 23** ([22, 32]). *If we restrict ourselves to schema mappings M defined by st-tgds, and* FO*-queries Q over M's target schema, then:*

1. $\textsc{Eval}_{CWA}(M, Q) \in$ co-NP*, and there is a schema mapping M defined by st-tgds, and a* FO*-query Q such that* $\textsc{Eval}_{CWA}(M, Q)$ *is* co-NP*-complete.*
2. $\textsc{Eval}_1(M, Q) \in$ co-NEXPTIME*, and there is a schema mapping M defined by st-tgds, and a* FO*-query Q such that* $\textsc{Eval}_1(M, Q)$ *is* co-NEXPTIME*-complete.*

3. *For all $k \geq 2$, there is a schema mapping M defined by st-tgds, and a* FO*-query Q such that* $\textsc{Eval}_k(M, Q)$ *is undecidable.*
4. *If Q has the form $\forall \bar{x} \exists \bar{y}\, \varphi$, where φ contains no quantifiers, then* $\textsc{Eval}_k(M, Q) \in$ co-NP *for all $k \geq 1$.*

The membership part of Theorem 23(2a) follows directly from Theorem 13, whereas for the hardness part we can use [33]. On the other hand, the membership part of Theorem 23(2b) requires more sophisticated techniques. It involves a games argument and is technically quite involved. Hardness is proved by a reduction from a NEXPTIME-complete version of the tiling problem to the complement of $\textsc{Eval}_1(M, Q)$. Theorem 23(2c) is established by a reduction from the finite validity problem for first-order logic [30].

▶ Remark. As shown in [22], the upper bound in Theorem 23(2a) holds for more general schema mappings, called *richly acyclic*, which form a subclass of the weakly acyclic schema mappings considered in [11]. It does not hold for weakly acyclic schema mappings, as there are weakly acyclic schema mappings M and FO-queries Q over M's target schema such that $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$ is undecidable [22].

Under the GCWA*-semantics, query answering seems to be harder:

▶ **Theorem 24** ([21]).
1. *There is a schema mapping M defined by st-tgds and a CQ Q with one negated atom such that* $\textsc{Eval}_{GCWA^*}(M, Q)$ *is* co-NP*-hard.*
2. *There is a schema mapping M defined by st-tgds and a* FO*-query Q of the form $\exists \bar{x} \forall y\, \varphi$, where φ contains no quantifiers, such that* $\textsc{Eval}_{GCWA^*}(M, Q)$ *is undecidable.*

Remember, though, that all of the above-mentioned results concern the data complexity of query evaluation. For instance, co-NP-hardness of $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$ holds for *some* schema mappings M and FO-queries Q, but there are many schema mappings M and FO-queries Q for which $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$ is easy. Think, for example, of schema mappings that contain only tgds without existentially quantified variables, and arbitrary FO-queries. For them, $\textsc{Eval}_{\mathrm{CWA}}(M, Q)$ is in PTIME. The same is of course true for $\textsc{Eval}_k$, $k \geq 1$, and $\textsc{Eval}_{\mathrm{GCWA}^*}$. It would be interesting to identify more general classes of schema mappings and queries for which these problems are in PTIME.

5.2 Universal Queries

A very general and natural class of queries for which tractability of query answering under a non-monotone semantics – namely, the GCWA*-semantics – could be established is the class of *universal queries*. Universal queries are FO-queries of the form $\forall \bar{y}\, \varphi$, where φ contains no quantifiers. For schema mappings defined by st-tgds, it is not hard to show:

▶ **Theorem 25** ([21]). *For all schema mappings M defined by st-tgds, and for all universal queries Q over M's target schema we have* $\textsc{Eval}_{GCWA^*}(M, Q) \in$ co-NP.

On the other hand, if we restrict consideration to schema mappings defined by *packed st-tgds*, then universal queries can be answered in *polynomial time*, as we shall see below.

▶ **Definition 26** (Packed st-tgd). An st-tgd $\forall \bar{x} \forall \bar{y} (\varphi(\bar{x}, \bar{y}) \to \exists \bar{z}\, \psi(\bar{x}, \bar{z}))$ is *packed* if every two distinct relation atoms in ψ share a common variable from $\bar{z}$.

▶ Remark. The schema mapping that is constructed in [21] for proving Theorem 24(2b) is defined by packed st-tgds.

Although quite restrictive, packed st-tgds still allow for non-trivial use of existentially quantified variables in heads of st-tgds. Note that every st-tgd $\forall\bar{x}\forall\bar{y}(\varphi(\bar{x},\bar{y}) \to \exists\bar{z}\,\psi(\bar{x},\bar{z}))$, where ψ contains at most two relation atoms with variables from $\bar{z}$, is logically equivalent to a set of packed st-tgds of size at most the number of relation atoms in ψ. Hence, the class of schema mappings defined by packed st-tgds forms an interesting class of schema mappings. An example of an st-tgd that is not packed is $\forall x(P(x) \to \exists y\exists z\exists u(E(x,y) \wedge E(y,z) \wedge E(z,u)))$. The remaining part of this section is devoted to the following result:

▶ **Theorem 27** ([21]). *Let M be a schema mapping defined by packed st-tgds, and let Q be a universal query over M's target schema. Then there is a polynomial time algorithm that takes the core solution for some source instance S for M as input and outputs $cert_{\mathrm{GCWA}^*}(Q,M,S)$.*

Combined with Theorem 3, this result leads to a polynomial time algorithm that takes a source instance S for M as input and outputs $cert_{\mathrm{GCWA}^*}(Q,M,S)$. In particular, we have $\mathrm{EVAL}_{\mathrm{GCWA}^*}(M,Q) \in \mathrm{PTIME}$. Also, recall from Section 3 that core solutions can be used to compute the certain answers to UCQs and other queries. As a consequence, one only needs to materialize the core solution in order to answer such queries and universal queries.

The proof of Theorem 27 is technically very involved. The core part consists of proving the following "decision variant" of Theorem 27:

▶ **Theorem 28** ([21]). *Let M be a schema mapping defined by packed st-tgds, and let Q be a universal query over M's target schema. Then there is a polynomial time algorithm that, given the core solution for some source instance S for M and a tuple $\bar{a}$ as input, decides whether $\bar{a} \in cert_{\mathrm{GCWA}^*}(Q,M,S)$.*

The remainder of this section presents a sketch of the proof of Theorem 28.

5.2.1 GCWA*-Answers and Core Solutions

Let us first see how GCWA*-answers can be obtained from core solutions. Consider a schema mapping M, a source instance S for M, an FO-query Q over M's target schema, and a tuple $\bar{a}$ over $\mathrm{dom}(S) \cup \mathrm{dom}(Q)$. Using the core solution for S under M, how can we decide whether $\bar{a} \in cert_{\mathrm{GCWA}^*}(Q,M,S)$?

Observe that $\bar{a} \notin cert_{\mathrm{GCWA}^*}(Q,M,S)$ if and only if there is a GCWA*-solution T' for S under M such that $\bar{a} \in \neg Q(T')$. Furthermore, recall that GCWA*-solutions are ground solutions that are the union of minimal solutions for S under M. In the case of schema mappings defined by st-tgds, this is equivalent to being a union of ground minimal solutions for S under M. Now let T be the core solution for S under M, and recall from Section 4.1.2 that it represents an incomplete instance $rep(T)$. The following lemma implies that $\bar{a} \notin cert_{\mathrm{GCWA}^*}(Q,M,S)$ if and only if there are $k \geq 1$ and minimal instances $T_1,\dots,T_k \in rep(T)$ such that $\bar{a} \in \neg Q(T_1 \cup \cdots \cup T_k)$.[10]

▶ **Lemma 29** ([21]). *Let M be a schema mapping defined by st-tgds, let S be a source instance for M, and let T be the core solution for S under M. Then the set of all ground minimal solutions for S under M coincides with the set of all minimal instances in $rep(T)$.*

If Q is a universal query, then $\neg Q$ is equivalent to an *existential query*, i.e., a FO-query of the form $\exists\bar{x}\,\varphi$, where φ is quantifier-free. Thus we have reduced the initial problem of deciding $\bar{a} \in cert_{\mathrm{GCWA}^*}(Q,M,S)$ to a *satisfiability problem for existential queries* over the

[10] As for solutions, an instance $T' \in rep(T)$ is *minimal* if there is no $T'' \in rep(T)$ with $T'' \subsetneq T'$.

set of all instances that are unions of minimal instances in $\mathit{rep}(T)$. Towards solving this satisfiability problem in its full generality, an important subproblem to be solved is the corresponding satisfiability problem for the case that Q has the form $\neg R(\bar{c})$ for some tuple $\bar{c}$ of constants. In this case, the problem simplifies to deciding whether there is a minimal instance in $\mathit{rep}(T)$ that contains $R(\bar{c})$. The next section shows how to find such an instance if one exists.

5.2.2 Finding Atoms in Minimal Possible Worlds

Let T be the core solution for S under M, and let $\mathit{rep}_{\min}(T)$ be the set of all minimal instances in $\mathit{rep}(T)$. Given an atom A, how can we find an instance $T' \in \mathit{rep}_{\min}(T)$ with $A \in T'$ if there is one? Note that in general there are infinitely many instances in $\mathit{rep}(T)$, since each null can be substituted by an arbitrary element of *Const*. However, as shown in [21], it suffices to restrict attention to finitely many representatives. Still, in the worst case there are exponentially many such representatives left, and it is not clear at all how to find a representative containing A.

A very nice structural property of core solutions under schema mappings defined by st-tgds comes to the rescue: that the number of nulls in the *atom blocks* of such core solutions does only depend on the schema mapping.

▶ **Definition 30** ([16])**.** The *Gaifman graph of the atoms* of T is the undirected graph which has the atoms of T as nodes, and an edge between two distinct atoms A and A' if there is a null that occurs both in A and A'. An *atom block* of T is a connected component in the Gaifman graph of the atoms of T.

It follows immediately from results in [13] that for every schema mapping M defined by st-tgds there is an integer s such that for every source instance S for M each atom block in the core solution for S under M contains at most s nulls. The obvious idea is now to try to look only at single atom blocks B of T, and search for an instance in $\mathit{rep}_{\min}(B)$ containing A. Unfortunately, this does not lead to a correct algorithm: If no instance in $\mathit{rep}_{\min}(B)$ contains A, then we can be sure that no instance in $\mathit{rep}_{\min}(T)$ contains A, but if there is an instance in $\mathit{rep}_{\min}(B)$ containing A, then this does not imply that there is also an instance in $\mathit{rep}_{\min}(T)$ that contains A. An example is given in [21].

Instead, it can be shown that there is a subset $\mathcal{S}$ of the representatives of instances in $\mathit{rep}_{\min}(T)$ such that $\mathcal{S}$ has size polynomial in the size of T, and such that it suffices to consider only the instances in $\mathcal{S}$ in order to decide whether there is an instance in $\mathit{rep}_{\min}(T)$ containing A. Furthermore, it is possible to enumerate the instances in $\mathcal{S}$ in polynomial time. The set $\mathcal{S}$ is defined using the following homomorphisms:

▶ **Definition 31.** Let B be an atom block of T, let $\bar{B} := T \setminus B$, and let $C \subseteq \mathit{Const}$.

- Let $\mathit{val}_C(T, B)$ be the set of all mappings $h\colon \operatorname{dom}(T) \to \operatorname{dom}(T) \cup C$ such that $h(c) = c$ for all $c \in \operatorname{const}(T)$, $h(\bot) = \bot$ for all $\bot \in \operatorname{nulls}(\bar{B})$, and for every atom $R(a_1, \ldots, a_k) \in B$ we have: if $R(h(a_1), \ldots, h(a_k)) \notin B$, then every null that occurs in $R(h(a_1), \ldots, h(a_k))$ also occurs in B.
- Let $\mathit{minval}_C(T, B)$ be the set of all mappings $h \in \mathit{val}_C(T, B)$ such that there is no $h' \in \mathit{val}_C(T, B)$ with $h'(T) \subsetneq h(T)$.

Let C be the set of all constants that occur in A. Then the set $\mathcal{S}$ is the set of all instances that are the core of $h(T)$ for some $h \in \mathit{minval}_C(T, B)$ and some atom block B of T. For enumerating $\mathcal{S}$, we simply enumerate all the atom blocks B of T and all $h \in \mathit{minval}_C(T, B)$, and compute the core of $h(T)$ using a slight modification of the *blocks algorithm* from [13].

Showing that (1) the instances in $\mathcal{S}$ are indeed representatives of instances in $rep_{\min}(T)$, and that (2) for every atom A contained in some instance in $rep_{\min}(T)$ there is an instance in $\mathcal{S}$ containing A is technically involved. For proving (2), the property that M is defined by packed st-tgds is very important. For details the reader should consult [21].

5.2.3 Solving the General Satisfiability Problem

Finally, let us see how we can put together the results so as to solve the satisfiability problem in its full generality. Let $M = (\sigma, \tau, \Sigma)$ be a schema mapping defined by packed st-tgds, and let Q be an existential query over τ. Given the core solution T for a source instance S under M and a tuple $\bar{a}$ over $\mathrm{dom}(S) \cup \mathrm{dom}(Q)$, how can we decide whether there are $k \geq 1$ and $T_1, \ldots, T_k \in rep_{\min}(T)$ such that $\bar{a} \in Q(T_1 \cup \cdots \cup T_k)$?

We first observe that Q is logically equivalent to a query of the form

$$Q'(\bar{x}) := \bigvee_{i=1}^{m} Q_i(\bar{x}),$$

where each $Q_i(\bar{x})$ is an existential query of the form

$$Q_i(\bar{x}) := \exists \bar{y}_i \bigwedge_{j=1}^{n_i} \varphi_{i,j},$$

and each $\varphi_{i,j}$ is an atomic FO-formula, or the negation of an atomic FO-formula. It therefore remains to decide whether for some $i \in [m]$, there are $k \geq 1$ and $T_1, \ldots, T_k \in rep_{\min}(T)$ such that $\bar{a} \in Q_i(T_1 \cup \cdots \cup T_k)$.

We can do this as follows. Let $i \in [m]$. For simplicity, assume that the relation atoms in Q_i are $\varphi_{i,1}, \ldots, \varphi_{i,\ell}$. Then for all $j \in [\ell]$, we consider the set $\mathcal{T}_j$ of all pairs (T_j, α_j), where T_j is an instance in the set $\mathcal{S}$ mentioned in Section 5.2.2, and α_j is an assignment for $\varphi_{i,j}$ with $(T_j, \alpha_j) \models \varphi_{i,j}$. Combining tuples from $\mathcal{T}_1, \ldots, \mathcal{T}_\ell$ that are *compatible* in a certain sense,[11] we obtain a set $\mathcal{T}$ of pairs (T^*, α^*) such that T^* has the form $T_1 \cup \cdots \cup T_k$ with each T_i isomorphic to an instance in $\mathcal{S}$, and α^* is an assignment for $\bigwedge_{j=1}^{\ell} \varphi_{i,j}$ such that $(T^*, \alpha^*) \models \bigwedge_{j=1}^{\ell} \varphi_{i,j}$. Finally, we check whether there is a pair $(T^*, \alpha^*) \in \mathcal{T}$ such that T^* can be padded with a large enough number of disjoint copies of T so that the resulting instance T^{**} satisfies $\bar{a} \in Q_i(T^{**})$. For the proof of correctness, which is technically quite involved, I refer the reader to [21].

6 Conclusions

Query answering is a fundamental task in data exchange and data integration. The standard semantics for queries in these areas is the certain answers semantics. While this is adequate for monotone queries, it may lead to counter-intuitive answers for *non-monotone* queries. This chapter surveyed various semantics (the CWA-semantics, the "mixed world" semantics, the endomorphic images semantics, and the GCWA*-semantics) that were designed for answering non-monotone queries. Each of these semantics is based on a variant of the CWA to reduce the set of solutions, and to answer queries with respect to the reduced set of

[11] Informally, (T_1, α_1) and (T_2, α_2) are compatible if T_1 and T_2 can be "glued together" by identifying the values assigned to some variable that occurs both in the domain of α_1 and in the domain of α_2, while leaving the other values untouched.

solutions. Answering non-monotone queries under any of these semantics may be co-NP-hard, or co-NEXPTIME-hard (in the case of the "mixed world" semantics with at most one open position per atom in an st-tgd), or even undecidable. Note, however, that these results speak about the *data complexity* of the problem. In particular, single schema mappings M and queries Q were exhibited for which the problem is hard. For many schema mappings and queries, the problem is easy, though, and it would be interesting to identify more general classes of schema mappings and queries for which the problem is tractable. We presented one such example: the GCWA*-answers to universal queries can be computed in polynomial time under schema mappings defined by packed st-tgds.

Apart from identifying more general classes of schema mappings and queries for which query answering is tractable, there are many more problems that still need to be solved. To give three examples: Since there are several semantics for non-monotone queries, it would be nice to have *formal criteria* (e.g., in the style of [7], see also [10]) for comparing them and to understand their strengths and weaknesses relative to each other. Furthermore, it is worth studying the *combined complexity* of query answering under non-monotone semantics. There are a few results on the combined complexity of computing the certain answers [5], but for non-monotone semantics there are no such results. Finally, a technical question concerning the polynomial time algorithm for computing GCWA*-answers to universal queries (Theorem 27): Can it be *extended* to more general schema mappings? It seems possible to do this for schema mappings defined by st-tgds.

References

1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

2 F. N. Afrati and P. G. Kolaitis. Answering aggregate queries in data exchange. In *Proceedings of the 27th ACM Symposium on Principles of Database Systems (PODS)*, pages 129–138, 2008.

3 M. Arenas, P. Barceló, R. Fagin, and L. Libkin. Locally consistent transformations and query answering in data exchange. In *Proceedings of the 23th ACM Symposium on Principles of Database Systems (PODS)*, pages 229–240, 2004.

4 M. Arenas, P. Barceló, L. Libkin, and F. Murlak. *Relational and XML Data Exchange*. Morgan & Claypool, 2010.

5 M. Arenas, P. Barceló, and J. Reutter. Query languages for data exchange: Beyond unions of conjunctive queries. *Theory of Computing Systems*, 49(2):489–564, 2011.

6 P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.

7 S. Brass and J. Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *Journal of Logic Programming*, 32(3):207–228, 1997.

8 E. P. F. Chan. A possible world semantics for disjunctive databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):282–292, 1993.

9 A. Deutsch, A. Nash, and J. Remmel. The chase revisited. In *Proceedings of the 27th ACM Symposium on Principles of Database Systems (PODS)*, pages 149–158, 2008.

10 J. Dix, U. Furbach, and I. Niemelä. Nonmonotonic reasoning: Towards efficient calculi and implementations. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 19, pages 1241–1354. The MIT Press, 2001.

11 R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005.

12 R. Fagin, P. G. Kolaitis, A. Nash, and L. Popa. Towards a theory of schema-mapping optimization. In *Proceedings of the 27th Symposium on Principles of Database Systems (PODS)*, pages 33–42, 2008.

13 R. Fagin, P. G. Kolaitis, and L. Popa. Data exchange: Getting to the core. *ACM Transactions on Database Systems*, 30(1):174–210, 2005.

14 H. Gallaire, J. Minker, and J.-M. Nicholas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, 1984.

15 F. Geerts and B. Marnette. Static analysis of schema-mappings ensuring oblivious termination. In *Proceedings of the 13th International Conference on Database Theory (ICDT)*, pages 183–195, 2010.

16 G. Gottlob and A. Nash. Efficient core computation in data exchange. *Journal of the ACM*, 55(2):Article 9, 2008.

17 G. Gottlob, R. Pichler, and V. Savenkov. Normalization and optimization of schema mappings. *The VLDB Journal*, 20(2):277–302, 2011.

18 S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1):93–104, 2010.

19 P. Hell and J. Nešetřil. The core of a graph. *Discrete Mathematics*, 109(1–3):117–126, 1992.

20 A. Hernich. *Foundations of Query Answering in Relational Data Exchange*. PhD thesis, Institut für Informatik, Goethe-Universität Frankfurt am Main, 2010. Published at Logos Verlag Berlin, ISBN 978-3-8325-2735-8, 2010.

21 A. Hernich. Answering non-monotonic queries in relational data exchange. *Logical Methods in Computer Science*, 7(3):Paper 9, 2011. Special Issue for the 13th International Conference on Database Theory, ICDT 2010.

22 A. Hernich, L. Libkin, and N. Schweikardt. Closed world data exchange. *ACM Transactions on Database Systems*, 36(2):Article 14, 2011.

23 A. Hernich and N. Schweikardt. CWA-solutions for data exchange settings with target dependencies. In *Proceedings of the 26th ACM Symposium on Principles of Database Systems (PODS)*, pages 113–122, 2007.

24 A. Hernich and N. Schweikardt. Logic and data exchange: Which solutions are "good" solutions? In G. Bonanno, B. Löwe, and W. van der Hoek, editors, *Logic and the Foundations of Game and Decision Theory (LOFT 8)*, volume 6006 of *Lecture Notes in Computer Science*, pages 61–85. Springer-Verlag, 2010.

25 T. Imielinski. Incomplete deductive databases. *Annals of Mathematics and Artificial Intelligence*, 3(2–4):259–294, 1991.

26 T. Imielinski and W. Lipski, Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

27 P. G. Kolaitis. Schema mappings, data exchange, and metadata management. In *Proceedings of the 24th ACM Symposium on Principles of Database Systems (PODS)*, pages 61–75, 2005.

28 G. Lausen, M. Meier, and M. Schmidt. On chase termination beyond stratification. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1):970–981, 2009.

29 M. Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the 21th ACM Symposium on Principles of Database Systems (PODS)*, pages 229–240, 2002.

30 L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

31 L. Libkin. Data exchange and incomplete information. In *Proceedings of the 25th ACM Symposium on Principles of Database Systems (PODS)*, pages 60–69, 2006.

32 L. Libkin and C. Sirangelo. Data exchange and schema mappings in open and closed worlds. *Journal of Computer and System Sciences*, 77(3):542–571, 2011.

33 A. Mądry. Data exchange: On the complexity of answering queries with inequalities. *Information Processing Letters*, 94(6):253–257, 2005.

34 B. Marnette. Generalized schema mappings: From termination to tractability. In *Proceedings of the 28th ACM Symposium on Principles of Database Systems (PODS)*, pages 13–22, 2009.

35 J. Minker. On indefinite databases and the closed world assumption. In D. W. Loveland, editor, *Proceedings of the International Conference on Automated Deduction (CADE)*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308. Springer-Verlag, 1982.

36 R. Reiter. On closed world data bases. In H. Galaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.

37 R. van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

38 A. H. Yahya and L. J. Henschen. Deduction in non-horn databases. *Journal of Automated Reasoning*, 1(2):141–160, 1985.

Peer Data Management *

Armin Roth[1] and Sebastian Skritek[2]

1 **Cartesiusstrasse 47, 89075 Ulm, Germany**
`armin@arminroth.de`

2 **Technische Universität Wien**
Favoritenstraße 9, 1040 Wien, Austria
`skritek@dbai.tuwien.ac.at`

Abstract

Peer Data Management (PDM) deals with the management of structured data in unstructured peer-to-peer (P2P) networks. Each peer can store data locally and define relationships between its data and the data provided by other peers. Queries posed to any of the peers are then answered by also considering the information implied by those mappings.

The overall goal of PDM is to provide semantically well-founded integration and exchange of heterogeneous and distributed data sources. Unlike traditional data integration systems, peer data management systems (PDMSs) thereby allow for full autonomy of each member and need no central coordinator. The promise of such systems is to provide flexible data integration and exchange at low setup and maintenance costs.

However, building such systems raises many challenges. Beside the obvious scalability problem, choosing an appropriate semantics that can deal with arbitrary, even cyclic topologies, data inconsistencies, or updates while at the same time allowing for tractable reasoning has been an area of active research in the last decade. In this survey we provide an overview of the different approaches suggested in the literature to tackle these problems, focusing on appropriate semantics for query answering and data exchange rather than on implementation specific problems.

1998 ACM Subject Classification H.2.5 [Database Management]: Heterogeneous Databases, H.2.4 [Database Management]: Systems

Keywords and phrases Peer Data Management, PDM, Peer Data Management Systems, PDMS, Survey, P2P

Digital Object Identifier 10.4230/DFU.Vol5.10452.185

1 Introduction

Peer Data Management (PDM) on the one hand describes the complete area of data management in peer-to-peer (P2P) systems, while on the other hand it is also used to denote a very specific type of data management systems. In this survey, we follow the second interpretation, referring to management of structured data in unstructured P2P networks only. Concentrating on *Peer Data Management Systems (PDMSs)*, we provide a summary of different approaches introduced in the literature to design and create such systems, and consider both theoretical aspects as well as actual implementations:

PDMSs consist of a set of *peers*, where each peer offers some data through a so called *peer schema*. If a peer is interested in enhancing the information published through its peer schema with the data provided by some other peer, mappings between these two peers,

* S. Skritek was supported by the Vienna Science and Technology Fund (WWTF), projects ICT08-032 and ICT12-15, and by the Austrian Science Fund (FWF): P25207-N23.

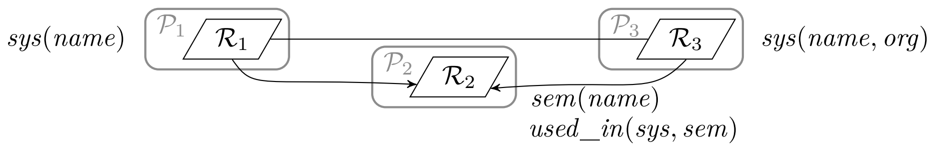


Figure 1 Example of a PDMS. Rectangles with rounded borders represent the peers P_1, P_2, P_3 and the trapezoids the peer schemas. Their schema is depicted next to the peers.

defined either on schema or instance level, are used to express the relationship between the data offered by these peers. Queries are posed against a single peer schema. Their answers do not only include the information stored locally at that peer, but also contain all the information implied by these mappings.

▶ **Example 1.** Figure 1 shows an example of a small PDMS. Consider first only the peers P_1 and P_2, both collecting and providing information about PDMSs. P_1 offers a list of prototype systems ($sys(name)$) and P_2 information about different semantics for PDMSs ($sem(name)$), and which prototype implements which semantics ($used\_in(sys, sem)$). Also, P_2 retrieves the information about available prototypes from P_1. Now assume another peer P_3 joins the group. It also offers a list of prototype systems, which contains beside the name of the system also the organization that built it ($sys(name, org)$). P_3 enhances its list by the data provided from P_1, and P_2 tries to complete its data by defining a mapping from P_3 to itself. ◀

Example 1 already illustrates some of the main advantages of PDMSs: They can be easily set up, members can join and leave the network at will, they support heterogeneous schemas and domains, and there is no need for global coordination, as e.g. required in typical data integration or multi-database systems. This allows each peer to take care of the mappings it is interested in only and no global coordination mechanism is required. However, the example also raises some questions. The probably most interesting one among them is how such mappings between two peers really look like and, related to this, how data sharing along them works. For example, P_3 could either import all relevant data from P_1, or retrieve the required information only at query time by answering queries not only on its local data, but also by forwarding them to P_1.

In fact, while PDMSs possess very promising properties, building such a system is a challenging task. One of the main problems is to find an appropriate formalism for defining the mappings between peers that is powerful enough to be useful, but at the same time allows for decidable (or preferably: efficient) reasoning (e.g., query answering). As a result, several different formalisms and semantics have been suggested in the literature for the specification of such mappings. Obviously, this leads to several different semantics that can be applied to a PDMS. Beside these problems of creating a suitable theory for PDM (like the relational model for single databases), due to the distribution and autonomy of the peers, also implementing such systems is no easy task. In combination with the lack of a clear semantics, several prototype systems have been created in the last years, addressing specific problems of building such systems.

In this survey we provide an overview of the different approaches taken so far to overcome these problems: We describe the most important and influential suggestions for useful (i.e. powerful, yet efficiently decidable) semantics for mappings in PDMSs, concentrating on relational systems, but also considering systems using other data models. One focus of this discussion is how schema mappings from data exchange or data integration have been applied to PDM. Beside those theoretical frameworks, we also provide an overview on existing prototype systems and point out specific and interesting properties of them. The organization of the paper is as follows: After some preliminary definitions in Section 2, we discuss general properties and characteristics of PDMSs in Section 3 and point out the differences between PDMSs, other P2P systems, and other kinds of distributed database systems. Next (Section 4), we describe the Local Relational Model, a concrete formalism for modeling PDMSs. Section 5 contains the discussion of the application of schema mappings to PDM, followed by alternative mapping strategies in Section 6. An overview on system prototypes is started in Section 7 which is completed in Section 8 with a discussion of PDMSs not applying the relational data model. We summarize and conclude in Section 9.

2 Preliminaries and System Model

Schemas and instances. A *relational schema* $\mathcal{R} = \{R_1, \dots, R_n\}$ is a set of relation symbols R_i each of a fixed arity k_i and with an assigned sequence of k_i attributes $(A_1, \dots, A_{k_i})$. Unless defined otherwise, an *instance* (or *interpretation*) I over a schema $\mathcal{R}$ consists of a k_i-ary relation R_i^I for each relation symbol $R_i \in \mathcal{R}$. We write $\vec{x}$ for a tuple $(x_1, \dots, x_n)$, but may also use $R_i(\vec{s}) \in I$ to denote a tuple $\vec{s} \in R_i^I$. By slight abuse of notation, we also refer to the set $\{x_1, \dots, x_n\}$ as $\vec{x}$. Hence, we may use expressions like $x_i \in \vec{x}$ or $\vec{x} \subseteq X$, etc. Tuples of the relations may contain two types of *terms*: *constants* and *labeled nulls*, taken from the sets *consts* and *null* respectively. Although (unless stated otherwise) the *domain* (or *universe*) $\mathit{dom} = \mathit{consts} \cup \mathit{null}$ is considered to be a countable infinite set, we only consider finite instances here, and denote with $\mathit{dom}(I) = \mathit{consts}(I) \cup \mathit{null}(I)$ the *active domain* of I.

Homomorphisms and conjunctive queries. Let I, J be instances. A *homomorphism* $h\colon I \to J$ is a mapping $\mathit{dom}(I) \to \mathit{dom}(J)$ s.t. (1) $h(c) = c$ for all $c \in \mathit{consts}(I)$ and (2) whenever $R(\vec{x}) \in I$, then $R(h(\vec{x})) \in J$, where by slight abuse of notation, for a tuple $\vec{x} = (x_1, \dots, x_n)$ we write $h(\vec{x})$ for $(h(x_1), \dots, h(x_n))$.

A *conjunctive query (CQ)* Q on a database schema $\mathcal{R}$ is of the form $Q\colon \mathit{ans}(\vec{x}) \leftarrow \exists \vec{y} \phi(\vec{x}, \vec{y})$, where $\phi(\vec{x}, \vec{y}) = \bigwedge_{i=1}^{n} r_i$ is a conjunction of atoms $r_i = R_j(\vec{z})$, s.t. $R_j \in \mathcal{R}$ with arity k, and $\vec{z} \subseteq \vec{x} \cup \vec{y}$ with $|\vec{z}| = k$. A tuple $\vec{s}$ is called an *answer* or *solution* to a CQ Q on an instance I if $\vec{s} = \mu(\vec{x})$, where $\mu\colon \vec{x} \cup \vec{y} \to \mathit{dom}(I)$ is a variable assignment on $\vec{x} \cup \vec{y}$, s.t. for every atom $R_i(\vec{z})$ occurring in Q it holds that $R_i(\mu(\vec{z})) \in I$.

Constraints, Mappings, and Theories. Given some logic L and a relational schema $\mathcal{R}$, a (L-)*theory over* $\mathcal{R}$ is a set of formulas of L over the relation symbols in $\mathcal{R}$. A formula that contains no free variables is called a *closed formula* or *sentence*. A *relational theory* over $\mathcal{R}$ consists of function free FO formulas over $\mathcal{R}$. Two special kinds of FO constraints are very well studied: A *tuple generating dependency (tgd)* is a FO formula $\forall \vec{x} \big(\exists \vec{z} \phi(\vec{x}, \vec{z}) \to \exists \vec{y} \psi(\vec{x}, \vec{y})\big)$, where ϕ and ψ are conjunctions of atoms. An *equality generating dependency (egd)* is a FO formula $\forall \vec{x} \big(\phi(\vec{x}) \to x_1 = x_2\big)$ where $x_1, x_2 \in \vec{x}$ and ϕ again is a conjunction of atoms.

Due to the structure of tgds, it is natural to identify a CQ with the lhs and rhs of a tgd each, i.e. to consider tgds as mappings $Q_1(\vec{x}) \to Q_2(\vec{x})$ for CQs Q_1, Q_2. Intuitively, a tgd then requests all answers to Q_1 also to be answers of Q_2. Further (based on this characterization) note that tgds can define GAV, LAV, and GLAV mappings, well known from data integration [54]. We therefore consider those mappings as special cases of tgds.

The *chase* is a well known procedure to repair instances that do not satisfy a set of tgds and egds by inserting tuples or unifying labeled nulls. For information on the chase, we refer to Chapter 1 of this book.

The *cyclicity* of a set of tgds (and mappings/constraints in general) is characterized via a graph representation of such sets, i.e. they are acyclic iff some corresponding directed graph is. Typically, for a set of mappings, the nodes of the graph are the relation symbols occurring in the mappings (or pairs of relation symbols and variables). Directed edges are added according to the structure of the mappings; for example for tgds from occurrences at the lhs of a tgd to occurrences on its rhs. Weaker notions of acyclicity do only forbid certain types of cycles. A prominent such example is *weak acyclicity* [20].

Peer Data Management Systems. We focus on the class of *Peer Data Management Systems* (PDMSs) that offer semantically well founded sharing of structured data. Following the terminology in [19], we consider a Peer Data Management System (PDMS) as a triple $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where $\mathcal{P} = \{P_1, \ldots, P_n\}$ is a set of autonomous *peers*, $\mathcal{R}$ is a set of *peer schemas* (one schema $\mathcal{R}_i$ for each $P_i \in \mathcal{P}$), and $\mathcal{M}$ is a set of *mappings*. We say that $\mathcal{R}_i$ is the schema of a local database if data is actually stored under $\mathcal{R}_i$. Alternatively, $\mathcal{R}_i$ can also be the global schema of some local data integration system at P_i (in this case, we refer to the source relations as $\mathcal{L}_i$ and assume the mappings between $\mathcal{L}_i$ and $\mathcal{R}_i$ for each $P_i \in \mathcal{P}$ to be contained in $\mathcal{M}$ as well), or a mediator schema (or view) defined by mappings from other peer schemas. Hence a peer may either bring new data into the system or just act as a mediator, restructuring (through corresponding mappings) data already present in the system. For the semantics of a peer, this does not make any difference, as in any case each peer offers some well defined data through its peer schema. Finally, the mappings in $\mathcal{M}$ may be either defined on schema or instance level, or both. As we will see, this depends on the concrete formalism used for defining the mappings. We use dom_i to denote the domain of P_i.

Instances, consistent global instances, and queries for PDMSs. Given a PDMS $\mathcal{S}$, an instance I for $\mathcal{S}$ is either just an instance for $\mathcal{R}$ (if each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database) or an instance for $\bigcup_{P_i \in \mathcal{P}} \mathcal{L}_i$ if the peers consist of local data integration systems. For an instance I for $\mathcal{S}$, let $I|P_i$ denote the restriction of I to the schema $\mathcal{R}_i$ (resp. $\mathcal{L}_i$). Given an instance I for $\mathcal{S}$, a *consistent global instance I' for $\mathcal{S}$ w.r.t. I* is in general an instance of $\mathcal{R}$, s.t. (i) I' satisfies $\mathcal{M}$ and (ii) if I is already an instance for $\mathcal{R}$, then there exists a homomorphism $h\colon I \to I'$. Thereby the notion "I' satisfies $\mathcal{M}$" depends on the concrete semantics applied, and will be a main focus of this survey. We denote with $\mathcal{I}_I$ the set of all consistent global instances w.r.t. I, and drop the I if clear from the context. However, some systems do not define a global instance for $\mathcal{S}$ as an instance for $\mathcal{R}$, but as a tuple $(\mathcal{I}_I[\mathcal{R}_i])_{\mathcal{R}_i \in \mathcal{R}}$, where each $\mathcal{I}_I[\mathcal{R}_i]$ is a *set* of instances for $\mathcal{R}_i$.

A query to a PDMS $\mathcal{S}$ is formulated over a single peer schema $\mathcal{R}_i \in \mathcal{R}$. Given an instance I for $\mathcal{S}$, the result of a query Q usually is defined as the *certain answers* $Q^c(I) = \bigcap_{I' \in \mathcal{I}_I} Q(I')$, where $Q(I')$ denotes the evaluation of Q over I'. If only $\mathcal{I}_I[\mathcal{R}_i]$ is defined, then $Q^c(I) = \bigcap_{I' \in \mathcal{I}_I[\mathcal{R}_i]} Q(I')$. Unless stated otherwise, we always assume Q to be a CQ.

3 PDMS: Characterization and Properties

In this section, we start with a quick overview over a classification of P2P systems and discuss how PDM fits into these concepts. We then relate PDMSs to traditional database systems, and finally take a look onto properties specific for PDMSs.

P2P systems are nowadays wide spread and used for a variety of applications in many different areas. Tasks solved by P2P systems cover, among others, sharing of data (e.g., Gnutella, BitTorrent) and other resources, communication (e.g. Skype), or the implementation of fail-safe systems. The reasons for the success of P2P systems include their scalability, low setup costs, the lack of need for central coordination, or a high reliability (since replicating and shifting resources and tasks between different nodes allows to compensate node failures).

A prominent way to classify P2P systems is according to the logical structure of the resulting P2P network (overlay network). In *pure* P2P systems, all peers are conceptually equal and have the same role, while in *hybrid* systems so called super-peers (which have more knowledge about the current state of the system) act as servers and control and coordinate the system. In the extreme case of *centralized P2P systems*, all requests are issued against a single central server who dispatches them to an appropriate network node. Pure P2P systems are further divided into *unstructured* and *structured* systems. In unstructured systems, each peer is free to choose which data to store, which peers to communicate with, or which requests to accept. In *structured* P2P systems, data placement or message routing follows strict rules determined by the system, which are enforced by the peers in a distributed manner. A prominent example for structured P2P systems are distributed hash tables (DHTs), where the placement of a data item is determined by a key or hash value assigned to each item: Each peer gets assigned a (not necessarily distinct) part of the keyspace, and items are stored at exactly those peers that cover their key values. The reason that we will concentrate on the management of structured data in unstructured P2P systems is that while most of the aspects and applications of P2P systems described above are already covered by surveys or books, to the best of our knowledge, a summary of Peer Data Management Systems is missing: We refer to [73] for a general introduction to – and overview on – P2P systems, including characterizations, architectures, applications and systems as well as aspects like routing, load balancing, security or trust. [69] provides an extensive summary of applications for P2P systems and general P2P techniques, including three chapters on DHTs. A specific overview over data management P2P systems can be found in [5, Chapter 16] with a focus on query evaluation and replica management. A short classification of P2P data management systems based on their structure is presented in [11]. Finally [68] provides an introduction into the combination of P2P and Semantic Web techniques and applications. However, the only overview paper in the area of PDMSs is [41], reviewing design and implementation aspects and challenges for PDMSs, but not providing a comprehensive summary of existing approaches. It can therefore be considered as a complement to the present paper.

One of the initial goals of PDMSs was to extend the idea of unstructured P2P file sharing systems to structured data [32], i.e. to allow for data sharing between a large number of highly autonomous participants, but supporting a rich semantics and expressive query languages (see [32, 6] for early visions of this idea). Note that PDMSs are not primarily intended to provide a distributed, fail safe storage system (like DHTs) or to provide load distribution, but just to provide an easy way to allow participants to share their data with others. Each peer can freely select its neighbors and define mappings to them. Data is shared according to the semantics of the mappings only, such that each peer has full control over the data it stores. Following these ideas, PDMSs obviously belong to the group of unstructured P2P-systems.

Compared to other distributed database systems like multi-databases [12], which allow for a similar distribution and heterogeneity of their members, PDMSs provide a higher autonomy for each peer. A classification and description of PDMSs w.r.t. traditional database approaches is given in [11]. Higher autonomy of each member in the network also distinguishes PDMSs from traditional data integration systems [54]. There, the data provided by each peer is integrated in terms of a global schema instead of pairwise mappings.

While the above discussion holds for all PDMSs, we next take a look onto properties that characterize and distinguish different approaches to PDM and shortly summarize the most important aspects of PDM that influence those properties. We start with considering design choices that influence the semantics of a PDMS, continue with characteristics resulting from these choices and finish with choices that deal with the concrete implementation but do not influence the semantics. Obviously a fundamental property of a PDMS with heavy influence on its semantics is the supported data model (e.g., relational, XML, RDF). Another important aspect are the P2P mappings, as they define what data is exchanged and how. The exchange could be either done by enforcing the constraints defined through the mappings on the data, i.e. by indeed materializing tuples in the different peers such that the mappings are satisfied over the stored instances (we refer to corresponding systems as *exchange systems*). In general there exist several (up to infinitely many) possible instances that could be materialized to satisfy the mappings. In this case, the goal is to identify some "best" instance to materialize. Most of the time, this means to materialize some most general instance that only contains information indeed implied by the mappings. Another desired property of the chosen instance is to allow to compute the certain answers for some query w.r.t. all these possible instances from the information in the instance only. Another possibility is not to materialize these extra information, but to use the mappings to infer additional information only for query answering at query time (*integration systems*), or to consider mappings only as rules for how to translate updates made at one peer to an update on another peer and to exchange only updates. Further aspects of mappings are the mapping language (e.g., FOL or restrictions thereof, coordination formulas — cf. Section 4, mapping tables — cf. Section 6.1), the semantics under which they are evaluated (e.g., local reasoning or global reasoning for FOL mappings — cf. Section 5.1), whether they are defined e.g. on schema or instance level, and whether they support different domains for each peer or assume a shared domain among all peers. The supported query language is another interesting property of a PDMS, just like the technique used for query answering. For example, the latter could be based on query rewriting, answering queries using views, temporarily materialization of data (universal solutions), or using logic programs (e.g., under the stable model semantics). Further aspects are the incorporation of trust, the capability to deal with inconsistencies in the data in a meaningful way, or the maintenance and use of data quality metrics.

Choices on the above properties have a direct influence on properties like the concrete degree of the autonomy, modularity or heterogeneity of the peers or the decidability and complexity of query answering. Finally, further properties of PDMSs arise from aspects related directly with the implementation of such systems. Those include the query planning algorithm (which can be centralized or distributed), the incorporation of query optimization (including relaxations on the correctness or completeness of answers), the maintenance of indices and/or replica, or optimizations of the inter peer mappings.

For a throughout discussion of these aspects related to PDM as well as pointers to different solutions for them, we refer to the recent survey in [41]. Further discussions of these topics can be found e.g. in [73, Chapter 4] and [5, Chapter 16]. In this paper, we provide a summary of different approaches to PDM and PDMSs. Thereby the main focus will be on the way the P2P mappings are defined and formalized, as they are the central component of PDMSs. Beside describing these approaches, we will use (some of) the properties listed above to discuss the effects of different approaches and to characterize them, but the main focus lies on their descriptions. We will further give an overview of prototype implementations of PDMSs, pointing out interesting or notable design decisions or specifics of these systems.

4 A Model for PDM: The Local Relational Model

After the rather general last section, we now start to take a look onto concrete approaches taken to formalize and implement PDMSs. One such proposal was the *Local Relational Model (LRM)* [6, 67], one of the first models proposed for PDMSs. Because it addresses several problems of PDM we use it to illustrate several different key concepts for PDMSs: In the LRM, dependencies between the different databases can be expressed by mappings on schema level. Peers are allowed to use different domains by providing a domain translation mechanism with a meaningful, well-defined semantic. Finally, queries can exploit all these mechanisms to incorporate information stored at several peers into the answer.

Because the LRM is a powerful mechanism addressing several problems of PDM, we will use it as a reference model and compare other approaches with it, or – if appropriate – will even describe other approaches in terms of the LRM.

▶ **Definition 2.** A LRM-PDMS is a PDMS $\mathcal{S}_{LRM} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database and $\mathcal{M} = \mathcal{M}_T \cup \mathcal{M}_{CF} \cup \mathcal{M}_R$, s.t. $\mathcal{M}_T$ contains for each $P_i \in \mathcal{P}$ a relational theory T_i over $\mathcal{R}_i \in \mathcal{R}$, $\mathcal{M}_{CF}$ is a set of *coordination formulas*, and $\mathcal{M}_R$ is a set of *domain relations*.

Domain relations $r_{ij} \subseteq dom_i \times dom_j$ provide domain translations between pairs (P_i, P_j) of peers in $\mathcal{P}$. They allow to support different domains dom_i (which, unlike in most other formalisms, are assumed to be finite) at each $P_i \in \mathcal{P}$. Domain relations need not be symmetric. In the following let $\mathcal{C} = \{i \mid P_i \in \mathcal{P}\}$. A *coordination formula (CF)* is an expression of the form $CF ::= i : \phi \mid CF \rightarrow CF \mid CF \wedge CF \mid CF \vee CF \mid \exists i : x(CF) \mid \forall i : x(CF)$, where ϕ is a function free FO formula over some $\mathcal{R}_i \in \mathcal{R}$ and $i \in \mathcal{C}$. Their intuition is as follows: $\forall i{:}x(CF)$ denotes that the universal quantification of x is over the domain dom_i (similar for $\exists$), while $i{:}\phi$ indicates that variables in ϕ shall be evaluated w.r.t. P_i (i.e. under dom_i). For variables occurring free in ϕ this may require to incorporate domain translation: If such a free variable is bound outside $i{:}\phi$ by a quantifier under a different context P_j (i.e. bound under domain dom_j), the translation from dom_j to dom_i must be considered for evaluation (see below for a formal definition of the semantic of such a translation). Although not required in [67], here we consider all CFs in $\mathcal{M}$ to be closed. Recall that a relational theory T_i over $\mathcal{R}_i$ consists of function free FO sentences over $\mathcal{R}_i$ (i.e. function free FO constraints over $\mathcal{R}_i$) with constants from dom_i. Obviously, each theory T_i can be easily encoded as a set $\{i{:}\phi \mid \phi \in T_i\}$ of CFs. Further, in [67] it was shown that also domain relations may be expressed as CFs. Hence $\mathcal{M}$ may be considered to contain coordination formulas only.

▶ **Example 3.** The following coordination formula adapted from [67] creates a record in the *Person* relation of peer *Hospital* based on the data in *Patient* at peer *Doc*:

$$\forall(Doc : fName, lName, gender).(Doc : Patient(SSN, fName, lName, gender) \rightarrow Hospital : \exists(persID, name, age).Person(persID, SSN, name, gender, age, Doc) \wedge name = concat(fName, lName))).$$

A unique value for $persID$ has to be generated and the unknown age has to be filled with a so-called Skolem constant. ◀

In the LRM, an instance $\mathcal{I}$ for $\mathcal{S}$ is not defined as an instance for $\mathcal{R}$, but $\mathcal{I} = (\mathcal{I}[\mathcal{R}_i])_{i \in \mathcal{C}}$ is a tuple of databases $\mathcal{I}[\mathcal{R}_i]$ for each $\mathcal{R}_i \in \mathcal{R}$. Each such database $\mathcal{I}[R_i]$ in turn is considered to consist of a set of instances I' of $\mathcal{R}_i$ that satisfy T_i while interpreting constants as themselves. Thereby the idea is that while $|\mathcal{I}[\mathcal{R}_i]| = 1$ describes the case of a traditional database instance,

$|\mathcal{I}[\mathcal{R}_i]| = 0$ indicates an inconsistent database at peer P_i and $|\mathcal{I}[\mathcal{R}_i]| > 1$ models incomplete databases. Next we define satisfiability of CFs. To be able to deal with domain translations, in the LRM a variable assignment μ for a set $\vec{x}$ of variables is a *set* of mappings $\mu = \{\mu_i\}_{i \in \mathcal{C}}$ with $\mu_i \colon \vec{x} \to dom_i$. For $i \in \mathcal{C}$ and $J \subset \mathcal{C}$, μ is an i-to-J-assignment (i-from-J-assignment) of a variable x if for all $j \in J$ with $j \neq i$, $(\mu_i(x), \mu_j(x)) \in r_{ij}$ ($(\mu_j(x), \mu_i(x)) \in r_{ji}$, resp.), i.e. if μ sticks to the domain translations defined by the domain relations between dom_i and dom_j for all $j \in J$. Now given $\mathcal{I}$, $\mathcal{M}_R$, and an assignment μ, a CF $i : \phi$ is satisfied by $(\mathcal{I}, \mathcal{M}_R)$ under μ (denoted $(\mathcal{I}, \mathcal{M}_R) \models i\!:\!\phi[\mu]$) if for each $I' \in \mathcal{I}[\mathcal{R}_i]$ it holds $I' \models \phi[\mu_i]$. I.e. $i : \phi$ is satisfied if ϕ, interpreted under the scope of P_i (in terms of the assignment μ_i), is satisfied over all instances in $\mathcal{I}[\mathcal{R}_i]$. For CFs $\forall i\!:\!x(A)[\mu]$ ($\exists i\!:\!x(A)[\mu]$) on the other hand let $J \subset \mathcal{C}$ contain all $j \in \mathcal{C}$ s.t. x occurs free in a subformula $j : \phi$ of A. The idea is, that while x is quantified over dom_i, it is evaluated in A under the scopes of the peers P_j ($j \in J$), i.e. under the domains dom_j. Hence when evaluating A, for every possible value for x over dom_i, the corresponding domain translations must be taken into account. Therefore $(\mathcal{I}, \mathcal{M}_R) \models \forall i : x(A)[\mu]$ if $(\mathcal{I}, \mathcal{M}_R) \models A[\mu']$ for all i-to-J-assignments μ' on x that differ from μ only on x. Further, $(\mathcal{I}, \mathcal{M}_R) \models \exists i\!:\!x(A)[\mu]$ if $(\mathcal{I}, \mathcal{M}_R) \models A[\mu']$ for some i-from-J-assignments μ' on x that differ from μ only on x. Intuitively, in case of universal quantification, for each possible assignment μ_i on x, each subformula $j\!:\!\phi$ must be satisfied under each translation of $\mu_i(x)$ to dom_j. The case for the existential quantification is similar, but a little bit more involved (note that in i-from-J-assignments we have r_{ji} instead of r_{ij}). Due to space restrictions, for a discussion of this we have to refer to [67]. Satisfaction of the connectives $\to, \wedge, \vee$ is defined as usual.

Queries Q against some $P_i \in \mathcal{P}$ are of the form $(i : q(\vec{x})) \leftarrow A(\vec{x})$, where $A(\vec{x})$ is a coordination formula with free variables $\vec{x}$, $|\vec{x}| = n$, and q is a new n-ary relation symbol. Given $(\mathcal{I}, \mathcal{M}_R)$, the answer to such a query is defined as $\{\vec{d} \in dom_i^n \mid (\mathcal{I}, \mathcal{M}_R) \models \exists i : \vec{x}(A(\vec{x}) \wedge i\!:\!(\vec{x} = \vec{d}))\}$. Queries can be defined recursively, i.e. $A(\vec{x})$ may use the result q' of another query Q' (these recursions may be cyclic).

Note that while this defines how a query is evaluated over $(\mathcal{I}, \mathcal{M}_R)$, $\mathcal{M}_{CF}$ is not taken into account for query answering. Originally, in the LRM P2P mappings were not considered for information exchange or reasoning, but just to express constraints between peers. For a query to also include data stored at different peers, the corresponding CFs need to be specified explicitly as part of the (recursive) query. As a result, there is no concept like consistent global instances (w.r.t. some instance $\mathcal{I}$ for $\mathcal{S}$). Also, while all $I' \in \mathcal{I}[\mathcal{R}_i]$ must satisfy T_i, it is not required that $\mathcal{I}[\mathcal{R}_i]$ contains indeed all models of T_i. So, given a pair $(\mathcal{I}, \mathcal{M}_R)$ as input, the LRM only defines if a CF is satisfied by $(\mathcal{I}, \mathcal{M}_R)$ and the result of a query over this instance. If $\mathcal{I}$ contains incomplete databases, then this answer is however certain w.r.t. this incompleteness (but not w.r.t. to possible repairs). Restricting to a certain class of CFs, [22] extends the LRM by a notion of certain answers also taking $\mathcal{M}_{CF}$ into account. Therefore it is assumed that some input instance I is already encoded into $\bigcup_{i \in C} T_i$, i.e. the tuples in I are expressed as part of the theory T_i. Basically [22] then defines the notion of global consistent instances by considering $\mathcal{I}[\mathcal{R}_i]$ to consist of all interpretations of the theory T_i (i.e. instances of $\mathcal{R}_i$ that satisfy T_i). This is then used to define the set of certain answers w.r.t. I as those answers that occur in each such instance $I' \in \mathcal{I}[\mathcal{R}_i]$. Obviously, this idea could be extended to the LRM in general. I.e. given I encoded into $\mathcal{M}$, the certain answers to a query would be $\bigcap_{\{\mathcal{I} \mid (\mathcal{I}, \mathcal{M}_R) \models \mathcal{M}\}} \{\vec{d} \in dom_i^n \mid (\mathcal{I}, \mathcal{M}_R) \models \exists i\!:\!\vec{x}(A(\vec{x}) \wedge i\!:\!(\vec{x} = \vec{d}))\}$.

5 Schema Mappings for PDMSs

We have seen that the LRM provides mappings on schema and instance level and supports different domains at each peer. Many approaches to PDM however assume a unique domain shared by all peers, and consider schema level mappings only. They can be generally described as PDMSs $\mathcal{S}_S = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where $\mathcal{M}$ is a set of formulas of some logic $\mathcal{L}$ over $\mathcal{R}$. In this section, we will show how such settings can be used to express various semantics for PDMSs.

The characterization of $\mathcal{S}_S$ is reminiscent of *schema mappings* as considered for example in data exchange [46]. And in fact, like for data exchange, most schema level mapping based PDM settings use tgds (or slight variations thereof) to define P2P mappings, and sets of tgds and egds on single peer schemas $\mathcal{R}_i \in \mathcal{R}$ to define local constraints. Unfortunately, just applying the typical semantics of schema mappings to PDMSs is no satisfying solution, as reasoning becomes undecidable and the structure of the system cannot be modeled appropriately, leading to a loss of peer autonomy. As this raised a lot of work on identifying suitable semantics that on the one hand resolve these problems and on the other hand support a wider range of applications, we devote this section to the discussion of these suggestions.

5.1 Global and Local Reasoning

A major distinction between such PDMSs $\mathcal{S}_S$ is made according to whether $\mathcal{M}$ is interpreted under *global* or *local* reasoning[1]. Global reasoning means that $\mathcal{M}$ is interpreted as a single (global) FO theory. This is the semantics obtained by extending data exchange [46] and data integration [54] scenarios to P2P settings. Under local reasoning, each peer is modeled as a distinct (local) theory, and inter-peer mappings are interpreted as to exchange certain facts between such theories only. It was explicitly suggested in [14, 16], and occurs implicitly in the LRM on implicational coordination formulas.

It is easy to see that under global reasoning, deriving all certain answers may become an undecidable problem, due to the network topology (e.g., P2P mappings could form not weakly acyclic sets of tgds [20], or in combination with local egds could form sets of 1-key conflicting inclusion dependencies [13]). Global reasoning further does not completely reflect the modularity of PDMSs [16]. On the other hand, it allows to derive more information than local reasoning. The latter two properties are illustrated in the following example.

▶ **Example 4** (cf. [22]). Assume a PDMS $\mathcal{S}_S = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, with $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$, $\mathcal{R}_1 = \{C\}$, $\mathcal{R}_2 = \{M, F\}$, $\mathcal{R}_3 = \{TP\}$, (let C stand for *Citizen*, F for *Female*, M for *Male*, TP for *TaxPayer*, and let all be unary) and $\mathcal{M} = \{C(x) \to M(x) \vee F(x), M(x) \to TP(x), F(x) \to TP(x)\}$[2], and queries $Q_1 \colon q(x) \leftarrow TP(x)$, $Q_2 \colon q(x) \leftarrow M(x)$, and $Q_3 \colon q(x) \leftarrow F(x)$.

For an instance $I = \{C(alice)\}$, $Q_2^c(I) = Q_3^c(I) = \emptyset$ (under both kinds of reasoning), as $\mathcal{I}_I$ contains two instances not containing $F(alice)$ and $M(alice)$ respectively. Under global reasoning however, $Q_1^c(I) = \{q(alice)\}$, as $TP(alice)$ is derivable in any $I' \in \mathcal{I}_I$. While entailing a maximal amount of information, this contradicts modularity in a way as mappings not only transfer the "visible" content of a peer, but the information exchanged depends on the complete structure of the network (hence it is to some extend unpredictable for a single peer). Under local reasoning on the other hand $Q_1^c(I) = \emptyset$, as mappings only exchange information present in every $I' \in \mathcal{I}_I$, which is neither the case for $F(alice)$ nor $M(alice)$. ◀

[1] Do not confuse the notions of global and local reasoning here with the global and local semantics in [22]. The (equivalent) latter two notions are concrete formalizations of local reasoning.

[2] We use disjunctive tgds for the sake of illustration only. Similar effects occur with ordinary tgds as well.

Note that mappings defined over a single peer schema behave equivalent under both kinds of reasoning. We next take a closer look onto these two formalisms and two concrete examples.

5.1.1 Global Reasoning & $\mathcal{PPL}$

If we consider a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where $\mathcal{M}$ consists of FO formulas, the typical semantic of $\mathcal{S}$ under global reasoning is defined in terms of a FO theory T that contains all formulas in $\mathcal{M}$. Given an instance I for $\mathcal{S}$ (i.e. either for $\mathcal{R}$ or for the source relations $\mathcal{L}_i$), $\mathcal{I}_I$ is defined by all models of T that agree with I. As already noted, considering inter-peer tgds and local tgds and egds leads to settings where query answering becomes undecidable (cf. [20, 16, 38]) as it requires to derive all information implied by T. The only ways to overcome this are either to drastically restrict the allowed mapping language or to restrict the structure of T, and therefore the topology of the inter-peer mappings. While the first choice often requires weak mapping languages that render the complete system useless, the latter reduces the autonomy of the peers, as they are no longer free to define mappings to neighbors arbitrarily. Hence, one has to find a trade-off between these two possibilities.

One of the first and most prominent examples for a PDMSs dealing with this trade-off is the Piazza PDMS [35, 36, 37, 38, 71]. Mappings in Piazza are defined using the *Peer Programming Language ($\mathcal{PPL}$)*, introduced for this purpose in [37, 38].

▶ **Definition 5.** A $\mathcal{PPL}$-PDMS $\mathcal{S}_{\mathcal{PPL}} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local data integration system (where possibly $\mathcal{L}_i = \emptyset$), and $\mathcal{M} = \mathcal{M}_L \cup \mathcal{M}_P$ is a set of mappings defined in $\mathcal{PPL}$. Thereby $\mathcal{M}_L$ contains the mappings between all $\mathcal{L}_i$ and $\mathcal{R}_i$, while $\mathcal{M}_P$ contains the mappings and constraints on $\mathcal{R}$.

Due to space restrictions, we can only give a short overview on $\mathcal{PPL}$. $\mathcal{PPL}$ distinguishes two general kinds of mappings, *storage descriptions* (used to define $\mathcal{M}_L$) and *peer mappings* (for $\mathcal{M}_P$). Storage descriptions are either exact or sound LAV mappings between $\mathcal{L}_i \in \mathcal{L}$ and $\mathcal{R}_i \in \mathcal{R}$. Peer mappings are either exact or sound GLAV mappings or certain GAV-style mappings (*definitional mappings*; defined as datalog rules with a single atom in the head and a CQ in the body) between two (not necessarily distinct) peer schemas $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}$. ([37, 38] allow all mappings in $\mathcal{M}$ to refer to arbitrary relations in $\mathcal{R}$. However, by introducing additional relations and mediator peers, the above definition is equally expressive, but more modular.) Given an instance I for $\mathcal{L}$, an instance I' for $\mathcal{R}$ satisfies all LAV and GLAV mappings according to the usual semantics (cf. [54]). The definitional mappings are satisfied by I' if for each relation R occurring in the head of such a mapping, $R^{I'} = \bigcup_{i=1}^{n} Q_i(I')$, where the Q_i are the bodies of the n definitional mappings where R is the head predicate symbol. Given I, the goal is not to materialize any data under $\mathcal{R}$, but at query time to return the certain answers w.r.t. all instances I' satisfying $\mathcal{M}$ that are equal to I on $\mathcal{L}$.

The expressive power of $\mathcal{PPL}$ requires to constrain the topology of mappings in $\mathcal{M}$ in order to allow for decidable query answering. Following the notion of acyclicity defined in Section 2, we obtain the following results.

▶ **Theorem 6** ([38]). *Given a $\mathcal{PPL}$-PDMS $\mathcal{S}_{\mathcal{PPL}}$, a CQ Q, and an instance I for $\mathcal{S}_{\mathcal{PPL}}$, computing $Q^c(I)$ is undecidable. If $\mathcal{M}$ contains only sound LAV and sound GLAV mappings, and $\mathcal{M}$ is acyclic, then computing $Q^c(I)$ is in polynomial time (data complexity).*

[38] also presented an algorithm that runs in polynomial time (data complexity) and computes certain answers w.r.t. a $\mathcal{PPL}$-mapping. While it is guaranteed to always return only certain answers, it also returns all of them for acyclic mappings. One key observation for this algorithm is that each GLAV mappings can be split into one LAV and one GAV mapping.

The algorithm then creates a rule-goal tree, by interleaving steps of query unfolding (for GAV style mappings) and answering queries using views [34] for LAV style mappings. At the end, the query is rewritten in terms of the local schemas $\mathcal{L}_i$.

Although the border of decidability and tractability can be pushed further by resorting to a little bit less restrictive constraints on $\mathcal{M}$ that still allow for decidable (tractable) query answering (see [38]), the general problem of using global constraints remains.

5.1.2 Local Reasoning: Exchanging Certain Answers

Local reasoning applies an interpretation to inter-peer tgds that allows for decidable query answering whenever query answering over each isolated peer can be decided. The basic idea is that P2P mappings are not satisfied by a single consistent global instance, but by the set of all such instances. Intuitively, they only exchange certain answers. This idea has been formalized in several (equivalent) ways: It was explicitly introduced in [14] by using the modal logic KT45 (by modeling a tgd as sentence $\forall\vec{x}\big(\mathbf{K}(\exists\vec{z}\phi(\vec{x},\vec{z})) \rightarrow \exists\vec{y}\psi(\vec{x},\vec{y})\big)$, where $\mathbf{K}$ is a modal operator expressing certainty) together with a distributed algorithm for query answering, and shown to support modularity better than global reasoning in [16]. In [22], two further formalisms for defining the semantics of a PDMS were introduced. Based on ideas of the LRM they consider a restricted form of coordination formulas and were shown to be equivalent with the formalism in [14]. Finally, in [27] local reasoning was formalized directly via sets of global consistent instances. Here, we follow this approach: Assume a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ with $\mathcal{M} = \mathcal{C}_M \cup \mathcal{M}_M$, where $\mathcal{C}_M = \{\mathcal{C}_i \mid P_i \in \mathcal{P}\}$ are formulas of some logic $\mathcal{L}$ defined over single peer schemas (including, if present, the mappings between $\mathcal{L}_i$ and $\mathcal{R}_i$), and $\mathcal{M}_M$ is a set of inter-peer tgds. Satisfiability of $\mathcal{C}_M$ is still defined for single instances: Given an instance I for $\mathcal{R}$, some instance I' satisfies $\mathcal{C}_M$ w.r.t. I if, for every $\mathcal{R}_i \in \mathcal{R}$, it satisfies the logical theory containing $\mathcal{C}_i$ and the facts in I. For $\mathcal{M}_M$ on the other hand, being satisfied is defined for *sets* of instances: Given an instance I, a set $\hat{\mathcal{I}}$ of ground instances I' for $\mathcal{R}$ satisfies $\mathcal{M}$ w.r.t. I if (i) each $I' \in \hat{\mathcal{I}}$ satisfies $\mathcal{C}_M$ w.r.t. I, and (ii) for each tgd $\tau \in \mathcal{M}_M$ with $\tau = \forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}\psi(\vec{x},\vec{y}))$, it holds that $\bigcap_{I' \in \hat{\mathcal{I}}} Q_\phi(I') \subseteq \bigcap_{I' \in \hat{\mathcal{I}}} Q_\psi(I')$, where Q_ϕ, Q_ψ are the CQs associated to the lhs and rhs of τ, respectively. I.e., instead of testing for each instance $I' \in \hat{\mathcal{I}}$ if all answers to Q_ϕ are also answers to Q_ψ, under local reasoning the certain answers to Q_ϕ w.r.t. $\hat{\mathcal{I}}$ must be contained in the certain answers to Q_ψ.

Hence, given some instance I the semantics of $\mathcal{S}$ (i.e. the information implied by $\mathcal{M}$) depends on the set $\hat{\mathcal{I}}$. Since there may exist more than one possible set to choose from, the question is which is the "right" one. However, note that if two distinct sets $\hat{\mathcal{I}}$ and $\hat{\mathcal{I}}'$ satisfy $\mathcal{M}$, so does $\hat{\mathcal{I}} \cup \hat{\mathcal{I}}'$. Hence there exists a unique maximal set that satisfies $\mathcal{M}$. The set $\mathcal{I}_I$ is thus defined as this maximal set, and mappings in $\mathcal{M}_M$ are interpreted w.r.t. $\mathcal{I}_I$. I.e., the data implied by $\mathcal{M}_M$ is defined as those information shared by all instances $I' \in \mathcal{I}_I$. Given some instance I, all information implied by $\mathcal{M}_M$ can be efficiently (data complexity) materialized using a variant of the chase. Recall that the chase "repairs" violations of tgds (witnessed by tuples $\vec{t} \in Q_\phi(I) \setminus Q_\psi(I)$) by adding tuples into I s.t. $\vec{t}$ is also an answer to Q_ψ in the resulting instance. Now while the traditional chase considers all tuples $\vec{t}$, for local reasoning it suffices to only consider tuples that contain no labeled nulls. Beside efficient reasoning, this procedure thus provides also an alternative, procedural description of the semantics of local reasoning: tgds (an thus, peers) exchange only ground tuples.

One of the most elaborated frameworks based on this semantics is the PDEI-system also defined in [27], which we discuss next and use to illustrate the above definitions.

▶ **Definition 7.** A PDEI-System (Peer Data Exchange and Integration) $\mathcal{S}_{PDEI} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M} = \mathcal{C}_E \cup \mathcal{C}_I \cup \mathcal{M}_E \cup \mathcal{M}_I$, where $\mathcal{C}_E$ and $\mathcal{C}_I$ are sets of tgds and egds defined over single peer schemas $\mathcal{R}_i \in \mathcal{R}$, and $\mathcal{M}_E$ and $\mathcal{M}_I$ are sets of inter-peer tgds.

According to the semantics described above, $\mathcal{C}_E$ and $\mathcal{C}_I$ are interpreted as FO-theories over $\mathcal{R}$, while $\mathcal{M}_E$ and $\mathcal{M}_I$ only need to be satisfied w.r.t. tuples that are present in all $I' \in \mathcal{I}_I$. In addition, PDEI-systems allow for both materialization of data and "virtual" exchange: The mappings in $\mathcal{C}_E \cup \mathcal{M}_E$ are enforced on the instance on $\mathcal{R}$, i.e. tuples are materialized to satisfy them, while those in $\mathcal{C}_I \cup \mathcal{M}_I$ are only considered for query answering (if some tuple can be derived by both kinds of mappings, it needs not to be materialized). Hence reasoning over a PDEI-system is twofold: Given some instance I for $\mathcal{R}$, the goal is on the one hand to materialize an instance over $\mathcal{R}$ that satisfies $\mathcal{C}_E \cup \mathcal{M}_E$ ("admissible instance"), and on the other hand to answer queries over such an instance by returning certain answers also w.r.t. $\mathcal{C}_I \cup \mathcal{M}_I$. For these tasks to be decidable, a PDEI-system $\mathcal{S}$ has to be *stratified*, i.e. $\mathcal{C}_E$ is weakly acyclic, $\mathcal{C}_I$ consists of legal key constraints and foreign key dependencies (FK) only, and no head of a FK appears in the lhs of any tgd in $\mathcal{C}_E$. Note that these are local constraints only, verifiable by each peer $P_i \in \mathcal{P}$ in separation. For such systems, computing admissible instances and query answering can be done by combining a variant of the chase that considers the special semantics of inter-peer tgds (to retrieve information implied by $\mathcal{C}_E \cup \mathcal{M}_E \cup \mathcal{M}_I$) with query rewriting (to access information implied by $\mathcal{C}_I$).

▶ **Theorem 8** ([27]). *Let $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ be a PDEI-System, and I an instance for $\mathcal{R}$. If $\mathcal{S}$ is stratified, then an admissible state I' for $\mathcal{R}$ and $Q^c(I')$ for a CQ Q can be computed in polynomial time (data complexity).*

5.1.3 Schema Mappings and the LRM

We shortly comment on the relationship between global and local reasoning over schema mappings and the LRM by discussing the translation of some schema mapping based PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ into a LRM-PDMS. Assuming a shared domain *dom* between all peers (i.e. $r_{i,j} = \{(a,a) \mid a \in dom\}$ for all $P_i, P_j \in \mathcal{P}$), we can neglect the effect of the domain relations.

For global reasoning, assume that $\mathcal{M}$ is a set of function free FO formulas over $\mathcal{R}$. This corresponds to a LRM-PDMS $\hat{\mathcal{S}} = (\{\hat{P}_1\}, \{\hat{\mathcal{R}}_1\}, \hat{\mathcal{M}})$, where $\hat{\mathcal{M}} = \{1{:}\phi \mid \phi \in \mathcal{M}\}$. This also illustrates nicely how the structure of the PDMS is lost under global reasoning. Given an instance I for $\mathcal{S}$, $\mathcal{I}_I$ w.r.t. $\mathcal{S}$ contains exactly those instances I' for $\hat{\mathcal{S}}$ that satisfy $\hat{\mathcal{M}}$ and s.t. there is a homomorphism from I into I'.

We already mentioned that when considering certain implicational coordination formulas, the LRM exhibits exactly the semantics of local reasoning. To see this, assume in accordance with the last subsection, that $\mathcal{M}$ contains function free FO formulas over single peer schemas and inter-peer tgds only. This translates to a LRM-PDMS $\hat{\mathcal{S}} = (\mathcal{P}, \mathcal{R}, \hat{\mathcal{M}})$, where $\hat{\mathcal{M}}$ contains one coordination formula $i : \phi$ for each formula $\phi \in \mathcal{M}$ over $\mathcal{R}_i$, and for each tgd $\forall \vec{x}\big(\exists \vec{z}\phi(\vec{x}, \vec{z}) \to \exists \vec{y}\psi(\vec{x}, \vec{y})\big)$ between peers P_i, P_j one coordination formula $\forall i{:}\ \vec{x}\big(i{:}\ (\exists \vec{z}\phi(\vec{x}, \vec{z})) \to j{:}\ (\exists \vec{y}\psi(\vec{x}, \vec{y}))\big)$. Again, for an instance I for $\mathcal{S}$, $\mathcal{I}_I$ w.r.t. $\mathcal{S}$ contains exactly those instances I' for $\hat{\mathcal{S}}$ that satisfy $\hat{\mathcal{M}}$ and agree with I.

5.2 Inconsistency Handling

Given a PDMS $\mathcal{S}$, we say an instance I for $\mathcal{S}$ is inconsistent (w.r.t. $\mathcal{S}$) if $\mathcal{I}_I = \emptyset$. One drawback of the semantics and approaches seen so far is that they cannot deal with such a situation.

They do not have sensible notions for query answering nor materialization in this case, and therefore their algorithms will either fail or return useless results. This is unsatisfactory, as it is very unlikely that a complete PDMS is consistent. In general, two kinds of inconsistencies are distinguished. *Local inconsistency* occurs if the data stored at a single peer is already inconsistent. In case of *P2P inconsistency* each peer is locally consistent, but the local data contradicts data implied by inter-peer mappings, or a peer imports contradicting information from different sources. In both cases, inconsistency occurring at a single peer immediately renders the complete system useless (as it leads to global inconsistency).

Local inconsistency is in general addressed by "excluding" locally inconsistent peers, i.e. by defining semantics that behave as if these peers were not part of the network. P2P inconsistency on the other hand is tackled by defining semantics that consider suitable repairs of the data, either by not importing contradicting facts or by ignoring the local data.

In [15], the first approach has been taken: In case of P2P inconsistencies, the local data at each peer is preferred, and a maximal amount of consistent data is imported. Applying local reasoning and considering an integration system, this semantics is formalized as an extension of [16] using the nonmonotonic, multi-modal epistemic logic $K45_n^A$.

▶ **Definition 9** ([15]). A P2PDIS is a PDMS $\mathcal{S}_{P2PDIS} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local data integration system and $\mathcal{M} = \mathcal{M}_L \cup \mathcal{M}_R \cup \mathcal{M}_P$. $\mathcal{M}_R$ is a set of constraints $\mathbf{K}_i\phi$ where ϕ is a function free FO formula over a single peer schema $\mathcal{R}_i \in \mathcal{R}$. $\mathcal{M}_L$ is a set of mappings $\mathbf{K}_i(\forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z})) \rightarrow (\exists\vec{y}\psi(\vec{x},\vec{y})))$ between some $\mathcal{L}_i$ and $\mathcal{R}_i$, and $\mathcal{M}_P$ contains inter peer mappings $\forall\vec{x}(\neg\mathbf{A}_i\bot_i \wedge \mathbf{K}_i(\exists\vec{z}\phi(\vec{x},\vec{z})) \wedge \neg\mathbf{A}_j(\neg\exists\vec{y}\psi(\vec{x},\vec{y})) \rightarrow \mathbf{K}_j(\exists\vec{y}\psi(\vec{x},\vec{y})))$ from P_i to P_j, where ϕ and ψ are conjunctions of atoms.

Again, $\mathbf{K}_i$ and $\mathbf{A}_i$ are modal operators from $K45_n^A$. Let I be an instance for $\mathcal{S}$. Intuitively, $\mathcal{M}_R$ and $\mathcal{M}_L$ express that the local mappings and constraints must be satisfied in each $I' \in \mathcal{I}_I$. The intuitive reading of the P2P mappings is as follows: If peer P_i is not locally inconsistent, and $\exists\vec{z}\phi(\vec{x},\vec{z})$ holds in every $I' \in \mathcal{I}_I$, and $\exists\vec{y}\psi(\vec{x},\vec{y})$ is consistent with the data at P_j, then $\exists\vec{y}\psi(\vec{x},\vec{y})$ should hold at P_j. Being a proper extension of local reasoning as described before, the drawback of the approach is the high complexity of query answering.

▶ **Theorem 10** ([15]). *Let $\mathcal{S}_{P2PDIS}$ be a P2PDIS where $\mathcal{M}_L$ is a set of GAV mappings and $\mathcal{M}_R$ contains key constraints only, and Q a CQ over some $\mathcal{R}_i \in \mathcal{R}$. Given an instance I for $\mathcal{L}$, and tuple t, deciding if $t \in Q^c(I)$ is coNP-complete (data complexity).*

Using repairs, [7, 8] considered both, omitting imported data and local data to resolve inconsistencies. This approach is able to deal with inconsistencies between imported and local data, but not with inconsistencies between data imported from different sources.

▶ **Definition 11** ([7]). A CPDE-System is a PDMS $\mathcal{S}_{CPDE} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$, where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance and $\mathcal{M} = \bigcup_{P_i,P_j\in\mathcal{P}} \mathcal{M}_{(P_i,P_j)} \cup \bigcup_{P_i\in\mathcal{P}} \mathcal{M}_{P_i}$. Each $\mathcal{M}_{(P_i,P_j)}$ and $\mathcal{M}_{P_i}$ may be empty or contain full disjunctive tgds and tgds with one atom in the lhs and rhs, defined over a single $\mathcal{R}_i \in \mathcal{R}$ ($\mathcal{M}_{P_i}$) or two $\mathcal{R}_i, \mathcal{R}_j \in \mathcal{R}$ ($\mathcal{M}_{(P_i,P_j)}$). In addition, each $\mathcal{M}_{(P_i,P_j)} \neq \emptyset$ is annotated with a trust relation $(P_i, [< \mid =], P_j)$.

P_j is a neighbor of P_i if $\mathcal{M}_{(P_i,P_j)} \neq \emptyset$, and $\mathcal{M}_{(P_i,P_j)} \neq \mathcal{M}_{(P_j,P_i)}$ is allowed. Based on the assumption that the transitive closure of the neighbor relation is acyclic, the semantics is inductively defined based on *neighborhood solutions* and *solution instances*: Given an instance I for $\mathcal{S}$, for a peer P_i with $\bigcup_{P_j\in\mathcal{P}} \mathcal{M}_{(P_i,P_j)} = \emptyset$, the set of *solution instances* $\mathcal{I}_I[P_i]$ for P_i is defined as the set of minimal repairs of $I|P_i$, and a *solution* $S(P_i) = \bigcap_{I'\in\mathcal{I}_I[P_i]} I'$. For a peer with neighbors $P^1, \ldots, P^n$, the *neighborhood solutions* are defined via repairs of the instance

$\hat{I} = I|P_i \cup \bigcup_{P^j} S(P^j)$ over the combined schemas of these peers, satisfying $\mathcal{M}_{P_i}$ and all P2P mappings $\mathcal{M}_{P_i,P^j}$. Thereby only those repairs are considered that are "closest" (cf. [7]) to $\hat{I}$ and agree with $\hat{I}$ on all $\mathcal{R}_j$ s.t. $(P_i, <, P_j)$, i.e. on data from neighbors trusted more than the local data. $\mathcal{I}_I[P_i]$ is then defined as the set of all neighborhood solutions restricted to $\mathcal{R}_i$.

Being an integration system, no data is materialized, but the goal is, given a query over some peer P_i and an instance I to compute the certain answers w.r.t. $\mathcal{I}_I[P_i]$. As shown in [7, 8], the problem can be encoded as answer set program. However, due to the disjunctive tgds and the repair semantics, it is intractable in general.

▶ **Theorem 12** ([7]). *Let $\mathcal{S}$ be an acyclic CPDE-System and Q a FO query over an $\mathcal{R}_i \in \mathcal{R}$. Given instance I for $\mathcal{R}$ and a tuple t, deciding if $t \in Q^c(I)$ is Π_2^P-complete (data complexity).*

In [22], the formalization of local reasoning presented there was also extended to address local inconsistency by redefining the semantics of mappings from locally inconsistent peers.

5.3 Update Exchange

Another problem not considered by the approaches presented so far are updates. For integration systems, in fact nothing changes in case of an update, as a query is answered on the data present at query time. In exchange systems however, updates pose several problems, based on the fact that later updates may revise and contradict earlier ones. Think e.g. of changing a non-key value. Standard methods like the chase would lead to inconsistencies. Another problem are e.g. deletions that lead to a mapping violation. Again, the chase would just undo this deletion, which is not satisfactory. Hence the methods discussed previously may not be appropriate in such settings, where data is both materialized and continuously changed (often referred to as *Collaborative Data Sharing (CDS)* or *Collaborative Data Integration*). The main idea behind approaches addressing these specific problems is not to exchange the information directly, but to exchange information about the updates. We will next take a look onto the most prominent examples of this approach.

5.3.1 Orchestra

One of the first and most cited approaches to realize CDS was formulated in the Orchestra project [43, 72, 31, 28]. Most notably its semantics can be defined almost completely in terms of schema mappings, although it differs a lot from the semantics seen so far.

▶ **Definition 13.** An O-PDMS is a PDMS $\mathcal{S}_{Orchestra} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instances, and $\mathcal{M}$ is a weakly acyclic set of tgds τ_i and key constraints. Each $\tau_i \in \mathcal{M}$ may contain relations from several peer schemas in both, its lhs and rhs and has a *trust condition* θ_i attached.

Trust conditions assign to each update propagated by a tgd a numerical *priority*, based on the content and the provenance of the update. The restriction to weak acyclicity is only made to guarantee the termination of the chase, so every set of tgds on which the chase terminates can be used.

So far, the setting looks similar to the previous ones. However, a different scenario is assumed, resulting in a completely different semantics. Here, every user works on its local database instance, i.e. queries are answered only locally and updates only affect the local instance. Further, updates done by users on their local databases are not immediately visible to the other peers in the network, but recorded in a local update log. At any time, a peer can decide to either publish its updates, which means that they are copied into some global update store. Or he can decide to import updates done at other peers into his local database. In this case, all updates published to the global update store by any peer since the last import are first translated according to the mappings $\mathcal{M}$, then filtered according to the trust conditions, and finally checked for mutual conflicts between these updates as well as for conflicts with the local updates. Thereby the system is explicitly designed to handle such conflicting updates: Local updates are always preferred to updates done at other peers, and conflicts between imported updates are either resolved using the trust conditions, and if this is not possible, the updates are deferred and a user has to select which to apply.

Formally, assume peer P_u chooses to import updates published by the others. Then the content of its peer relations is defined by the following PDMS $\mathcal{S}' = (\mathcal{P}, \mathcal{R}', \mathcal{M}')$, where $\mathcal{R}'$ contains for each peer relation $R \in \bigcup_{\mathcal{R}_i \in \mathcal{R}, R_j \in \mathcal{R}_i} R_j$ five relations: R^ℓ (*local contributions* table), R^r (*rejections* table), R^i (*input* table), R^t (*trusted input* table), and R^o (*output table*). Further, $\mathcal{M}'$ contains the following mappings: For each $\tau \in \mathcal{M}$, $\mathcal{M}'$ contains a mapping τ' obtained from τ by replacing any relation symbol in the lhs by the corresponding R^o, and each relation symbol on the rhs by R^i. Further, for each R, $\mathcal{M}'$ contains the tgds $R^i(\vec{x}) \wedge \mathit{trusted}(\vec{x}) \rightarrow R^t(\vec{x})$, $R^t(\vec{x}) \wedge \neg R^r(\vec{x}) \rightarrow R^o(\vec{x})$, and $R^\ell(\vec{x}) \rightarrow R^o(\vec{x})$. Thereby *trusted* is no real relation, but just denotes the filtering of updates according to the trust conditions and resolving of conflicts using priorities (see below). In fact, in [28], $R^t(\vec{x})$ was defined as $R^t(\vec{x}) = \mathit{trusted}(R^i(\vec{x}))$.

The content of the relations in $\mathcal{R}'$ is defined based on the information retrieved from the global update store: First, the sequence of updates in the store is *flattened* [43], i.e. dependencies between updates are removed (if for example a tuple t is first inserted and then changed to t', these updates are replaced by just inserting t'. Also, if a tuple is first inserted and then deleted, both updates are removed), such that no update depends on another one. Based on this flattened update sequence, an instance I for the relations in $\mathcal{R}'$ is defined as follows: Each R^ℓ contains all tuples locally inserted into R. R^r contains all tuples that were not inserted locally into R, but were delete from R according to the log (this means, the tuple was imported during an earlier update, and then deleted. It should therefore not be reinserted). R^i, R^t, and R^o are left empty. For each $R \in \mathcal{R}_u$, the content of R after the update is now defined as the content of R^o after chasing I with $\mathcal{M}'$.

Inconsistencies occur whenever for the same key values, different updates show up (e.g., if two different values are assigned to the non-key values). Due to space restrictions, we only sketch the basic idea of the conflict resolution algorithm (for details see [72, 28]): Updates in R^i are considered as candidate updates. If such a candidate update conflicts with local data, always the local data is preferred. Conflicts between candidate updates are resolved by the trust mappings: The update with the higher priority is chosen, the other discarded. In case that several conflicting updates have the same priority, the update is deferred until the user selected one to apply. As the priority value of an update depends heavily on its provenance, using an appropriate provenance model is important. In Orchestra, provenance semirings are used, as on the one hand they are powerful enough to provide all required information, and on the other hand their provenance expressions can be nicely transformed into trust expressions. A description of the provenance model of provenance semirings can be found in [29], a detailed discussion of how trust (trust or distrust an update) can be derived from provenance expression is given in [28]. The reconciliation algorithm for conflicting updates based on priorities was presented in [72], and recently developed further in [26].

Note that the description above was meant to define *what* tuples should be contained in the local instance after the update, and not *how* these values are indeed computed, as this would require to recompute the complete content. Instead, the instances can be computed incrementally. [28] presents algorithms for both, computing the effect of insertions and deletions, where the main problem is to determine the effect of a deletion, as propagating this deletion means to find all tuples that are consequences from the deleted one and can no longer be derived from other local insertions.

5.3.2 Youtopia

Yet another approach was chosen more recently in the Youtopia system [52].

▶ **Definition 14.** A Y-PDMS is a PDMS $\mathcal{S}_{Youtopia} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M}$ is a set of tgds.

Note that except inter-peer tgds – which<f can always be repaired by inserting tuples – no further constraints exist on $\mathcal{R}$. Violations of some $\tau_i \in \mathcal{M}$ introduced by an update are repaired using a variant of the chase, which, in the presence of tgds only, cannot fail. As usual, a violation occurs if in an instance I for $\mathcal{S}$, there is a set of tuples (called *witness*) matching the lhs of a tgd τ, without appropriate tuples matching the rhs of τ. Youtopia now distinguishes two kinds of violations: A *lhs-violation* occurs if the tuple affected by an update is part of the witness (e.g., after tuple insertion or replacing all occurrences of a labeled null by the same constant). It is corrected by a *forward chase*, which means that new tuples are added to I just like in the traditional chase. If, however, the tuple affected by the update is not part of the witness (e.g., if the violation is due to a tuple deletion), this is called a *rhs-violation*. Repairing such a violation by a forward chase would just undo the update, which is probably undesired. Therefore, Youtopia resolves this via a *backward chase*, that deletes tuples in the witness from I such that the tgd is no longer violated. If this causes again a violation, it is again a rhs-violation, and therefore resolved the same way.

In this setting, there are now two unresolved problems: First, the forward chase may not terminate (the backward chase terminates after deleting all tuples the latest), and second, there might be several possibilities for deleting tuples to satisfy a violated tgd. Both issues are resolved by asking the user for input: For the backward chase, whenever there is more than one possibility for deletion, the user is prompted to select one of them. On the other hand, whenever the forward chase produces a tuple t for some relation $R^I \in I$ that can be mapped via a homomorphism h onto a tuple $t' \in R^I$, the chase is stopped, and the user has to decide whether to add t to R^I, or to apply h to I (thus indicating that t contains no new information). Note that both actions satisfy τ. Then the following was shown.

▶ **Theorem 15** ([52]). *Any forward chase will either stop along all paths and ask for user input or terminate after finitely many steps.*

This does still not guarantee the termination of the forward chase, and even if it eventually terminates, it might be running (or waiting for user interaction) for quite a long time. To not freeze the system while waiting for termination, Youtopia allows different chase sequences to run in parallel. A well-defined semantics for these concurrent chases is provided by defining useful notions of serialization (e.g., extending final-state serializability) and safety (of executing and interleaving chase steps). A discussion of the concurrency related notions and results is out of the scope of this survey. We thus have to refer to [52] for any details.

5.3.3 ECA Rules

Event-Condition-Action (ECA) rules are a general, often used technique to coordinate distributed systems by triggering actions based on the occurrence of certain events. Not completely fitting to our definition of schema mapping based PDMSs, their use in PDMSs has been considered in [44, 45, 74], especially as a possibility for implementing schema mappings in exchange systems. We therefore close this section by a short review of them.

▶ **Definition 16.** A PDMS $\mathcal{S}_{ECA} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is the schema of a local database instance, and $\mathcal{M}$ is a set of ECA-rules.

Following [74], we consider ECA rules in a generic way as rules of the form **WHEN** $<$ *event* $>$, **IF** $<$ *condition* $>$, **THEN** $<$ *action* $>$, where the "IF" part is optional. ECA-rules easily allow to react to changes and updates in the system. On the other hand, the event based character makes it hard to formally define $\mathcal{I}_I$ given an instance I for $\mathcal{S}$. The idea is therefore to see them as an implementation of schema mappings, that ensure that in case of updates a materialized instance remains consistent w.r.t. a set of schema mappings.

Work done in this area addresses two main topics: Creation and evaluation of ECA-rules. [45] proposed a distributed evaluation mechanism for ECA-rules: Given one rule including several peers, the idea is to split it into several subrules that are then distributed among the peers involved and allow for a distributed evaluation of the rule. [45] further introduces a powerful event language and algebra. [44, 74] both consider the problem of semi-automatically creating ECA-rules: Given default rules between standard schemas for a certain domain and mappings between these schemas and concrete peer schemas, the goal is to translates the default rules into rules between the peer schemas.

6 Alternative Semantics

In the previous section, we concentrated on approaches based on schema level mappings between different peers. While they represent an important part of the discussion on PDMSs, those systems do not represent the complete range of possible semantics. In this section, we discuss semantics and approaches to PDMSs that are not based on mappings defined on the schema level. As an example of data mappings, we will take a closer look onto *mapping tables*. After this we discuss the idea of not just mapping schemas to schemas and data to data, but to also map between data and schemas on the example of *data-schema interplay*.

6.1 Instance based mappings: Mapping Tables

One assumption common to all approaches in the previous section was that all peers share the same domain, i.e. that they use the same domain elements to represent the real world. However, this assumption might be too strong for certain applications and, in addition, restricts peer autonomy. Recall that the LRM uses domain relations to extend the semantics of schema mappings to also support domain translations. This idea was developed further by dropping the schema mapping and defining P2P mappings solely on the data level by specifying value correspondences only. One such approach, that can be considered as an extension of the LRMs domain relations, are *mapping tables* [49, 47].

▶ **Definition 17.** A MT-PDMS $\mathcal{S}_{MT} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ is a PDMS where each $\mathcal{R}_i \in \mathcal{R}$ is either the schema of a local database instance or of a local data integration system, and $\mathcal{M}$ is a set of *mapping tables*.

Note that in [49, 47], the data is assumed to be stored directly under each $\mathcal{R}_i$ only. However, as we will see, mapping tables are only used to translate queries between peer schemas, so this does not make any difference. Next we will first describe mapping tables and their semantics as introduced in [49], and then, following [47], discuss how they can be used for query answering in PDMSs.

Mapping tables describe a relationship between data stored under two sets of attributes, extending domain relations in two ways: First, they need not be binary relations but may have bigger arities, and second, they may also contain variables and domain restrictions for these variables instead of just constants from the domain. Also, several mapping tables between two peers may exist.

In the following, assume that each attribute A_i appearing in $\mathcal{R}$ has its own domain $dom(A_i)$, and let $\mathcal{V}$ be a set of variables. An *attribute mapping* over a set $\vec{A}$ of attributes is a tuple t that contains for each $A_i \in \vec{A}$ either some $c \in dom(A_i)$, some $v \in \mathcal{V}$, or an expression $v - D$ where $v \in \mathcal{V}$ and D is a finite subset of $dom(A_i)$, i.e. $t = (t_1, \ldots, t_n)$ where $t_i \in (dom(A_i) \cup \mathcal{V} \cup \mathcal{D})$ and $\mathcal{D}$ is the set of expressions $v - D$. A *mapping table* T_M between two sets $\vec{A}, \vec{B}$ of attributes is a set of attribute mappings t^i over $\vec{A} \cup \vec{B}$ s.t. no variable occurs in two different attribute mappings $t^i, t^j \in T_M$ $(i \neq j)$. Further, a *mapping table constraint* τ is a triple $(T_M, \vec{A}, \vec{B})$. Intuitively, each mapping table T_M associates values for $\vec{B}$ to given values for $\vec{A}$. Formally, this is defined by means of *valuations*: For a variable v in an attribute mapping, let $\vec{A}_v$ be the set of attributes under which v appears. A valuation μ is a function mapping all constants in T_M onto themselves, each variable v in T_M into $\bigcap_{A_i \in \vec{A}_v} dom(A_i)$, and satisfies $\mu(v) \notin D$ for all expressions $v - D$ in T_M. Given a mapping table T_M and values $\vec{a}$ for $\vec{A}$, the set of values associated to $\vec{a}$ by T_M is $\vec{B}_{T_M}(\vec{a}) = \{\vec{b} \mid \text{there exists a valuation } \mu \text{ s.t. } t[\vec{A}] = \vec{a} \text{ and } t[\vec{B}] = \vec{b} \text{ for some } t \in \mu(T_M)\}$. Finally a tuple t for attributes $\vec{C}$ with $\vec{A} \cup \vec{B} \subseteq \vec{C}$ satisfies $\tau = (T_M, \vec{A}, \vec{B})$ if $t[\vec{B}] \in \vec{B}_{T_M}(t[\vec{A}])$, and a relation R^I satisfies τ if each $t \in R^I$ satisfies τ. Even more expressive mappings can be constructed by composing mapping table constraints to *mapping table formulas (MTF)* as $MTF = \tau | (MTF \wedge MTF) | (MTF \vee MTF) | \neg MTF$, where τ is a single mapping table constraint, and the definition of whether a tuple satisfies a MTF is a straight forward extension of the corresponding notion for mapping table constraints.

According to the above definitions, values for attributes $\vec{A}$ not appearing in the mapping table have no translation to $\vec{B}$. Assuming that a mapping table contains only partial information, another interpretation considered in [49] is that such values map to any values for $\vec{B}$. However, this behavior can be explicitly defined under the above semantics using the $v - D$ construct. We therefore omit its discussion here.

Next, following [47], we discuss how to use mapping tables to define P2P mappings. The general idea is to use mapping tables to rewrite a query on one peer to a query on another peer, to forward this query, and to repeat these steps. Unlike similar procedures seen in the previous sections, in this case answers of the resulting queries are not merged into a single answer, but returned separately. One reason for this is that in a mapping table constraint it need not be that $|\vec{A}| = |\vec{B}|$, hence the arities of the rewritten queries may not match.

Towards this goal it is first necessary to define when such a rewriting is correct, defined as *sound rewritings* in [47]. Informally, the idea is that a rewritten query should return only such tuples t' that are translations of correct answers t to the original query (but t may not be derived directly due to missing data). The formal definition for a sound rewriting from $\mathcal{R}_i$ to $\mathcal{R}_j$ is based on a mapping table T_M covering all attributes in $\mathcal{R}_i \cup \mathcal{R}_j$. Such a mapping table constraint can be composed from several mapping tables (basically as conjunctive mapping table formula matching uncovered attributes to all values — see [47] for details). Let Q_1 be a CQ. For the ease of notation, assume for the moment that equalities in Q_1 are not expressed by reusing variables but explicitly, and let $\psi(\vec{x}, \vec{y})$ be the conjunction of these equalities, i.e. Q_1: $ans(\vec{x}_1) \leftarrow \exists \vec{y}_1 \phi(\vec{x}_1, \vec{y}_1) \wedge \psi(\vec{x}_1, \vec{y}_1)$. As now every variable occurs only once in ϕ, we can identify variables with the attribute for the position where they occur, and therefore $\vec{x}_1$ defines a set of attribute names. A query $Q_2 \colon ans(\vec{x}_2) \leftarrow \exists \vec{y}_2 \phi'(\vec{x}_2, \vec{y}_2) \wedge \psi(\vec{x}_2, \vec{y}_2)$ is a a *sound rewriting* of Q_1 over $\mathcal{R}_j$ w.r.t. T_M if for every instance I for $\mathcal{R}_j$ and every $t' \in Q_2(I)$ there exists a valuation μ s.t. $t[\vec{x}_2] = t'$ holds for some $t \in \mu(T_M)$. A rewriting is further *complete* if it is sound and for every sound rewriting Q'_2 and instance I it holds that $Q'_2(I) \subseteq Q_2(I)$.

The problem of testing if a CQ is a sound rewriting of another CQ was shown to be $\Pi_2 P$-complete in [47]. There, also an algorithm for computing a complete rewriting of a CQ w.r.t. a mapping table T_M was presented, that we cannot discuss here due to space restrictions. Within a PDMSs, termination of the query forwarding is achieved in two ways. On the one hand, a maximal number of forwards and rewritings for each query can be specified. On the other hand for each query its path through the system is recorded. This record is used for cycle detection. Once a cycle is detected, forwarding on this path stops.

Returning our attention to mapping table formulas in general, there are two interesting problems that can be studied for sets of MTFs: The consistency and the inference problem. Given a mapping table formula τ and an attribute set $\vec{A}$, the consistency problem asks if there exists a relation for $R(\vec{A})$ that satisfies ϕ. The inference problem asks, given a set Σ of MTFs and a single MTF τ, if every relation satisfying Σ also satisfies τ. While the consistency problem is of obvious interest, the main interest in the inference problem stems from the wish to create mapping tables automatically. Given a set of mapping tables, the goal is to automatically derive new explicit mapping tables from them, as this may allow for more and more exact query rewritings. Unfortunately, both problems (being interreducible to each other) were shown to be NP-complete for the general case in [49].

Mapping tables have been combined with other data translation mechanisms (e.g., merge and conversion rules in [56]) or schema mappings (in form of ECA rules in [44]). They are also the main P2P mapping language in the Hyperion project [48, 4, 63] (see next section).

6.2 Data-Schema Interplay

Approaches for so-called data-schema-interplay extend the coordination formulas introduced in Section 4. In such formalisms, the domain dom_i also contains the names of the relations and attributes of the peer schema $\mathcal{R}_i$. Consequently, the formulas ϕ occurring in the coordination formulas can refer to both data and metadata. Pioneering work in this area was presented in [53]. An important example from the peer data management domain is HepToX [10].

7 PDMS prototype systems

So far, we discussed semantic approaches for PDMSs. We will now give a short overview on existing prototype systems, pointing out specifics or notable ideas. Due to space restrictions, we cannot give a general or detailed discussion on the implementation of PDMSs. For a deeper discussion of implementation related aspects, see [41].

A first suggestion for the general structure of a peer in a PDMS was already presented in the vision paper [6] that also first suggested the LRM. Not surprisingly, a peer consists of three main layers: The local data is managed by the storage layer. A P2P layer on the one hand is responsible for establishing and managing mappings and connections to other peers. On the other hand it also handles query rewritings, update exchange, domain translations, or any other kind of information exchange supported by the system. Basically all algorithms of interest for PDM are part of the P2P layer. Finally each peer is controlled through the interaction layer, containing e.g. the user interface. Most of the systems presented in the literature follow this schema.

Hyper. The *Hyper* framework [17] considered the implementation of local reasoning as introduced in [16] on a Data Grid architecture, with the main focus on how query answering under local semantics can be deployed on Grid infrastructure. We are not going to discuss the implementation on Grids here, but use this opportunity to shortly sketch the general idea for query answering under local reasoning with tgds $\tau\colon \forall\vec{x}(\exists\vec{z}\phi(\vec{x},\vec{z}) \to \exists\vec{y}\psi(\vec{x},\vec{y}))$ as inter-peer mappings (cf. Section 5.1.2). Recall that we can identify a CQ $Q_\tau\colon \mathit{ans}(\vec{x}) \leftarrow \exists\vec{z}\phi(\vec{x},\vec{z})$ with the lhs of each τ, and consider a PDEI-system $\mathcal{S} = (\mathcal{P},\mathcal{R},\mathcal{M})$ with $\mathcal{C}_E = \mathcal{M}_E = \emptyset$ but let $\mathcal{C}_I$ contain arbitrary local constraints, and an instance I for $\mathcal{S}$. The basic idea of the query answering algorithm is as follows: For each τ from P_i to P_j, add a new relation symbol R_τ of arity $|\vec{x}|$ to $\mathcal{R}_j$ and a new local constraint $\forall\vec{x}R_\tau(\vec{x}) \to \exists\vec{y}\psi(\vec{x},\vec{y})$ to $\mathcal{C}_I$. Now given a (arbitrary) query Q over some $\mathcal{R}_i$, first compute a *perfect reformulation* Q' of Q w.r.t. I and $\mathcal{C}_I$. (A perfect reformulation of a query w.r.t. an instance and a set of constraints is a rewriting of the query that, evaluated over the instance, returns exactly the certain answers w.r.t. the set of constraints.) Q' may contain original relation symbols from $\mathcal{R}_i$ (these parts of Q' can be evaluated over I immediately) as well as some of the new R_τ. To evaluate Q' on those, their content must be retrieved first. This is done by posing Q_τ on the peer the lhs of τ is defined on, where this procedure is repeated unless a cycle was detected. Finally Q' can be evaluated by iteratively evaluating the (reformulations of the) queries Q_τ and adding the results to R_τ until a fixpoint is reached. This general idea can be implemented e.g. using the chase (cf. [27]), or as datalog program (cf. [16]), and works for local reasoning in general.

coDB. In Section 5.1.2, we pointed out that [22] proposed a formalization of inter peer mappings that resolves to local reasoning in terms of (restricted) coordination formulas. This approach was implemented in the *coDB* PDMS [23, 21, 24]. Inter peer mappings are modelled as coordination formulas $i_1 : \phi_1(\vec{x},\vec{y}_1) \wedge \cdots \wedge i_k : \phi_k(\vec{x},\vec{y}_k) \to i : h(\vec{x})$ (where $\vec{x} = \bigcup_{i=1}^{k} \vec{x}_i$), and domain relations are not considered. The very basic idea of the query answering algorithm is similar to that presented for Hyper above. Therefore we do not discuss it here. A detailed description of its distributed implementation can be found in [23, 24]. Notably, the authors consider the problem of changes in the mappings while the algorithm is running. They show that their algorithm is sound and complete w.r.t. those mappings that remain stable during the runtime of the algorithm. Evaluation results of the coDB system are presented in [23].

PeerDB. *PeerDB* [59, 62, 61] takes a completely different approach than those seen so far. Instead of defining semantic mappings between the peers, for each relation and attribute name a set of metadata (basically a list of keywords) is maintained. If a query is posed against a peer, the system identifies relations at other peers that might be worth also querying by finding matches between the keywords attached to the relations and attributes used in the query and those stored at the other peers. If a match is found, the corresponding relation is added to a list of candidate relations. This search is done by sending software agents to all neighbors of a peer, where they search for matches and are again forwarded. Forwarding is stopped after a certain number of times. The list of possible matches is sent back to the initiating peer, where it is ranked and presented to the user, who selects those relations to use. The query is then rewritten accordingly and sent to the corresponding peers that return the answer. Besides this completely different kind of mappings, the system further adapts the topology of the network such that peers that contribute a lot of answers and matches become a direct neighbor. (Being a neighbor just means to send agents directly to this peer.) PeerDB further supports caching of answers to reduce the required bandwidth.

Orchestra. We already presented the *Orchestra* system in Section 5.3.1. As mentioned there, the ideas of update translation and exchange as well as conflict reconciliation based on trust mappings and provenance have been all implemented in the Orchestra prototype system [30, 43]. One focus of the implementation is how to perform the update exchange incrementally, i.e. how to identify which tuples to add or to delete, without recomputing all instances from scratch (for a detailed discussion of the corresponding algorithms see [28, 43]). Towards these goals, Orchestra always tries to reuse existing relational database management systems (RDBMSs) as much as possible and to find efficient implementations of all these aspects on top of traditional RDBMSs. One example for this is the encoding of provenance information in an RDBMS. Another important aspect is the implementation of the global update store. [72] provides a comparison between a centralized and a distributed solution for the storage of the published update logs.

Youtopia. Another system we already discussed earlier is *Youtopia* [52] (Section 5.3.2). The distinguishing property of its implementation is probably its capability to support several concurrent chases. Although providing algorithms to identify conditions under which it is safe for a chase to proceed, Youtopia does not block chases until their execution is guaranteed to be save, but applies an optimistic strategy that allows further chase steps to be scheduled even if they are not safe. This may lead to conflicts that are detected and resolved by aborting the corresponding chase. As this in turn may lead to cascading aborts, [52] considers three different scheduling algorithms and provides an experimental comparison of them. It was show that the number of aborts can be reduced to what seems to be an acceptable number.

Hyperion. The *Hyperion* project [60] was a large project on PDMSs. Part of it was the development of the *Hyperion* PDMS [48, 4, 63]. The main focus of this system was the use of mapping tables (see Section 6.1) for P2P mappings. The prototype provides algorithms for checking consistency of mapping table formulas, deriving new mapping expressions from existing ones, and computing rewritings of queries according to mapping tables. Further, in addition to mapping tables, Hyperion also uses ECA-rules (see Section 5.3.3) to support update exchange between peers, which is used to keep different peer instances consistent. Obviously those updates are also translated according to the information provided by the mappings tables. Consisting of three main components, the Hyperion PDMS basically follows the general structure of PDMSs presented at the beginning of this section.

Humboldt Peers. Humboldt Peers is a full-fledged relational PDMS that offers different strategies for completeness-driven query answering [64, 65]. As such, it follows a best-effort approach. To preserve peer autonomy as much as possible, Humboldt Peers resorts completely to local reasoning both in query answering and in building statistics of the data distribution accessible through neighboring peers. For that purpose, query answers are exploited to maintain multi-dimensional histograms on the potential cardinality of query answers received from neighboring peers. Building on that statistics, each peer performs local optimization in that it cuts off less promising peers from further query processing. To limit resource consumption for the highly redundant problem of query answering in large PDMS, Humboldt Peers sends a time budget along with each query. Using these pruning strategies, it was shown in [64, 65] that response time for query answering can be cut down by one or more orders of magnitude while still yielding a high completeness of the query result that can be satisfying for many applications.

8 Non-relational PDM

So far we concentrated solely on systems based on the relational data model. In this section, we will discuss approaches based on other data models like XML or RDF. We will also loosen the restriction only to consider unstructured P2P systems a little bit and will include some hybrid systems as, unlike the relational case, they are very common in this area.

Piazza. It might be surprising to find the *Piazza* PDMS here, as we already discussed it in Section 5.1.1 related to schema mappings. This was because the mapping language $\mathcal{PPL}$ and the algorithm for query answering were introduced and formalized in terms of relational schemas [37, 38]. However, the Piazza system was designed and implemented to work on XML [35] and even to support both XML and RDF, including mappings between both data models [36] by resorting to the XML representation of RDF. Nevertheless all basic ideas described in Section 5.1.1 remain unchanged: Mappings are still either storage descriptions or directed peer mappings (both can either be inclusion or equality mappings), but expressed in an (adapted) fragment of XQuery instead of relational CQs. Using XQuery allows the definition of more complex mappings that account to the nesting structure of XML. The main focus of the prototype system lies on the algorithm for query answering and its optimization. Implemented as a centralized algorithm, the reformulation step is computed at that peer the query is posed on. A global system catalog allows to retrieve all mappings currently present in the system. Recall that the idea of the algorithm is to compute rewritings of the query via a rule-goal tree. The resulting queries over the source relations are then sent to the corresponding peers to be executed. As the size of this tree grows quickly w.r.t. the number of mappings, optimizing query rewriting is crucial. Several strategies have been considered in [70], including pruning (i.e. identifying subtrees whose expansion will not create any new answers), finding good strategies for which nodes to expand next, or the computation of “shortcuts” by mapping compositions. Further, instead of returning the complete answers at the end, whenever the rewriting produces a query over some source relations, it is immediately executed and the result is presented to the user. Concerning the implementation of PDMSs, these optimization methods are specific to Piazza.

AXML. Another XML-based approach that gained a lot of attraction is *Active XML (AXML)* [42, 2]. First of all, AXML is an extension of XML that allows XML-documents not to contain all information explicitly, but to embed web service calls. Executing those calls then retrieves new data that is appended to the document. In general these may be arbitrary web services just returning AXML documents, not giving rise to any PDMS. However, as part of the AXML project so called *AXML peers* were created (cf. [2]). Each of these peers contains a set of AXML documents, is capable of performing webservice calls, and may publish its own services. These services, defined as (parameterized) queries over the stored documents, give rise to the following AXML-PDMS[3]: Each peer stores a set of AXML documents and may provide access to parts of these information through a web service, defined as parameterized query over its documents and returning AXML documents. Information offered by other peers can be accessed by including calls to their web services into the own AXML documents. I.e., P2P mappings are defined in terms of queries on other peers, a concept similar to tgds, that can be considered to describe the data to import also in terms of queries. Within an AXML document, service calls are encoded as special

[3] Note that our definition of a PDMS $\mathcal{S} = (\mathcal{P}, \mathcal{R}, \mathcal{M})$ only makes sense for schema based systems.

XML elements whose children represent the call parameters. When activated, the root node of the result-AXML document is appended to the document as sibling of the call element. Several policies can be applied to define how long the result remains valid or what happens with the result in case the service is called again. One difficulty with this materialization of data is that since the result of a web service call can be an arbitrary AXML document, it may contain further call elements, hence materializing all implicit information might not be possible. As a result, when computing the answer to a query an important goal is to call only those web services necessary to answer the query, but maybe even return service calls (instead of their return values) in the answer (lazy evaluation). However, as web services in general are just black boxes for the caller, reasoning about these questions (termination, was already enough data materialized for query answering) is not possible. But as in AXML-PDMSs the service definitions are known (at least from some global point of view), these problems have been investigated in [1] for AXML-PDMSs where web services are defined using a monotone conjunctive fragment of XQuery. Still undecidable in the general case, several decidable fragments of these problems have been identified. We close the discussion by pointing to two (out of many) extensions of AXML: [18] suggested a trust model for services, and [3] proposed an algebra for evaluating AXML expressions. For a general overview on AXML see [2].

SmurfPDMS. *SmurfPDMS* focuses on skyline-querying in a PDMS setting [40, 39]. Similarly to *Humboldt Peers*, this prototype system employs so-called data summaries to route queries through the network of peers. This means, that paths are pruned if they do not promise a certain size of contribution to the query answer. The statistics in the data summaries are maintained by exchanging updates between peers. To limit this update traffic in the network, the updates are only propagated over a certain number of peer mappings. So the statistics at each peers have a limited horizon.

HePToX. We already mentioned *HePToX* [9, 10] shortly in Section 6.2. Unlike most of the other systems, that require the user to provide a complete specification of the mappings, HePToX heavily supports the mapping creation. The data stored at each peer must be structured according to a DTD. All the user has to do is to draw some arrows between elements of the schemas that relate to each other, and to visually group different elements that match to the same element in the other schema. The system then translates these mappings into HePToX mapping language, which are datalog-style rules, adapted to be able to deal with the nesting structure of XML documents. They are somewhat similar to nested tgds [25], using Skolem functions to identify nodes (instead of using existential variables like FO-tgds). Using these mappings, queries posed against the local schema of a peer and formulated in a fragment of XQuery are translated to match the schemas of the neighbors. Thereby a rewriting is considered to be correct if evaluating the rewritten query over the data of the other peer returns the same result as applying the transformation to the data and evaluating the original query there.

XPeer. XPeer [66] is the first hybrid P2P system that we consider. Peers, clustered around super-peers, publish a schema of their locally stored data to their super peer in terms of so called *tree-guides* that are automatically generated from the data. Basically these tree-guides represent the structure of the XML data but omitting the actual data. The super-peer network forms a tree and super-peers exchange information such that each super-peer knows about the schema information stored at its children. Queries, issued against single peers, are handed over to the corresponding super-peer, from where they are routed through the super-peer network by trying to find matches between the query and the peer schemas stored at the super-peers. To improve this search for peer clusters that might be relevant for answering a query, the systems aims at assigning peers with similar schemas to the same super-peer, i.e. into the same cluster. Further, peers within the same cluster may replicate data from each other to speedup query answering. Also, the super-peer network adapts the number and distribution of super-peers automatically to the system load to avoid bottlenecks.

A survey of XML data management in P2P systems can be found in [51], focusing on the use of indices, clustering, replication and query processing in such systems.

The idea of PDM has been also picked up by the Semantic Web community, resulting in a variety of RDF based PDMSs. Interestingly, most of these systems differ greatly from the systems we have seen so far, introducing several new ideas for defining mappings. In the remainder of this section we will introduce some of those systems and discuss a selection of those approaches. Although arguable not all of them are covered by our definition of a PDMS we gave at the beginning, we think it is worth to mention them nevertheless. Probably influenced by the idea of the web, many of these systems do not define explicit mappings between pairs of peers: all peers in the network are considered as possible candidates for query forwarding, and the task is to identify those peers that indeed contain relevant data.

Edutella. One of the most prominent RDF based PDMSs is Edutella [55, 57, 58], that was originally designed for sharing educational resources by publishing RDF metadata describing these resources and to provide a querying service on this metadata. The main goal of Edutella is to provide an efficient mechanism for query routing, that forwards the query quickly to all peers in the network that may contribute to its answer, but avoiding forwarding the query to peers that do not. To reach this goal, a strong focus was laid on indices for query routing, which is specific to Edutella. Although Edutella offers several other functionalities, we will only shortly sketch the idea of using indices to guide query answering in a PDMS. Edutella is a *hybrid* P2P system where each peer connects to exactly one super-peer. Those super-peers, arranged in some predefined topology are then responsible for efficient routing. Queries posed against a peer are handed to the super-peer, from where they are routed through the super-peer network to peers that may contribute to the answer. Routing is based on several indices maintained by the super-peers: First of all, each super-peer stores information about the data provided by the peers connected to it that allows to determine if a peer can understand the query. This includes information on the peer schema and for which parts of this schema the peer actually contains data, but also the ranges of present values or other value summaries. If a query arrives at a super-peer, these indices are used to identify suitable peers for the query. In addition to the information about the peers connected to it, each super-peer also stores indices about its neighboring super-peers. These are summaries of the peer indices hold by the neighbors, describing the overall data offered by all peers connected to a neighbor. Routing within the super-peer network is directed by these indices.

To ensure that this routing strategy does not lead to basically broadcasting the query through the network, it is necessary that peers that may contribute to the answer are not distributed randomly in the network. Edutella therefore tries to cluster peers based on the data they offer. To do so, each super-peer can define certain constraints (like e.g. the supported schemas) peers have to satisfy in order to be allowed to connect to this super peer.

SQPeer and Bibster. We have just seen how index information are used for guiding query answering. Another possible use of data descriptions offered by a peer are advertisements, which gives rise to a different way of defining mappings between peers. The general idea of advertisements is that peers announce description of their data. If another peer decides to store such an advertisement, a mapping between these peers is established: Every time a peer receives a query, it will not only try to answer it over its own data, but will also check all of its stored advertisements whether they are relevant for this query. Two systems following this approach in different ways are *Bibster* [33] and *SQPeer* [50].

Bibster was designed for the very limited scope of sharing bibliographic information stored in Bibtex files at each peer. Advertising is done referring to some global ontology (for the bibliographic domain, this was the ACM classification hierarchy). Each peer describes its *expertise*, i.e. that parts of the ontology for which it actually provides data, as subsets of this ontology. When a query is posed against a peer, it is first answered locally, and then the stored expertises are used to identify peers that may be worth forwarding the query to. This decision is based on a similarity measure computed between the query and advertisements. Note that due to the need of a global ontology, Bibster is not a PDMS according to our definition, but still an interesting approach as we think.

SQPeer takes a little bit different approach than Bibster. Storing again data under RDF schemas, similar to the situation in Edutella (and considered in several RDF based systems), a peer may not contain data for all parts of this schema. The advertisements of a peer consist of descriptions of the so called *active schema*, i.e. of those parts of the schema a peer actually stores data for. Queries are again sent through the network based on the published and stored advertisements. But instead of immediately answering the query, each peer uses its stored advertisements to identify which parts of the query could be answered by which peer and annotates the query with the corresponding information. At the end, the annotated query is sent back the peer where the query was originally issued. This peer then uses the annotated information from the query to contact all relevant peers and collects their answers.

9 Conclusion

In this paper, we provided a survey of Peer Data Management Systems (PDMSs), concentrating on the management of structured data in unstructured peer-to-peer (P2P) systems. The promise of these systems is the combination of a strong semantics (like for relational data integration or exchange) with the high flexibility and autonomy offered by P2P systems. As the design of a PDMS raises many questions, several suggestions have been made in the literature to overcome the different design problems. Providing a summary of these approaches, we laid a strong focus on the formalisms and semantics of the P2P mappings, that more or less determine the semantics of the overall system.

P2P mappings can be designed for different purposes. They may be used for integrating data at query time, for data exchange or update propagation. Mappings may be able to deal with inconsistencies or support translations between different domains. In summary, several characteristics of inter-peer mappings can be identified. They may be defined on schema or instance level or even between instances and schemas. Further, while some offer a well-defined semantics and allow for reasoning along them, others just define explicit rewriting or translation rules for queries or data. Another distinction can be made on whether they are explicitly defined or created on the fly at runtime, e.g. based on a query issued by a user and some additional metadata.

Beside the description of the main theoretical concepts, we also discussed some of the problems arising from the implementation of such systems, pointing out specifics of several prototype systems published in the literature.

Table 1 An overview on the most important systems and approaches presented in this survey. (p) in the "special mentions" column indicates that a prototype implementation exists, (e) and (i) in the "semantics" column denote exchange and integration systems, respectively, and $\subset L$ is used to denote a fragment of L. (* syntactically restricted FO formulas)

	data model	mapping level	mapping language	query language	semantics	special mentions	Ref.
trad. data exchange/ integration	relational	schema	tgds	UCQs	global		[46, 54]
Piazza	relational/ XML	schema	$\mathcal{PPL}$	UCQs $\subset$XQuery	global (i)	(p), restricted topology	[38]
PDEI-Framework	relational	schema	tgds	UCQs	local (e/i)	eff. decidable	[27]
LRM	relational	schema & instance	CFs/dom. relations	CFs	local (i)	domain translation	[67]
Orchestra	relational	schema	tgds	UCQs	global (e)	(p), trust, update exchange	[28]
Hyperion	relational	schema & instance	ECA & mapping tables	CQs	- (e/i)	(p)	[4]
[7],[8]	relational	schema	UDECs* RDECs*	FO	repair (i)	inconsistency handling	[7, 8]
Hyper	relational	schema	tgds	UCQs	local (i)	(p)	[17]
AXML	XML	schema	WS calls	various	- (i)	(p)	[2]
Youtopia	relational	schema	tgds	keyword & structured	global (e)	backward chase concurrency,(p)	[52]
coDB	relational	schema	$\subset$CFs	UCQs	local (i)	$\exists$ ext. for local inconsistency,(p)	[22]
HepToX	XML	data/schema interplay	datalog style rules	$\subset$XQuery	- (i)	graphical mapping generation,(p)	[10]
Humboldt Peers	relational	schema	GLaV	CQs w/o projections	local (i)	(p), compl. driven, data statistics	[65]
Smurf-PDMS	relational, XML	schema	GLaV	CQs	local (i)	(p), data statistics	[40]
PeerDB	relational	schema	keywords	CQs	- (i)	(p)	[59]

A summary of the more important systems and approaches discussed in this survey is presented in Table 1. It lists for each of them the main characterizing properties: The data model used, whether the P2P mappings are on schema or instance level (or a mixture of that), the formalism used to define the inter-peer mappings, how these mappings are interpreted (i.e. the semantics applied to the mappings), and whether the system is an exchange or integration system. The table further shows the query language that is either supported by the approach or discussed in detail within the context of the specific proposal. Any special characteristics of a system are stated in the "special mentions" column, where (p) indicates that a prototype implementation exists. Note that the table only contains a selection of systems and is not complete: even some systems mentioned in this survey do not show up.

Despite all these different approaches, none of them seems to have been established yet as *the* model for PDM. This may indicate that not all problems have been resolved in a satisfactory way yet. One of these problems might be data inconsistency. Although very elegant ways have been suggested for how to deal with contradicting data, most of the time they come for the price of a high computational complexity. In general, performance is another critical aspect, and how to further speed up query answering or the data exchange also is an interesting research direction. Altogether, although the semantics of PDMSs is already understood quite well and the interest in those systems was decreasing the last two years, there is still research potential in PDM.

References

1 Serge Abiteboul, Omar Benjelloun, and Tova Milo. Positive active xml. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 35–45, 2004.

2 Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: an overview. *VLDB J.*, 17(5):1019–1040, 2008.

3 Serge Abiteboul, Ioana Manolescu, and Emanuel Taropa. A framework for distributed xml data management. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, LNCS, pages 1049–1058. Springer, 2006.

4 Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos. The hyperion project: from data integration to data coordination. *SIGMOD Record*, 32(3):53–58, 2003.

5 M.T. Ăzsu and P. Valduriez. *Principles of Distributed Database Systems.* Springer, 2011.

6 Philip A. Bernstein, Fausto Giunchiglia, Anastasios Kementsietsidis, John Mylopoulos, Luciano Serafini, and Ilya Zaihrayeu. Data management for peer-to-peer computing : A vision. In *Proc. of the ACM SIGMOD Workshop on The Web and Databases (WebDB)*, pages 89–94, 2002.

7 Leopoldo E. Bertossi and Loreto Bravo. The semantics of consistency and trust in peer data exchange systems. In *Proc. of the Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, volume 4790 of *LNCS*, pages 107–122. Springer, 2007.

8 Leopoldo E. Bertossi and Loreto Bravo. Information sharing agents in a peer data exchange system. In *Proc. of the Int. Conf. on Data Management in Grid and Peer-to-Peer Systems*, pages 70–81, 2008.

9 Angela Bonifati, Elaine Qing Chang, Terence Ho, and Laks V. S. Lakshmanan. HepToX: Heterogeneous peer to peer XML databases. *CoRR*, abs/cs/0506002, 2005.

10 Angela Bonifati, Elaine Qing Chang, Terence Ho, Laks V. S. Lakshmanan, and Rachel Pottinger. HePToX: Marrying XML and heterogeneity in your P2P databases. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 1267–1270. ACM, 2005.

11 Angela Bonifati, Panos K. Chrysanthis, Aris M. Ouksel, and Kai-Uwe Sattler. Distributed databases and peer-to-peer databases: past and present. *SIGMOD Record*, 37(1):5–11, 2008.

12 Athman Bouguettava, Boualem Benatallah, and Ahmed Elmagarmid. An overview of multidatabase systems: Past and present. In Ahmed K. Elmagarmid, Marek Rusinkiewicz, and Amit Sheth, editors, *Management of heterogeneous and autonomous database systems*, pages 1–32. 1999.

13 Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 260–271. ACM, 2003.

14 Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in p2p systems. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 77–90. Springer, 2003.

15 Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Inconsistency tolerance in p2p data integration: An epistemic logic approach. *Inf. Syst.*, 33(4–5):360–384, 2008.

16 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 241–251. ACM, 2004.

17 Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Riccardo Rosati, and Guido Vetere. Hyper: A framework for peer-to-peer data integration on grids. In *Semantics for Grid Databases, First Int. IFIP Conf. on Semantics of a Networked World (ICSNW). Revised Selected Papers*, volume 3226 of *LNCS*, pages 144–157. Springer, 2004.

18 Etienne Canaud, Salima Benbernou, and Mohand-Said Hacid. Managing trust in active xml. In *Proc. of the Int. Conf. on Services Computing (SCC 2004), 15-18 September 2004, Shanghai, China*, pages 41–48, 2004.

19 Philippe Cudre-Mauroux. Peer data management system. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 2055–2056. Springer US, 2009.

20 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

21 Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. Queries and updates in the codb peer to peer database system. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, VLDB '04, pages 1277–1280. VLDB Endowment, 2004.

22 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 64–76. Springer, 2003.

23 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. The coDB robust Peer-to-Peer database system. In *Proc. of the Twelfth Italian Symposium on Advanced Database Systems (SEBD)*, pages 382–393, 2004.

24 Enrico Franconi, Gabriel M. Kuper, Andrei Lopatenko, and Ilya Zaihrayeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 446–455. Springer, 2004.

25 Ariel Fuxman, Mauricio A. Hernández, C. T. Howard Ho, Renée J. Miller, Paolo Papotti, and Lucian Popa. Nested mappings: Schema mapping reloaded. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 67–78, 2006.

26 Wolfgang Gatterbauer and Dan Suciu. Data conflict resolution using trust mappings. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 219–230. ACM, 2010.

27 Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On reconciling data exchange, data integration, and peer data management. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 133–142. ACM, 2007.

28 Todd Green, Grigoris Karvounarakis, Zachary Ives, and Val Tannen. Update exchange with mappings and provenance. Technical report, University of Pennsylvania, 2007. Department of Computer and Information Science; Technical Report No. MS-CIS-07-26.

29 Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 31–40. ACM, 2007.

30 Todd J. Green, Gregory Karvounarakis, Nicholas E. Taylor, Olivier Biton, Zachary G. Ives, and Val Tannen. ORCHESTRA: facilitating collaborative data sharing. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 1131–1133. ACM, 2007.

31 Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 675–686. ACM, 2007.

32 Steven D. Gribble, Alon Y. Halevy, Zachary G. Ives, Maya Rodrig, and Dan Suciu. What can databases do for peer-to-peer? In *WebDB Workshop on Databases and the Web*, pages 31–36, 2001.

33 Peter Haase, Jeen Broekstra, Marc Ehrig, Maarten Menken, Peter Mika, Mariusz Olko, Michal Plechawski, Pawel Pyszlak, Björn Schnizler, Ronny Siebes, Steffen Staab, and Christoph Tempich. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proc. ISWC*, volume 3298 of *LNCS*, pages 122–136. Springer, 2004.

34 Alon Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

35 Alon Y. Halevy, Zachary G. Ives, Jayant Madhavan, Peter Mork, Dan Suciu, and Igor Tatarinov. The Piazza peer data management system. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

36 Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. Piazza: data management infrastructure for semantic web applications. In *Proc. of the Int. World Wide Web Conf. (WWW)*, pages 556–567. ACM, 2003.

37 Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 505–516. IEEE Computer Society, 2003.

38 Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDB J.*, 14(1):68–83, 2005.

39 Katja Hose. *Processing Rank-Aware Queries in Schema-Based P2P Systems.* PhD thesis, Technische Universität Ilmenau, 2009.

40 Katja Hose, Christian Lemke, and Kai-Uwe Sattler. Processing relaxed skylines in PDMS using distributed data summaries. In *Proc. of the Conf. on Information and Data Management (CIKM)*, 2006.

41 Katja Hose, Armin Roth, Andre Zeitz, Kai-Uwe Sattler, and Felix Naumann. A research agenda for query processing in large-scale peer data management systems. *Inf. Syst.*, 33(7–8):597–610, 2008.

42 INRIA. Active xml website (www.activexml.net/). `http://www.activexml.net/`, 2009. Accessed 22. April 2011.

43 Zachary G. Ives, Nitin Khandelwal, Aneesh Kapur, and Murat Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *CIDR*, pages 107–118, 2005.

44 Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietsidis, and Marcelo Arenas. Coordinating peer databases using ECA rules. In *Proc. of the Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, volume 2944 of *LNCS*, pages 108–122. Springer, 2003.

45 Vasiliki Kantere, John Mylopoulos, and Iluju Kiringa. A distributed rule mechanism for multidatabase systems. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE - OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3–7, 2003*, volume 2888 of *LNCS*, pages 56–73. Springer, 2003.

46 Anastasios Kementsietsidis. Data sharing and querying for Peer-to-Peer data management systems. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 177–186. Springer, 2004.

47 Anastasios Kementsietsidis and Marcelo Arenas. Data sharing through query translation in autonomous sources. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 468–479. Morgan Kaufmann, 2004.

48 Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Managing data mappings in the hyperion project. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 732–734. IEEE Computer Society, 2003.

49 Anastasios Kementsietsidis, Marcelo Arenas, and Renée J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 325–336. ACM, 2003.

50 Giorgos Kokkinidis and Vassilis Christophides. Semantic query routing and processing in p2p database systems: The ics-forth sqpeer middleware. In *EDBT 2004 Workshops, Revised Selected Papers*, volume 3268 of *LNCS*, pages 486–495. Springer, 2004.

51 Georgia Koloniari and Evaggelia Pitoura. Peer-to-peer management of xml data: issues and research challenges. *SIGMOD Record*, 34(2):6–17, 2005.

52 Lucja Kot and Christoph Koch. Cooperative update exchange in the youtopia system. *PVLDB*, 2(1):193–204, 2009.

53 L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. Schemasql: A language for interoperability in relational multidatabase systems. In *22nd Conference on Very Large Databases, Bombay,India, 1996*, pages 239–250. Morgan Kaufman Publishers, 1996.

54 Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the Symposium on Principles of Database Systems (PODS)*, pages 233–246, 2002.

55 Alexander Löser, Wolf Siberski, Martin Wolpers, and Wolfgang Nejdl. Information integration in Schema-Based Peer-To-Peer networks. In *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*, volume 2681 of *LNCS*, pages 258–272. Springer, 2003.

56 Mehedi Masud, Iluju Kiringa, and Anastasios Kementsietsidis. Don't mind your vocabulary: Data sharing across heterogeneous peers. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE, OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 – November 4, 2005, Proceedings, Part I*, volume 3760 of *LNCS*, pages 292–309. Springer, 2005.

57 Wolfgang Nejdl, Boris Wolf, Changtao Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *Proc. of the Int. World Wide Web Conf. (WWW)*, pages 604–615. ACM, 2002.

58 Wolfgang Nejdl, Boris Wolf, Wolf Siberski, Changtao Qu, Stefan Decker, Michael Sintek, Ambjorn Naeve, Mikael Nilsson, Matthias Palmer, and Tore Risch. Edutella: P2p networking for the semantic web. Technical report, Hannover University: Distributed Systems Institute - Knowledge Based Systems, 2003. Accessed 1. August 2009.

59 Wee Siong Ng, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. PeerDB: A P2P-based system for distributed data sharing. In *Proc. of the Int. Conf. on Data Engineering (ICDE)*, pages 633–644. IEEE Computer Society, 2003.

60 University of Toronto Database Group. Hyperion project website (www.cs.toronto.edu/db/hyperion/index.html. `http://dblab.cs.toronto.edu/project/hyperion/`, 2009. Accessed 19. May 2011.

61 Beng Chin Ooi, Yanfeng Shu, and Kian-Lee Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record*, 32(3):59–64, 2003.

62 Beng Chin Ooi, Kian-Lee Tan, Aoying Zhou, Chin Hong Goh, Yingguang Li, Chu Yee Liau, Bo Ling, Wee Siong Ng, Yanfeng Shu, Xiaoyu Wang, and Ming Zhang. PeerDB: Peering into personal databases. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, page 659. ACM, 2003.

63 Patricia Rodríguez-Gianolli, Maddalena Garzetti, Lei Jiang, Anastasios Kementsietsidis, Iluju Kiringa, Mehedi Masud, Renée J. Miller, and John Mylopoulos. Data sharing in the hyperion peer database system. In *Proc. of the Int. Conf. on Very Large Databases (VLDB)*, pages 1291–1294. ACM, 2005.

64 Armin Roth. Completeness-driven query answering in peer data management systems. In *Proc. of the VLDB 2007 PhD Workshop*, 2007.

65 Armin Roth. *Efficient Query Answering in Peer Data Management Systems.* PhD thesis, Humboldt Universität zu Berlin, 2012.

66 Carlo Sartiani, Paolo Manghi, Giorgio Ghelli, and Giovanni Conforti. Xpeer: A self-organizing xml p2p database system. In *EDBT Workshops, Revised Selected Papers*, LNCS, pages 456–465. Springer, 2004.

67 Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *Proc. of the Int. and Interdisciplinary Conf. on Modeling and Using Context*, volume 2680 of *LNCS*, pages 286–299. Springer, 2003.

68 Steffen Staab and Heiner Stuckenschmidt, editors. *Semantic Web and Peer-to-Peer – Decentralized Management and Exchange of Knowledge and Information*. Springer, 2006.

69 Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*. Number 3485 in LNCS. Springer, 2005.

70 Igor Tatarinov and Alon Y. Halevy. Efficient query reformulation in peer-data management systems. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 539–550. ACM, 2004.

71 Igor Tatarinov, Zachary G. Ives, Jayant Madhavan, Alon Y. Halevy, Dan Suciu, Nilesh N. Dalvi, Xin Dong, Yana Kadiyska, Gerome Miklau, and Peter Mork. The piazza peer data management project. *SIGMOD Record*, 32(3):47–52, 2003.

72 Nicholas E. Taylor and Zachary G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*, pages 13–24. ACM, 2006.

73 Quang Hieu Vu, Mihai Lupu, and Beng Chin Ooi. *Peer-to-Peer Computing - Principles and Applications*. Springer, 2010.

74 Dan Zhao, John Mylopoulos, Iluju Kiringa, and Verena Kantere. An ECA rule rewriting mechanism for peer data management systems. In *Proc. of the Int. Conf. on Extending Database Technology (EDBT)*, volume 3896 of *LNCS*, pages 1069–1078. Springer, 2006.

Management of Inconsistencies in Data Integration *

Ekaterini Ioannou[1] and Sławek Staworko[2]

1 Technical University of Crete, Greece
ioannou@softnet.tuc.gr
2 Mostrare, INRIA Lille – Nord Europe
University of Lille 3, France
slawomir.staworko@inria.fr

Abstract

Data integration aims at providing a unified view over data coming from various sources. One of the most challenging tasks for data integration is handling the inconsistencies that appear in the integrated data in an efficient and effective manner. In this chapter, we provide a survey on techniques introduced for handling inconsistencies in data integration, focusing on two groups. The first group contains techniques for computing consistent query answers, and includes mechanisms for the compact representation of repairs, query rewriting, and logic programs. The second group contains techniques focusing on the resolution of inconsistencies. This includes methodologies for computing similarity between atomic values as well as similarity between groups of data, collective techniques, scaling to large datasets, and dealing with uncertainty that is related to inconsistencies.

1998 ACM Subject Classification H.2.m [Database Management]: Miscellaneous

Keywords and phrases Data integration, Consistent query answers, Resolution of inconsistencies

Digital Object Identifier 10.4230/DFU.Vol5.10452.217

1 Introduction

Data integration aims at providing a unified view over data coming from various sources, for example data from different applications, collections, or databases [55]. Providing efficient data integration has received considerable attention by the database community and a variety of approaches have been suggested, spanning from integrating relational databases with the same schema to integrating unstructured, highly heterogeneous data collections. One of the most challenging tasks that existing techniques for data integration focused on is the efficient handling of inconsistencies that appear in the integrated data. The focus of this survey is to present and discuss existing techniques that are able to manage/handle inconsistencies in an efficient and effective manner.

Inconsistencies in data integration can appear for various reasons. One of the most common sources is the use of different schemata and formats in the data that must be integrated. As an example, consider a scenario where we need to integrate three databases

* This research has been co-financed by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council, FEDER through the Contrat de Projets Etat Region (CPER) 2007–2013, Codex project ANR-08-DEFIS-004, the European Union (European Social Fund – ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) – Research Funding Program: Thalis. Investing in knowledge society through the European Social Fund.

Data Exchange, Integration, and Streams. *Dagstuhl Follow-Ups*, Volume 5, ISBN 978-3-939897-61-3.
Editors: Phokion G. Kolaitis, Maurizio Lenzerini, and Nicole Schweikardt; pp. 217–235
DAGSTUHL FOLLOW-UPS Dagstuhl Publishing
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany

providing basic information about Muppets, i.e., the CBS trivia, the Vanity Fair magazine, and the DMV database. A fraction of the data from these databases is as follows:

CBS

Name	Job	DoB
Kermit	Manager	14.03.1965
J. Statler	Old Man	12.04.1946
Miss Piggy	Diva	21.06.1976
Gonzo	Stunman	01.03.1982

VF

Name	Job	DoB
Kermit	Manager	14 May 1965
J. Statler	Old Man	18 June 1942
Mlle Piggy	Star	1 April 1936
Gonso	Stunman	1 March 1982

DMV

Name	Job	DoB
Kermit	Manager	03/14/65
J. Statler	Old Man	06/18/42
Ms. Piggy	Diva	01/09/90
Gonzo	Daredevil	03/01/82

We can easily observe that integrating the data of these three databases causes inconsistencies. For instance, inconsistencies arise from the use of different formats that represent the dates (i.e., the DoB attributes), and the existence of spelling mistakes (i.e., in the name of Gonzo). Two additional reasons of inconsistencies are the use of variance, such as for representing “Miss Piggy”, and the use of close synonyms, such as “Diva” with “Star”, and “Stunman” with “Daredevil”.

Modern systems, e.g., Web 2.0 applications, have introduced new challenges to handling inconsistencies, which include the use of unstructured data, and higher levels of heterogeneity. As also illustrated in the previous example, to effectively handle inconsistencies we need to consider text variations, i.e., using similar strings for the same objects. Variations in text can appear due to introduced spelling mistakes, or due to the use of acronyms (e.g., “ICDE” for ‘International Conference on Data Engineering”), or abbreviations (e.g., “J. Web Sem.” for “Journal of Web Semantics”). Another important source of data inconsistencies is the evolving nature of the data. In essence, as time passed, data is added, removed, or modified [69]. For example, the famous ex-lady of US was born as “Jacqueline Lee Bouvier” but this was later changed to “Jackie Kennedy” and then to “Jackie Onassis”. In addition, each source providing data for integration will provide data in a way most adequate for its purpose. For instance, a publication will describe a person using the full name and affiliation, whereas an email will use the email address. This is also amplified by the lack of a global coordination for identifier assignment that forces each source to create and use its own identifiers.

In this chapter, we provide a survey on techniques introduced for handling inconsistencies in data integration, as for example the ones discussed in the previous paragraphs. More specifically, we present and discuss two group of techniques. The first group focuses on techniques for computing consistent query answers, and the second group focuses on the resolution of inconsistencies.

For the first group of techniques, we assume that the user specifies additionally a set of integrity constraints on the global schema. Because integrity constraints play an important role in the way the user formulates queries, it is essential that this information is incorporated into the processing. One easy methodology to do this is to remove from consideration any solutions that do not satisfy the integrity constraints. This naive approach may, however, easily lead to trivialization because even in very simple data integration setting, such as data merging, there is no consistent solution. Consequently, we focus on techniques for consistent query answers that adjust the semantics of queries to alleviate the possible impact of the inconsistencies on the query answers.

The second group of techniques focuses on the resolution of inconsistencies, and in particular on detecting and merging data fragments that describe the same real-world object. In its simplest form, this involves computing the similarity and resemblance between data fragments, and then merging the data fragments that have a similarity value exceeding a predefined threshold. The whole process is performed offline, and thus at run-time, query answering is performed over the resulted merged data. A significant amount of research proposals focusing on efficiently and effectively addressing this challenge already exist. They can be found in the literature under different names, such as merge-purge [46], deduplication [71], entity identification [59], reference reconciliation [30], or entity resolution [76].

The remaining chapter is organized as follows. Section 2 presents and discusses techniques related to consistent query answers, including mechanisms for the compact representation of repairs, query rewriting, and logic programs. Section 3 techniques related to the resolution of inconsistencies, and more specifically methods for computing atomic similarity, computing similarity between groups of data, collective techniques, scaling to large datasets, and dealing with uncertainty that is related to inconsistencies. Finally, Section 4 provides conclusions.

2 Consistent query answers

In this section we discuss the framework of consistent query answers introduced by Arenas et al. in [8] to alleviate the impact of inconsistencies in a database on the quality of query answers. We begin by recalling standard database notions (Section 2.1) and the framework of consistent query answers (Section 2.2). Next, we discuss exists methods of computing consistent query answers and outline complexity results that indicate inherent challenges laying in this task (Section 2.3).

2.1 Basic notions

We recall the standard notions of relational databases [1]. We assume a fixed *database schema* $\mathcal{S}$, which is a set of relation names of fixed arity. Every relation attribute is typed but for simplicity we assume two domains only: strings and rational numbers. We define in the standard fashion the *first-order language* $\mathcal{L}$ of formulas over $\mathcal{S}$ and the usual build-in comparison predicates ($=$, $\neq$, $<$, $\leq$, $>$, $\geq$ with their natural interpretation). A formula is: *closed* if it has no free variables, *ground* if it has no variables whatsoever, and *atomic* if it consists of one predicate only (other than the built-in predicates). In the sequel, we will denote: relation symbols by R, R_1, R_2,..., atomic formulas by A_1, A_2,..., tuples of constant by t, t_1, t_2,..., tuples of variables by $\bar{x}$, $\bar{y}$,..., and Boolean combinations of built-in predicates by φ.

A *database instance* I is a structure over $\mathcal{S}$ but often we will view I as a finite set of facts. An *integrity constraint* is any closed formula in $\mathcal{L}$. A database instance I is *consistent* with a set of integrity constraints Σ iff $I \models \Sigma$ in the standard model-theoretic way; otherwise I is *inconsistent*. We identify the following basic classes of constraints (all are closed formulas):

- *Universal constraints*: $\forall \bar{x} A_1 \wedge \ldots A_k \wedge \varphi \rightarrow A_{k+1} \vee \ldots \vee A_n$.
- *Tuple-generating dependencies*: $\forall \bar{x} A_1 \wedge \ldots A_k \wedge \varphi \rightarrow \exists \bar{y} A$. The dependency is *full* when there are no existentially quantified variables.
- *Denial constraints*: $\forall \bar{x} A_1 \wedge \ldots A_k \wedge \varphi \rightarrow$ **false**.

- *Functional dependencies* (FDs): $\forall\bar{x},\bar{y},\bar{y}',\bar{z},\bar{z}'.\ R(\bar{x},\bar{y},\bar{z})\wedge R(\bar{x},\bar{z},\bar{z}')\rightarrow\bar{y}=\bar{z}$ with a more common formulation $R: X\rightarrow Y$, where X and Y are the sets of attributes corresponding respectively to $\bar{x}$ and $\bar{y}$ (and $\bar{z}$).
- *Key constraints*, a special subclass of functional dependencies: $R : X\rightarrow Y$ is a key constraint if $X\cup Y$ is the set of all attributes of R. Key constraint $R : X\rightarrow Y$ is *primary* if it the sole constraint imposed on R.
- *Inclusion dependencies* (INDs): $\forall\bar{x},\bar{y}.\ \exists\bar{z}.R(\bar{x},\bar{y})\rightarrow P(\bar{y},\bar{z})$ with a common formulation $R[Y]\subseteq P[Y']$, where Y and Y' are the sets of attributes of respectively R and P that correspond to $\bar{y}$.

A *query* is a formula of $\mathcal{L}$ and we distinguish the class of *conjunctive queries* i.e., formulas of the form $\exists\bar{x}A_1\wedge\ldots\wedge A_k$. A tuple t is an *answer* to query q in an instance I iff $I\models q(t)$. In the sequel, we do not treat separately closed (i.e., Boolean) queries, but simply, we define **true** to be the answer of a closed query to be synonymous to the empty tuple () being the only answer to the query.

2.2 The framework of consistent query answers

The framework of consistent query answers is based on the notion of a repair of a (possibly) inconsistent database, which is essentially a consistent database instance minimally different from the original database instance. The original definition used the notion of symmetric difference between database instances to define acceptable repairs. Formally, the *symmetric difference* between two database instances I and I' is $\Delta(I,I')=(I\setminus I')\cup(I'\setminus I)$. Essentially, $\Delta(I,I')$ is the set of all facts that need to be either deleted or inserted to obtain I' from I. Now, given database instance I and two possible candidate repairs I' and I'', we use the symmetric difference to identify the candidate repair that is easier to obtain from I: essentially, I'' is closer to I than I' iff $\Delta(I,I'')\subset\Delta(I,I')$.

▶ **Definition 1.** Given a set of integrity constraints Σ and two database instances I and I', we say that I' is a *repair* of I w.r.t. Σ iff $I'\models\Sigma$ and there is no database instance I'' consistent with Σ and such that $\Delta(I,I'')\subset\Delta(I,I')$. By $\mathit{Repairs}_\Sigma(I)$ we denote the set of all repairs of I w.r.t. Σ. ◀

▶ **Example 2.** Take a simplified Muppet schema $\mathit{Muppet}(\mathit{Name},\mathit{Age})$ with one key constraint $\Sigma_0=\{\mathit{Muppet}:\mathit{Name}\rightarrow\mathit{Age}\}$. Consider an inconsistent database

$$\begin{aligned}I_0=\{&\mathit{Muppet}(\text{Miss Piggy},36),\mathit{Muppet}(\text{Miss Piggy},86),\mathit{Muppet}(\text{Miss Piggy},26),\\&\mathit{Muppet}(\text{J. Statler},73),\mathit{Muppet}(\text{J. Statler},83),\mathit{Muppet}(\text{Kermit},43)\}.\end{aligned}$$

I has 6 repairs w.r.t. Σ_0 that follow:

$$\begin{aligned}I_1&=\{\mathit{Muppet}(\text{Miss Piggy},36),\mathit{Muppet}(\text{J. Statler},73),\mathit{Muppet}(\text{Kermit},43)\},\\I_2&=\{\mathit{Muppet}(\text{Miss Piggy},86),\mathit{Muppet}(\text{J. Statler},73),\mathit{Muppet}(\text{Kermit},43)\},\\I_3&=\{\mathit{Muppet}(\text{Miss Piggy},26),\mathit{Muppet}(\text{J. Statler},73),\mathit{Muppet}(\text{Kermit},43)\},\\I_4&=\{\mathit{Muppet}(\text{Miss Piggy},36),\mathit{Muppet}(\text{J. Statler},83),\mathit{Muppet}(\text{Kermit},43)\},\\I_5&=\{\mathit{Muppet}(\text{Miss Piggy},86),\mathit{Muppet}(\text{J. Statler},83),\mathit{Muppet}(\text{Kermit},43)\},\\I_6&=\{\mathit{Muppet}(\text{Miss Piggy},26),\mathit{Muppet}(\text{J. Statler},83),\mathit{Muppet}(\text{Kermit},43)\}.\end{aligned}$$

◀

Intuitively, repairs represent (all) possible ways that the inconsistent database may be repaired. A consistent answer to a query is an answer that is present in every such possibility.

▶ **Definition 3.** Given an instance I, a set of integrity constraints Σ, and a query q, we say that a tuple t is a *consistent answer* to a query q in I w.r.t. Σ iff t is the answer to q in every repair of I w.r.t. Σ. ◀

Hence, if we take the query

$$q_0(x) = \exists y.\ \mathit{Muppet}(x, y) \land y \geq 65$$

asking for all Muppets eligible for senior discount, only J. Statler is the consistent answer to q_0 in I_0 w.r.t. Σ_0. On the other hand, *Miss Piggy* is not a consistent answer because of the repair I_1.

2.3 Computing consistent query answers

The main challenge in using the framework of consistent query answers lies in the fact that an inconsistent database may have an exponential number of repairs even for very simple sets of integrity constraints.

▶ **Example 4.** Fix $n \geq 0$ and consider a database instance over the schema $R(A, B)$:

$$I_n = \{R(1, 0), R(1, 1), \ldots, R(n, 0), R(n, 1)\}.$$

In the presence of a single key constraint $R : A \to AB$, the instance I_n has 2^n repairs. ◀

Consequently, a significant amount of research has been put into finding methods aiming to use the framework without materialization of all repairs. To identify classes of queries and integrity constraints for which this aim can be attained two basic decision problems have been proposed and their complexity studied: *consistent query answering* and *repair checking*. In virtually all research, the measure of *data complexity* has been adopted. This measure, widely adopted for relational databases [75], expresses the complexity of a problem in terms of the database size only, while the query and the integrity constraints are assumed to be fixed. The first decision problem allows to identify for which classes of queries and integrity constraints computing consistent query answers is tractable.

Consistent query answering Check whether *true* is the consistent answer to a given closed query in a given database w.r.t. to a given set of integrity constraints i.e., the complexity of the following set

$$\mathcal{D}_{\Sigma,Q} = \{I \mid \forall I' \in \mathit{Repairs}_\Sigma(I).\ I' \models Q\}.$$

We point out that the restriction to closed (Boolean) queries only does not make $\mathcal{D}_{\Sigma,Q}$ a special, simpler case of the more general problem of computing consistent query answers. Along the lines of [10] and [20], the treatment of an open query $q(\bar{x})$ can be reduced to a series of checks for closed query $q(t)$ with t ranging over some set of candidate tuples obtained by evaluating a simple derivative of $q(\bar{x})$. The second problem aims at identifying the complexity inherent to integrity maintenance.

Repair checking Check whether a database instance is a repair of a given database instance w.r.t. the given set of integrity constraints i.e., the complexity of the following set

$$\mathcal{B}_\Sigma = \{(I, I') : I' \in \mathit{Repairs}(I, \Sigma)\}.$$

This problem is a natural formulation of model checking for repairs and negative results highlight limitation of integrity enforcement mechanisms [2]. Another reason for the interest in this problem is its close connections to the data cleaning task. Finally, if the class of integrity constraints includes inclusions dependencies, then repair checking is know to be logspace-reducible to the complement of consistent query answers [19], which makes it an alternative tool for characterizing the complexity of consistent query answering.

Several different methods for computing consistent query answers have been proposed. They can be divided into three categories: *query rewriting*, *compact representation of all repairs*, and *logic programs*. We begin by presenting the first two approaches as they yield computing consistent query answers, and the aforementioned decisions problems, tractable for applicable classes of queries and integrity constraints. Next, we summarize a number of intractability results, which essentially precludes the use of approaches from the first two categories. The solutions in the third category use logic programming, a framework know to be capable of solving even problems complete for Π_2^p, and therefore more suited for handling difficult cases of consistent query answers.

2.3.1 Compact representation of all repairs

While approaches based on compact representation of all repairs has not been historically the first one, we begin with this direction because it allows to present some useful notions and tools. The most popular approach belonging to this category is based on the notion of the conflict graph (for FDs only). First, we define the notion of a conflict: two facts $R(t_1)$ and $R(t_2)$ are *mutually conflicting* w.r.t. a functional dependency $R : X \to Y$ iff $t_1[X] = t_2[X]$ and $t_1[Y] \neq t_2[Y]$.

▶ **Definition 5** ([10]). Given a database instance I and a set of functional dependencies Σ, the *conflict graph* of I w.r.t. Σ is a graph $G(I, \Sigma)$ whose set of nodes is I and edges connect pairs of mutually conflicting facts in I. ◀

The conflict graph corresponding for the instance from Example 2 is presented in Figure 1.

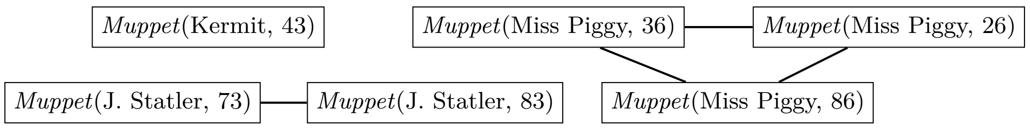


Figure 1 Conflict graph for the instance from Example 2.

The main reason for using conflict graphs lays in the simple observation that any maximal independent set of $G(I, \Sigma)$ is a repair of I w.r.t. Σ and vice versa. Let us recall that a maximal independent set of a graph is any maximal set of nodes containing no edge and note that any independent set can be extended to a maximal independent set.

The main use of conflict graph, and its variants, is to perform a repair existence check: given two sets of facts, required facts $\{A_1, \ldots, A_k\}$ and forbidden facts $\{A_{k+1}, \ldots, A_m\}$, check whether there is a repair that contains all required facts and none of the forbidden ones, i.e., a repair that satisfies the query $\Omega = A_1 \wedge A_k \wedge \neg A_{k+1} \wedge \ldots \wedge \neg A_m$. This test attempts to construct an independent set of nodes consisting of the required facts $A_1, \ldots, A_k$ and facts $B_{k+1}, \ldots, B_m$ blocking addition of facts $A_{k+1}, \ldots, A_m$, respectively. A fact B *blocks* addition of A if $\{A, B\}$ is an edge (i.e., A and B are conflicting) and thus the presence of B precludes the presence of A in the constructed instance. The test is performed by exhaustive enumeration of all combinations of edges adjacent to the forbidden facts. The test succeeds if an independent set is found, which implies the existence of a repair that satisfies Ω and consequently does not satisfy the following (disjunctive) Boolean query:

$$\Psi = \neg\Omega = \neg A_1 \vee \ldots \vee \neg A_k \vee A_{k+1} \vee \ldots \vee A_m.$$

This implies that **true** is not a consistent answer to Ψ. This check allows to compute consistent query answers to arbitrary Boolean quantifier-free queries: if we take a Boolean quantifier-free query in CNF $\Phi = \Psi_1 \wedge \ldots \wedge \Psi_n$, then **true** is not the consistent query answer to Φ if and only if there is some Ψ_i such that **true** is not consistent query answer.

This approach has been proposed by Chomicki and Marcinkowski [19] to handle denial constraints that requires a generalization of conflict graphs to conflict hypergraps. This algorithm is the basis of the Hippo system allowing to compute consistent answers to the class of projection-free SQL queries [21, 20]. The conflict hypergraph has been further extended to handle conflicts created in the presence of universal constraints. This work has been the basis of a polynomial time repair check algorithm for sets of denial constraints, join dependencies, and acyclic sets of full tuple-generating dependencies [72].

Another compact representation of all repairs is *nucleus* [77, 78]. In this approach all repairs are represented by a tableau (a table with free variables), and queries are evaluated in the standard way (answers with variables are discarded). We note that for some classes of constraints, constructing the nucleus may, however, require time exponential in the size of the input database.

2.3.2 Query rewriting

Query rewriting was the original approach proposed to compute consistent query answers, and, in principle, it functions as follows. Given a query $q \in \mathcal{Q}$ and a set of integrity constraints Σ, we construct a query $q' \in \mathcal{Q}'$ such that for any database I evaluating q' over I yields the consistent query answers to q in I w.r.t. Σ. This approach is parametrized by the class of integrity constraints (containing Σ) and the class of queries $\mathcal{Q}$ the user can use to formulate her queries but also the class of target language for the rewritten queries. Typically, $\mathcal{Q}'$ is richer and more expressive than $\mathcal{Q}$ but the query rewriting aims at using classes of target languages that enjoy efficient query evaluation (in terms of data-complexity), and consequently, the query rewriting yields efficient means of computing consistent query. Note that the *rewritten query* q', called often the *rewritten query*, is constructed independently of the database instance.

▶ **Example 6.** Recall from Example 2 the schema $\mathit{Muppet}(\mathit{Name}, \mathit{Age})$ and the key constraint $\mathit{Muppet} : \mathit{Name} \to \mathit{Age}$, and consider the query $q_0(x) = \exists y.\ \mathit{Muppet}(x, y) \wedge y \geq 65$. Note that the key constraint written as logic formula has the following form

$$\nexists x, y, y'.\ \mathit{Muppet}(x, y) \wedge \mathit{Muppet}(x, y') \wedge y \neq y'.$$

This formulation allows to identify for a fact $\mathit{Muppet}(x, y)$ the facts, $\mathit{Muppet}(x, y') \wedge y \neq y'$, that are conflicting with $\mathit{Muppet}(x, y)$ and may be present in a repair instead of $\mathit{Muppet}(x, y)$. Consequently, we wish to know if $\mathit{Muppet}(x, y)$ satisfying the query may be replaced in some repair by a fact $\mathit{Muppet}(x, y')$ that does not satisfy the query, i.e., $\mathit{Muppet}(x, y') \wedge y \neq y' \wedge y' < 65$. Together, we obtain the rewritten query

$$q_0'(x) = \exists y.\ \mathit{Muppet}(x, y) \wedge y \geq 65 \wedge \neg(\exists y'.\ \mathit{Muppet}(x, y') \wedge y \neq y' \wedge y' < 65).$$

◀

The fact that the rewriting is constructed independently of the database instance has its strong and weak points. On the one hand, this approach has no overhead in the architecture when adapting existing applications: it suffices to replace its queries by the rewritten versions. On the other hand, rewriting introduces a next level of complexity to the queries, which may have a negative impact on the performance of the system. It is also known that there exists relational queries that are not rewritable within the class of relational queries while computing their consistent answers is tractable.

Query rewriting was the first approach proposed to compute consistent query answers [8]. It uses the notion of *residues* obtained from constraints to identify potential impact of integrity violations on the query results. The residues are used to construct rewriting rules for the atoms used in the query. This approach has been shown to be applicable to quantifier-free conjunction of literals in the presence of binary universal constraints.

Chomicki and Marcinkowski [19] observed that if the set of constraints contains one FD per relation only, the conflict graph is a disjoint union of full multipartie graphs. This simple structure allows to construct rewriting for conjunctive queries without repeated relation names and no variable sharing. They also show that relaxing the conditions imposed on the queries and constraints leads to intractability: consistent query answering becomes coNP-complete.

The result of Chomicki and Marcinkowski has been further generalized by Fuxman and Miller [37] to allow restricted variable sharing (joins) in the conjunctive queries. The class C_{forest} of allowed queries is defined using the notion of *join graph* of a query whose vertices are the literals used in the query and an edge runs from a literal R_i to literal R_j if there is a variable that occurs on a non-key attribute of R_i and any attribute of R_j (both occurrences have to be different if $i = j$). The class C_{forest} consist of queries whose join graph is a forest, the joins are full and the join conditions are non-key to key.

Fuxman et al. [36], presented the ConQuer system that computes consistent answers to queries from C_{forest}. The queries can also use aggregates, and then *range*-consistent answers are computed [10]: minimal intervals containing the set of values of the aggregate obtained over the repairs. This allows the system to compute consistent answers to 20 out of 22 queries of the TCP-H decision support benchmark. The experimental evaluation of the system shows that the system performs reasonably well and is scalable w.r.t. both the size of the database and the number of conflicts in the database.

2.3.3 Complexity results

The rewriting scheme presented in [8] renders consistent query answering polynomial for quantifier-free conjunctive queries with negative atoms in the presence of binary universal constraints, which include functional dependencies and full inclusion dependencies. In a followup work, Cali et al. [17] showed that allowing arbitrary inclusion dependencies, together with functional dependencies, leads to undecidability. This large increase in complexity comes from the fact that a violation of non-full inclusion dependencies, caused by absence of a tuple, can be repaired by inserting a tuple chosen among a possibly infinite set of tuples. Furthermore, if the set of constraints has cycles, a cascading effect can occur.

▶ **Example 7.** Consider schema consisting of one relation symbol $R(A, B)$ and one (cyclic) inclusion dependency $R[B] \subseteq R[A]$, which written as a formula is $\forall x, y.\ R(x, y) \to \exists z.\ R(y, z)$. Now, take this inconsistent instance $I_0 = \{R(0, 1)\}$. The empty instance $I'_0 = \varnothing$ is one of repairs of I_0 but also for any $n \geq 1$ so is the instance $I'_n = \{R(0, 1), R(1, 2), \ldots, R(n - 1, n), R(n, n)\}$. Hence, not only does I_0 have an infinite number of repairs but also there is no bound on their size. ◀

One way to tackle the problem of infinite choice is to consider repairs obtained by deleting facts only, a setting studied in [19]. In the previous example, this yields only the empty repair $I_0' = \varnothing$. In this setting, the complexity of consistent query answering becomes Π_p^2-complete. Another approach proposed in [16] by Bravo and Bertossi uses a *null* value to instantiate the existentially quantified attributes in the facts to be inserted. The semantics of constraint satisfaction is adapted to the null value so that the presence of tuple with null value may satisfy the constraints but not violate it. For instance, the repairs of I_0 from the previous example obtained this way in this setting are the empty repair $I_0' = \varnothing$ and the repair $I_0'' = \{R(0,1), R(1, \mathit{null})\}$. On the one hand, the presence of $R(0,1)$ requires the presence of a fact of the form $R(1,y)$ and the fact $R(1, \mathit{null})$ fits the role perfectly. On the other hand, the presence of $R(1, \mathit{null})$ does not require the presence of any other fact.

There are two natural classes of constraints, universal dependencies and full tuple-generating dependencies, that similarly to full inclusion dependencies, may be violated by the absence of some tuples but repairing a violation requires choosing a tuple to insert from a finite set. A recent study by Staworko and Chomicki [72] showed that consistent query answering is Π_2^p-complete for arbitrary universal dependencies, coNP-complete for denial constraints and arbitrary full tuple-generating dependencies, and in PTIME for denial constraints, join dependencies, and acyclic full tuple-generating dependencies.

Establishing the computational complexity of consistent query answering has also served to determine the boundaries of query rewriting for consistent query answering. The data complexity of computing answers to relational queries is known to be in AC^0, a complexity class properly contained in P, and therefore, it is impossible for a relational query to express a coNP-hard problem. For instance, Chomicki and Marcinkowski have shown in [19] coNP-completeness of consistent answering to a conjunctive query in the presence of primary key constraints (i.e., one key constraint per relation), which precludes the applicability of rewriting for the full class of conjunctive queries. Because the class of conjunctive queries and the class of primary key constraints is most commonly found in practice, a considerable amount of effort has been put into finding a subclasses allowing tractable consistent query answering, e.g., Fuxman and Miller have proposed in [37] a practical subclass C_{forest} of conjunctive queries with tractable consistent query answering. This direction of research goes often together with an attempt of establishing a dichotomy for consistent query answers: essentially, finding a subclass of (conjunctive) queries containing only queries for which consistent query answering is either intractable or can be accomplished with query rewriting. An extension C^* of the class C_{forest} was believed to have this property, until very recently Wijsen has found otherwise [80]. Wijsen has also characterized sufficient and necessary conditions for first-order rewritability for a subclass acyclic conjunctive queries [79]. An interesting approach to the dichotomy question, based on structural properties of conflict graphs, is currently pursed by Pema [66].

As for repair checking, while the repair characterization based on the conflict (hyper)graph gives a PTIME repair checking for the class of denial constraints [19], adding arbitrary inclusion dependencies leads to intractability, and under the subset repair semantics (deletions only) repair checking is shown to be coNP-complete for functional dependencies and arbitrary inclusion dependencies. Various restrictions allow to bring the complexity back to PTIME, e.g., the class of functional dependencies and acyclic inclusion dependencies [19], the class of denial constraints and full tuple-generating dependencies [72], the class of weekly acyclic LAV depenencies [2], and semi-LAV dependencies [39]. Repair checking is also coNP-complete for the class of universal constraints [72].

2.3.4 Logic programs

Several different approaches have been developed to compute consistent query answers using logic programs with disjunction and classical negation [9, 11, 33, 41, 42, 74]. Approaches based on logic programs can be seen as a special case of query rewriting: essentially, we incorporate in the program that defines the original query, a special program that defines repairs. The main difference lays in the fact that evaluation of disjunctive logic programs is known to be Π_2^p-complete while query rewritting uses a target language with tractable query evaluation.

Virtually all approaches falling into the category of logic programs use disjunctive rules to model the process of repairing violations of constraints and stable models of the program correspond to the repairs of the inconsistent database. A query evaluated under the *cautious* semantics returns the answers present in every model, which naturally yields the consistent query answers.

▶ **Example 8.** Consider the schema $\mathit{Muppet}(\mathit{Name}, \mathit{Age})$ from Example 2 with the key constraint $\mathit{Muppet}: \mathit{Name} \to \mathit{Age}$. The repairing logic program consists of the following rules:

- *Triggering* rule which identifies conflicts and specify the possible repairing actions

$$\neg \mathit{Muppet}'(X,Y) \vee \neg \mathit{Muppet}'(X,Y') \leftarrow \mathit{Muppet}(X,Y) \wedge \mathit{Muppet}(X,Y') \wedge Y \neq Y'.$$

- *Stabilizing* rule which ensures that the constructed instance is consistent

$$\neg \mathit{Muppet}'(X,Y) \leftarrow \mathit{Muppet}'(X,Y) \wedge \mathit{Muppet}'(X,Y') \wedge Y \neq Y'.$$

- *Persistence* rule which copies facts from the original instance unless the fact has been banned by the repairing process

$$\mathit{Muppet}'(X,Y) \leftarrow \mathit{Muppet}(X,Y) \wedge \mathbf{not}\, \neg \mathit{Muppet}'(X,Y).$$

Note that this program uses the classical negation $\neg$ and the negation as failure **not**. Essentially, $\neg A$ means that it is known that A is not true while **not** A captures the assertion that it is not know whether A is true (or the failure of proving that A is true).

The program above is evaluated together with the facts present in the instance and the predicates used in the query need to be interpreted accordingly, e.g., the query $q_0(x)$ becomes

$$Q_0(X) \leftarrow \mathit{Muppet}'(X,Y) \wedge Y \geq 65.$$

There is an one-to-one correspondence between the stable models of this program and the repairs. For instance, the stable model corresponding to the repair I_1 of the instance I_0 (Example 2) is

$$\begin{aligned}\mathcal{M}_1 = \{&\mathit{Muppet}(\text{Miss Piggy}, 36), \mathit{Muppet}(\text{Miss Piggy}, 86), \mathit{Muppet}(\text{Miss Piggy}, 26),\\ &\mathit{Muppet}(\text{J. Statler}, 73), \mathit{Muppet}(\text{J. Statler}, 83), \mathit{Muppet}(\text{Kermit}, 43),\\ &\mathit{Muppet}'(\text{Miss Piggy}, 36), \neg\mathit{Muppet}'(\text{Miss Piggy}, 86), \neg\mathit{Muppet}'(\text{Miss Piggy}, 26),\\ &\mathit{Muppet}'(\text{J. Statler}, 73), \neg\mathit{Muppet}'(\text{J. Statler}, 83), \mathit{Muppet}'(\text{Kermit}, 43),\\ &Q_0(\text{J. Statler})\}.\end{aligned}$$

◀

The main advantage of using logic programs is the generality of this approach: typically arbitrary first-order (or even Datalog$^\neg$) queries are handled in the presence of universal constraints. Also, the repairing programs can be easily evaluated with existing logic program environments like Smodels or `dlv` [32]. We note, however, that the systems computing answers to logic programs usually perform grounding, which may be cost prohibitive if we wish to work with large databases. Another disadvantage of this approach is the fact that the class of disjunctive logic programs is known to be Π_p^2-complete.

These difficulties are addressed in the INFOMIX system [33] with several optimizations geared toward effective execution of repairing programs. One is *localization* of conflicts with identification of the *affected database* that consists of all tuples involved in constraint violations and all syntactically propagated *conflict-bound* tuples. Another optimization involves using bit-vectors to encode tuple membership to each repair and subsequent use of bitwise aggregate function to find tuples present in every repair. This optimization, however, may be insufficient to handle databases with large numbers of conflicts because typically the number of repairs is exponential in the number of conflicts.

Recently, this deficiency has been addressed with *repair factorization* [34]. Essentially, the affected database is decomposed into parts that are conflict-disjoint (no two mutually conflicting tuples are in separate parts). When computing consistent answers to a query only parts that are simultaneously spanned by the query are considered at a time. The presented experimental results validate this approach: the system computes consistent query answers in a reasonable time and is scalable w.r.t. the size of the database and the number of conflicts. Tests with up to 200^{1000} conflicts are reported.

3 Resolution of Inconsistencies

In this section, we present and discuss techniques that can be used for the resolution of inconsistencies. More specifically, we focus on inconsistencies arising from the use of different **representations** for describing the same real-world object, for example the same conference, person, or location. The techniques we present here aim at detecting such representations. Once detected, the representations with a similarity higher than a predefined threshold are merged together. The final results are used for replacing the original representations in the integrated data, and thus, query processing is performed over the merged data.

The following paragraphs present the techniques for resolution of inconsistencies grouped into five categories according to the data included in the representation (that are used during the processing): (i) atomic similarity techniques for comparing representations that are strings (Section 3.1); (ii) similarity techniques for comparing representations corresponding to groups of data (Section 3.2); (iii) collective techniques that also use inner-relationships between representations (Section 3.3); (iv) techniques for scaling the processing to datasets of large sizes (Section 3.4); and (v) dealing with the uncertainty that is related to the inconsistencies (Section 3.5).

Additional information related to existing techniques in this domain, can be found in surveys [28, 38, 35] and tutorials [54, 44].

3.1 Atomic Similarity Techniques

This category includes techniques that compute similarity when the representations are either a single word, or a small sequence of words. Few examples of representations for this category are: r_1="John D. Smith", r_2="J. D. Smith", r_3="Transactions on Knowledge and Data Engineering", and r_4= "IEEE Trans. Knowl. Data Eng.". As already discussed in Section 1, such differences in representations (i.e., single words or sequence of words) are a common situation that is typically resulted from misspellings, or naming variants due to the use of abbreviations, acronyms, etc. The merging of two such representations (e.g., “John D. Smith” with “J. D. Smith”) is performed when the technique detects high resemblance between the text values composing the representations.

The first group of techniques that belong to the category of atomic similarity techniques are based on the characters composing the string. These techniques compute the similarity between two representations (i.e., strings) as a cost that indicates the total number of the operations needed to convert the string of the first representation to the string of the second representation. The basic method of edit distance, named Levenshtein distance [56], counts the number of character deletions, additions, or modifications that are required for converting the first to the second string. The variations of this technique extends it with additional aspects, such as operation cost depending on the character’s location, consideration of additional operations, including open gap, and extend gap [60]. Jaro [49] computes similarity by considering the overlapping characters in the two strings along with their locations. It suitable to small strings, for instance first and last names. An extension of this technique is the Jaro-Winkler [81]. This technique gives higher weight to the prefix (i.e., first characters) of the string, and thus it increases the applicability of this approach to person names.

A second group of techniques are the ones that compute the similarity between collections of words. The basic techniques from this group are the Jaccard similarity coefficient, and the TF/IDF similarity [70]. Fuzzy matching similarity [18] is another technique of this category. It is a generalized edit distance similarity that combines transformation operations with edit distance techniques. Another method is the Soundex similarity. The Soundex method converts each word into a phonetic encoding by assigning the same code to the string parts that sound the same. The similarity between two words is then calculated as the difference between the corresponding phonetic encodings of these words. Finally, [23] and [15] describe and discuss an experimental comparison of various basic similarity techniques used for matching names.

Although, the existing techniques are successful in identifying similar representations, the idea of merging representations based on their string similarity is only partly correct, since the objects to which the context of these representations refer is totally ignored. For example, consider two representations for people with the exact same name. Using a similarity technique from this category would result in incorrectly merging the representation of these people. For this reason, these representations are typically used only as part of the initial steps of more sophisticated representations, in order to identify *potential merges*, which can be then further processed.

3.2 Computing Similarity between Groups of Data

In contrast to the previous category, the techniques of this category focus on dealing with representations that are composed by a group of data. Few examples of representations for this category are: r_1={“John D. Smith”, “male”, “United States of America”}, and r_2= {“J. D. Smith”, “male”, “USA”}. They extend techniques of the previous category since they combine basic string similarity with more complicated methodologies.

The first group of techniques for this category are those that consider the data of each tuple (i.e., record) as the representation. The approaches suggested in [53] and [22] concatenate all data composing each tuple and create a string. These strings are then compare using one of the string similarity techniques (Section 3.1). One of the most known techniques of this category is the merge-purge [46], aiming in identifying whether two relational records refer to the same real-world object. Merge-purge considers every database relation (i.e., record) as a representation. This approach first sorts the relations using the different available column names, and then uses the sorting to easy compare between similar information. The merging of records is performed according to the found resemblances.

The techniques proposed in [73] and [29] aim at matching representations by discovering possible mappings from one representation to another representation. More specifically, in [73] a mapping is identified by applying a collection of *transformations*, such as abbreviation, stemming, and initials. For the same purpose, Doan et al. [29] apply *profilers*, which are described as predefined rules with knowledge about specific representations. Profilers are created by various sources, such as domain experts, learned from training data, or constructed from external data.

Cohen et al. [24] use techniques for string similarity (presented in the previous category) to create techniques to adaptively modify the document similarity metrics. Li et al. [57] also focus in handling multiple types of representations, addressing the problem as this appears in the context of the text documents.

3.3 Collective Techniques

This category includes techniques that identify matches between two representations by using not only the information available in the specific representations but also related information from other representations. In particular, these techniques discover and exploit the inner-relationships that exist among all representations of the given data collection. These inner-relationships can be seen as links, or associations, between the representations and parts of the representation data. As an example consider co-authorship in publications, which is widely used by collective approaches. By knowing that a publication has α, β, and γ as authors, and another publication has β', and γ as authors, we can increase our belief that β describes the same author as β'. Thus, we now have two sources for computing the belief we have that authors β and β' describe the same real-world object: the first is that their strings are similar (computed using a technique for Sections 3.1-3.2), and the second is that both authors have a publication with γ author.

To capture the inner-relationships found inside a data collection, the techniques of this category model the collection into an intermediary structure. For instance, the technique in [6] uses dimensional hierarchies, and the techniques introduced in [13] and [52] use graphs. Ananthakrishna et al. [6] exploit dimensional hierarchies to detect fuzzy duplicates in dimensional tables. The hierarchies are build by following the links between the data from one table to data other tables. Representations are matched when the information along these generated hierarchies is found similar. Getoor et al. [13, 14] model the metadata as a graph structure. The nodes in this graph correspond to the information describing the representations, and edges are the inner-relationships between representations. The technique uses the edges from the graphs to cluster the nodes, and the clusters detected are then used to identify the common representations.

In [52, 51], the data collection is also modeled as a graph following a similar methodology as the previous methods. These techniques also generate other possible relationships to represent the candidate matches between representations. The additional relationships became edges that enhance the generated graph. Then, graph theoretic techniques are applied for analyzing the relationships in the graph and deciding the possible matches between representations. Other techniques follow a different methodology to create their internal supportive structures. In [65], the nodes represent the possible matches between two representations (and not one node representing one representation) and the edges the inner-relationships between the possible representation matches. The relationships from the structure are then used to decide the existence of nodes (matches between representations), and information encapsulated in identified matches is propagated to the rest of the structure.

Some of the proposed techniques of this category are from the area of metadata management. The TAP system [43] uses a process named *Semantic Negotiation* to identify common representations (if any) between the different resources. These common representations are used to create a unified view of the data. Benjelloun et al. [12] identify the different properties on which the efficiency of such a technique depends on, and introduce different techniques to address the possible combinations of the found properties.

Another well-know technique is the *Reference Reconciliation* [30]. Here, the authors begin their computation by identifying possible associations between representations by comparing their corresponding data. The information encoded in the found associations is propagated to the rest of the representations in order to enrich their information and improve the quality of final results. The approach in [5] is a modified version of the reference reconciliation algorithm that is focused on detecting conflict of interests in paper reviewing processes. The approach introduced in [48] models the resolution-related information a Bayesian network, and uses probabilistic inference for computing the probabilities of representation matches and for propagating the information between matching.

3.4 Scaling to Large Datasets

As noted in [35], applying processing to datasets of a large size can be achieved through data blocking, i.e., instead of comparing each representation with all other representations, the representations are separated into blocks, and only the representations of the same block are compared. The challenge is to create blocks of representations that are most likely to refer to the same real-world objects. The majority of the proposed techniques typically associate each representation with a *Blocking Key Value* (BKV) summarizing the values of selected attributes and then operate exclusively based on the BKVs.

For instance, the Sorted Neighborhood technique [45], sorts blocks according to their BKV and then slides a window of fixed size over them, comparing the representations it contains. The StringMap techniques [50] maps the BKV of each representation to a multi-dimensional Euclidean space, and employs suitable data structures for efficiently identifying pairs of similar representations. Alternatively, the q-grams based blocking presented in [40] builds overlapping clusters of representations that share at least one q-gram (i.e., sub-string of length q) of their BKV. Canopy clustering [58] employs a cheap string similarity metric for building high-dimensional overlapping blocks, whereas the Suffix Arrays technique, coined in [4] and enhanced in [27], considers the suffixes of the BKV instead. The technique in [62] introduces a mechanism for eliminating the redundancy of blocking methods by removing superfluous comparisons.

More recently introduced techniques based on blocking focused not only on scaling the resolution process to large datasets, but also on capturing additional issues related to resolution. Papadakis et al. [61, 63, 64] investigated how to apply the blocking mechanism on heterogeneous semi-structured data with loose schema binding. Among other, the authors introduce an attribute-agnostic mechanism for generating the blocks, and explain how efficiency can be improved through scheduling the order of block processing and identifying when to stop the processing. The approach introduced by Whang et al. [76], iteratively processes blocks in order to use the results of one block when processing other blocks, and thus include the advantages illustrated by collective approaches (i.e., discussed in Section 3.3). The idea of iteratively block processing was also studied in [67], which provided a principled framework with message passing algorithms for generating a global solution for the resolution over the complete collection.

3.5 Dealing with Uncertainty related to Inconsistencies

Uncertain data management approaches deal with a variation of inconsistency resolution. More specifically, they consider the existence of probabilities that model the belief related to the inconsistencies. For example, [68, 26] considers the existence of more than one representations (modeled as relational relations) for the same real-world object. Thus, for each real-world object the database contains a small set of possible-alternative representations, with each representation accompanied by a probability that indicates the belief we have that this is the correct representation.

The approach suggested by the Trio system [3] focuses on creating a database that support uncertainty along with inconsistency and lineage, while also dealing with duplicate tuples, i.e., representations. Dalvi and Suciu [25] follow the "possible worlds" semantics to introduce query processing for independent probabilistic data that model alternative matches between representations, and introduced a methodology for efficiently evaluating queries.

Dong et al. [31] investigate the use of the probabilistic mappings between the attributes of the contributing sources with a mediated schema. Applying this method on representations would have considered the possible mappings between the attribute names as given by contributing sources with a mediated schema S. This means that an attribute of representations α, β, and γ is mapped to an attribute from S with a probability to show the uncertainty of each mapping. The authors explain how answering queries over the mediated schema S can be performed using these mappings.

Andritsos et al. [7] do not focus on the schema information, as the approach presented in [31], but on the actual data. The authors assume that the duplicate tuples for each representation are given, for example as the results computed by a technique from Sections 3.1-3.3. Thus, all tuples describing alternative representations of the same representation have the same identifier. The tuples of the alternative representations are considered as disjoined, which means that only one tuple for each identifier can be part of the final resulted representation. The approach in [47] addresses more challenges of heterogeneous data. In particular, this approach does not assume that the alternative representations of representations are known, but that an representation collection comes with a set of possible linkages between representations. Each linkage represents a possible match between two representations and is accompanied with a probability that indicates the belief we have that the specific representations are for the same real-world object. Representations are compiled on-the-fly, by effectively processing the incoming query over representations and linkages, and thus, query answers reflect the most probable solution for the specific query.

4 Conclusions

In this chapter we elaborated on the management of inconsistencies in data integration. More specifically, we presented and discussed two group of techniques: (i) computing consistent query answers, focusing on mechanisms for the compact representation of repairs, query rewriting, and logic programs; and (ii) resolution of inconsistencies, focusing on methods for computing similarity between atomic values or groups of data, collective techniques, scaling to large datasets, and dealing with uncertainty that is related to inconsistencies.

References

1 S. Abiteboul, R. Hull, and V Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

2 F. Afrati and P. Kolaitis. Repair checking in inconsistent databases: Algorithms and complexity. In *ICDT*, pages 31–41, 2009.

3 P. Agrawal, O. Benjelloun, A. Sarma, C. Hayworth, S. Nabar, T. Sugihara, and J. Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

4 A. Aizawa and K. Oyama. A fast linkage detection scheme for multi-source information integration. In *WIRI*, pages 30–39, 2005.

5 B. Aleman-Meza, M. Nagarajan, C. Ramakrishnan, L. Ding, P. Kolari, A. Sheth, I. Arpinar, A. Joshi, and T. Finin. Semantic analytics on social networks: Experiences in addressing the problem of conflict of interest detection. In *WWW*, pages 407–416, 2006.

6 R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In *VLDB*, pages 586–597, 2002.

7 P. Andritsos, A. Fuxman, and R. Miller. Clean answers over dirty databases: A probabilistic approach. In *ICDE*, 2006.

8 M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

9 M. Arenas, L. Bertossi, and J. Chomicki. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming*, 3(4-5):393–424, 2003.

10 M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar aggregation in inconsistent databases. *Theoretical Computer Science (TCS)*, 296(3):405–434, 2003.

11 P. Barcelo and L. Bertossi. Logic programs for querying inconsistent databases. In *PADL*, pages 208–222, 2003.

12 O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1):255–276, 2009.

13 I. Bhattacharya and L. Getoor. Deduplication and group detection using links. In *LinkKDD*, 2004.

14 I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *DMKD*, pages 11–18, 2004.

15 M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23, 2003.

16 L. Bravo and L. E. Bertoss. Semantically correct query answers in the presence of null values. In *IIDB Workshop co-located with EDBT*, pages 336–357, 2006.

17 A Cali, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.

18 S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. Robust and efficient fuzzy match for online data cleaning. In *SIGMOD*, pages 313–324, 2003.

19 J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Information and Computation*, 197(1-2):90–121, February 2005.

20 J. Chomicki, J. Marcinkowski, and S. Staworko. Computing consistent query answers using conflict hypergraphs. In *CIKM*, pages 417–426, 2004.

21 J. Chomicki, J. Marcinkowski, and S. Staworko. Hippo: A system for computing consistent answers to a class of SQL queries. In *EDBT*, pages 841–844, 2004.

22 W. Cohen. Data integration using similarity joins and a word-based information representation language. *ACM Transactions on Information Systems (TOIS)*, 18(3):288–321, 2000.

23 W. Cohen, P. Ravikumar, and S. Fienberg. A comparison of string distance metrics for name-matching tasks. In *IIWeb co-located with IJCAI*, pages 73–78, 2003.

24 W. Cohen and J. Richman. Learning to match and cluster entity names. In *MF/IR Workshop co-located with SIGIR*, 2001.

25 N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. *VLDB Journal*, 16(4):523–544, 2007.

26 N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *PODS*, pages 1–12, 2007.

27 T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009.

28 A. Doan and A. Halevy. Semantic integration research in the database community: A brief survey. *AI Magazine*, 26(1):83–94, 2005.

29 A. Doan, Y. Lu, Y. Lee, and J. Han. Object matching for information integration: A profiler-based approach. In *IIWeb co-located with IJCAI*, pages 53–58, 2003.

30 X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD*, pages 85–96, 2005.

31 X. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

32 T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem-solving in dlv. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Springer, 2001.

33 T. Eiter, M. Fink, G. Greco, and D. Lembo. Efficient evaluation of logic programs for querying data integration systems. In *ICLP*, pages 163–177, 2003.

34 T. Eiter, M. Fink, G. Greco, and D. Lembo. Repair localization for query answering from inconsistent databases. Technical Report 1843-07-01, Institut Fur Informationssysteme, Technische Universitat Wien, 2007.

35 A. Elmagarmid, P. Ipeirotis, and V. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 19(1):1–16, 2007.

36 A. Fuxman, E. Fazli, and R. J. Miller. Conquer: Efficient management of inconsistent databases. In *SIGMOD*, pages 155–166, 2005.

37 A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. In *ICDT*, pages 335–349, 2005.

38 L. Getoor and C. Diehl. Link mining: a survey. *SIGKDD Explorations*, 7(2):3–12, 2005.

39 G. Grahne and A. Onet. Data correspondence, exchange and repair. In *ICDT*, pages 219–230, 2010.

40 L. Gravano, P. Ipeirotis, H. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001.

41 G. Greco, S. Greco, and E. Zumpano. A logic programming approach to the integration, repairing and querying of inconsistent databases. In *ICLP*, pages 348–364, 2001.

42 G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(6):1389–1408, 2003.

43 R. Guha and R. McCool. TAP: a semantic web platform. *Computer Networks*, 42(5):557–577, 2003.

44 O. Hassanzadeh, A. Kementsietsidis, and Y. Velegrakis. Data management issues on the semantic web. In *ICDE*, pages 1204–1206, 2012.

45 M. Hernández and S. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995.

46 M. Hernández and S. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

47 E. Ioannou, W. Nejdl, C. Niederée, and Y. Velegrakis. On-the-fly entity-aware query processing in the presence of linkage. *PVLDB*, 3(1):429–438, 2010.

48 E. Ioannou, C. Niederée, and W. Nejdl. Probabilistic entity linkage for heterogeneous information spaces. In *CAiSE*, pages 556–570, 2008.

49 M. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *American Statistical Association*, 84, 1989.

50 L. Jin, C. Li, and S. Mehrotra. Efficient record linkage in large data sets. In *DASFAA*, 2003.

51 D. Kalashnikov and S. Mehrotra. Domain-independent data cleaning via analysis of entity-relationship graph. *ACM Transactions on Database Systems (TODS)*, 31(2):716–767, 2006.

52 D. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In *SIAM SDM*, 2005.

53 N. Koudas, A. Marathe, and D. Srivastava. Flexible string matching against large databases in practice. In *VLDB*, pages 1078–1086, 2004.

54 N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006.

55 M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.

56 V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

57 X. Li, P. Morie, and D. Roth. Semantic integration in text: From ambiguous names to identifiable entities. *AI Magazine*, 26(1):45–58, 2005.

58 A. McCallum, K. Nigam, and L. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2000.

59 A. Morris, Y. Velegrakis, and P. Bouquet. Entity identification on the semantic web. In *SWAP*, 2008.

60 G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

61 G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, pages 535–544, 2011.

62 G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Eliminating the redundancy in blocking-based entity resolution methods. In *JCDL*, pages 85–94, 2011.

63 G. Papadakis, E. Ioannou, C. Niederée, T. Palpanas, and W. Nejdl. Beyond 100 million entities: large-scale blocking-based resolution for heterogeneous data. In *WSDM*, pages 53–62, 2012.

64 G. Papadakis, E. Ioannou, T. Palpanas, C. Niederée, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, (to appear).

65 Parag and P. Domingos. Multi-relational record linkage. In *MRDM Workshop co-located with KDD*, pages 31–48, 2004.

66 E. Pema. On the tractability ond intractability of consistent conjunctive query answering. In *Ph.D. Workshop co-located with EDBT/ICDT*, 2011.

67 V. Rastogi, N. Dalvi, and M. Garofalakis. Large-scale collective entity matching. *PVLDB*, 4(4):208–218, 2011.

68 C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

69 F. Rizzolo, Y. Velegrakis, J. Mylopoulos, and S. Bykau. Modeling concept evolution: A historical perspective. In *ER*, pages 331–345, 2009.

70 G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

71 S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *KDD*, pages 269–278, 2002.

72 S. Staworko and J. Chomicki. Consistent query answers in the presence of universal constraints. *Information Systems*, 35(1):1–22, 2010.
73 S. Tejada, C. Knoblock, and S. Minton. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*, pages 350–359, 2002.
74 D. Van Nieuwenborgh and D. Vermeir. Preferred answer sets for ordered logic programs. In *JELIA*, pages 432–443, 2002.
75 M. Vardi. The complexity of relational query languages. In *STOC*, pages 137–146, 1982.
76 S. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009.
77 J. Wijsen. Condensed representation of database repairs for consistent query answering. In *ICDT*, pages 378–393, 2003.
78 J. Wijsen. Database repairing using updates. *TODS*, 30(3):722–768, 2005.
79 J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*, pages 179–190, 2010.
80 J. Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Information Processing Letters*, 110(21):950–955, 2010.
81 W. Winkler. The state of record linkage and current research problems, 1999.

Algorithmic Techniques for Processing Data Streams *

Elena Ikonomovska[1] and Mariano Zelke[2]

1 **Jožef Stefan Institute**
Jamova cesta 39, 1000 Ljubljana, Slovenia
elena.ikonomovska@ijs.si

2 **Institute for Computer Science**
Goethe-University
60325 Frankfurt am Main, Germany
zelke@em.uni-frankfurt.de

Abstract

We give a survey at some algorithmic techniques for processing data streams. After covering the basic methods of sampling and sketching, we present more evolved procedures that resort on those basic ones. In particular, we examine algorithmic schemes for similarity mining, the concept of group testing, and techniques for clustering and summarizing data streams.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases streaming algorithm, sampling, sketching, group testing, histogram

Digital Object Identifier 10.4230/DFU.Vol5.10452.237

1 Introduction

The opportunity to automatically gather information by myriads of measuring elements proves to be both a blessing and a challenge to science. The volume of available data allows a problem examination to be more profound than ever before; climate prediction [31] and particle physics [17] are unthinkable without exploring mounds of data. However, the challenge is posed by the necessity to inspect these amounts of information. The particle physics experiment at the large hadron collider of CERN will soon produce data of a size of 15 petabytes annually [51] corresponding to more than 28 gigabytes on average every minute.

This challenge puts the basic principle of the traditional RAM-model, cf. [3], in question: It is unreasonable to take a main memory for granted that includes the whole input and allows fast random access to every single input item. On the contrary, for applications as the above ones massive input data must be processed that goes beyond the bounds of common main memories and can only be stored completely on external memory devices. Since random access is very time-consuming on these devices, traditional algorithms depending on random access show unfeasible running times.

Streaming algorithms drop the demand of random access to the input. Rather, the input is assumed to arrive in arbitrary order as an *input stream*. Moreover, streaming algorithms are designed to settle for a working memory that is much smaller than the size of the input.

Because of these features, streaming algorithms are the method of choice if emerging data must be processed in a real-time manner without completely storing it. In addition,

* This work was partially supported by the project "Foundations of data stream processing at FAIR computing" and the Slovenian Research Agency.

streaming algorithms can also benefit from their properties when processing data that is stored on large external memory devices. Compared to their slow random access, the output rates of such devices grow by magnitudes when the data content is dispensed in the order it is stored, i. e., as a stream that can be handled by a streaming algorithm.

There is a large variety of streaming algorithms. They vary in several aspects such as the number of passes that are permitted over the input stream, the size of the consumed memory, or the time required to process a single input item. Such algorithms can be deterministic or randomized, they might process streams comprising of numerical values, a graph's edges, coordinates of points, or parts of an XML document. For an overview of the rich literature on streaming algorithms, we refer the reader to [9] and [58].

This work covers basic algorithmic techniques that are utilized by a multitude of streaming algorithms. These techniques often form the foundation to which more sophisticated methods revert to. After giving the necessary definitions in Section 2, we present the techniques of sampling and sketching in Section 3 and Section 4, respectively. Then we build upon these ideas and present some more advanced algorithmic techniques for similarity mining in Section 5, the concept of group testing and its application to tracking hot items in Section 6, and techniques for clustering and summarizing data streams based on robust approximation in Section 7.

2 Preliminaries

Let U be a universe of n elements. Even though the members of U can be any objects, it is convenient to identify them with natural numbers, thus, we assume $U = \{1, 2, \ldots, n\}$. There are several kinds of *streams*; the most natural one is simply a sequence $a_1, a_2, \ldots, a_\omega$ of ω items where each item is an element of U. Elements of U may occur once in the stream, several times, or not at all. The sequence $2, 1, 2, 5$ serves as an example. Such a sequence is called a stream in the *cash register model*. Streams of the cash register model are widely considered in practice; a sequence of IP addresses that access a web server is a typical instance.

For an element j in our universe U, we can consider the number of occurrences of j in the cash register stream. This way, we get the *frequency* of j that is denoted as f_j. More formally, $f_j = |\{i : a_i = j,\ 1 \leq i \leq \omega\}|$, that is, the number of positions in the stream at which element j appears. If we know the frequency for every element in U, we can arrange a *frequency vector* $f = (f_1, f_2, \ldots, f_n)$ containing the number of occurrences in the stream for each $j \in U$ at the corresponding position. For our example stream, the frequency vector is $(1, 2, 0, 0, 1, 0, \ldots, 0)$ containing a zero for every element of U that is not part of the stream.

If we read a cash register stream item-wise from left to right, we can perceive this as gradual updates to the frequency vector of U: Starting with the all-zero n-dimensional vector, every a_i in the stream causes an increment of the corresponding vector entry by one. After processing the whole stream this way, the frequency vector emerges.

It is not hard to figure a generalization of the described update scheme: Instead of a single element $j \in U$ as a stream item incrementing f_j by one, we can imagine a stream item $(j, z) \in U \times \mathbb{Z}$. Such a pair in the stream changes f_j by the amount of z, i. e., f_j is increased or decreased by $|z|$ depending on the sign of z. A stream composed of such pairs is called *turnstile model* stream.

A turnstile stream represents a frequency vector f of U since it describes f_j for each $j \in U$ as the sum of all changes in the stream that are made on j. If for every prefix of the stream the represented vector consists of nonnegative entries only, we call this the *strict turnstile model.* The sequence $(3, 4), (2, 2), (5, 2), (1, 1), (5, -1), (3, -4)$ is an example for a strict turnstile stream that gives rise to the same frequency vector of U as the previously mentioned cash register example stream.

For the *non-strict turnstile model*, we allow the represented frequency vector to have negative entries as well. An example is given by the sequence $(2, -1)$,$(5, 1)$,$(3, -3)$,$(2, 3)$,$(1, 1)$,$(3, 3)$ representing the same frequency vector as previous examples.

It easy to imagine a strict turnstile stream as a sequence of insert/delete operations to a database. Every item (j, z) with positive (negative) z corresponds to inserting (deleting) item j $|z|$ times into (from) the database. The strict case applies here because at every moment no entry is deleted from the database that has not been inserted before. We will see the usage of this model in Section 6 when tracking frequent items in a database. As it turns out in Section 5, the non-strict model has applications when examining the similarity of two streams.

For some applications, it is common to use a certain restriction of the turnstile model. In the turnstile model, the stream is a sequence $(a_1, z_1), (a_2, z_2), \dots, (a_\omega, z_\omega)$ of pairs. Now let us assume that for each element in U there is exactly one pair in the stream and the pairs are ordered by the first component, that is, $a_i = i$ for every pair. Thus, we get a stream like $(1, z_1), (2, z_2), \dots, (n, z_n)$ of n pairs. Since every a_i is defined by its position in the stream, we can drop the a_i's and end up with a stream $z_1, z_2, \dots, z_n$. Such a stream is called a *time series model* stream and it represents a frequency vector of U in the most elementary way: It is just a sequence of the frequency vector entries written from left to right, i. e., $f_j = z_j$ for every $j \in U$.

The only time series stream possible representing the same frequency vector as previous example streams is the sequence $1, 2, 0, 0, 1, 0, \dots, 0$ of length n. The time series model has applications in areas like sensor networks or stock markets where periodical updates like measurements or share values are monitored. Each input item gives the observed value at the corresponding moment and characteristics of the stream describe the value's behavior.

For any given stream $a_1, a_2, \dots, a_\omega$, a *streaming algorithm* reads the stream item by item from left to right. It is forbidden to have random access to the stream. For the cash register and turnstile model, such an algorithm cannot make any assumptions on the item's order, that is, it must be prepared for any order. Furthermore, the size of the memory for a streaming algorithm is restricted: It must be sublinear in the size n of the universe and sublinear in the *cardinality* of the stream which is defined as $\sum_{j \in U} |f_j|$. We denote this cardinality by m. Notice that for the cash register model, m equals ω, i. e., the number of items in the stream. Hence, we will often write $a_1, a_2, \dots, a_m$ for an input stream in the cash register model omitting the ω. For the strict turnstile model, m is the total number of items that have been inserted and not deleted.

Apparently, we assumed all our streams to be finite, that is, we have a first item a_1 and a last item a_ω or a_m. That seems to contradict many applications; sequences of IP addresses accessing a web server or streams of operations to a database do not have a predefined end. However, from the perspective of a streaming algorithm—and this very perspective we take—the end of the stream is reached when the algorithm is queried about the stream. At this moment, the last item is fixed, that is, the finite initial segment of a potentially infinite stream is determined and framed as the object of investigation. However, the precise end of the stream may not be known in advance which serves as a challenge for a streaming algorithm that must must be prepared for answering a query about the stream at any moment.

It is important to note that this broad definition of a streaming algorithm spans a large spectrum of algorithms. There are streaming algorithms consuming a memory that is polynomially smaller than the input size, e. g. [42], others are content with a polylogarithmic amount, e. g. [6]. While one-pass algorithms [60] are designed for a single run over the input stream, there are also algorithms that read the input stream several times. Some of those multi-pass algorithms assume the input stream to be unchanged between different passes, e. g. [19], others have the ability to influence the order of the input items prior to every pass, e. g. [2]. However, in this work we restrict ourselves to the case of one-pass algorithms.

Since most streaming algorithms work in a randomized fashion, we utilize tools from probability theory for their presentation. For an introduction to this area as well as for definitions and properties of terms as expectation and variance, further for inequalities due to Markov, Chebyshev, and Chernoff, we refer the reader to [55].

3 Sampling

Generally, *sampling* denotes a rule-based process that selects a smaller number of items out of a larger group. It is easy to see that such an approach can be useful in the streaming context. In particular, if we assume the cash register model—and that is what we do for the whole section—the sampling approach smoothly applies: Out of the large group of all items $a_1, a_2, \ldots, a_m$ in the input stream, the algorithm chooses a group of size smaller, in most cases much smaller, than m to be kept in memory to consume a space sublinear in m. The idea is that at the end of the stream or whenever the algorithm is queried, it uses the memorized items, that is, the sample, to gain information about the whole stream. Of course, the accuracy of this information heavily depends on how well the sample represents the whole stream according to the query. We will see instances of representative samples in Section 3.1. To draw a characteristic sample is the challenge for any sampling approach.

Though there are some deterministic sampling methods in the area of streaming algorithms, e. g. [36, 62], the predominant part of sampling approaches in this area is randomized and hence subject of this chapter.

3.1 Reservoir Sampling

Assume we want to sample from the input stream $a_1, a_2, \ldots, a_m$ a single item s uniformly at random, that is, in such a way that the probability of being the sample is the same for every input item. Hence, we require $Pr[a_\ell \text{ is the sample } s] = 1/m$ for $1 \leq \ell \leq m$.

It is important to note here that we draw a uniform sample over all *input items* and not over the *elements* of the universe U. Hence, for our sampling purposes, we make a difference between two input items a_i and a_j as long as $i \neq j$, even if a_i and a_j denote the same element of U.

From the input stream $2, 1, 2, 5$ for example, we want to pick one of the four input items uniformly at random, that is, each with probability $1/4$, and we do not care that two of those items represent the same element of U. Of course, the universe element 2 is chosen as the sample with probability $2/4$ because of the two corresponding input items a_1 and a_3 while the universe elements 1 and 5 are each sampled with probability $1/4$. Indeed, this is intended since the element 2 occurs as twice as often as each of 1 and 5.

We see that an element's frequency of occurrence proportionally affects the probability for being the sample. Therefore, by drawing and examining samples we can try to deduce information about the frequency distribution of the input stream; an example for doing so is given later in this subsection.

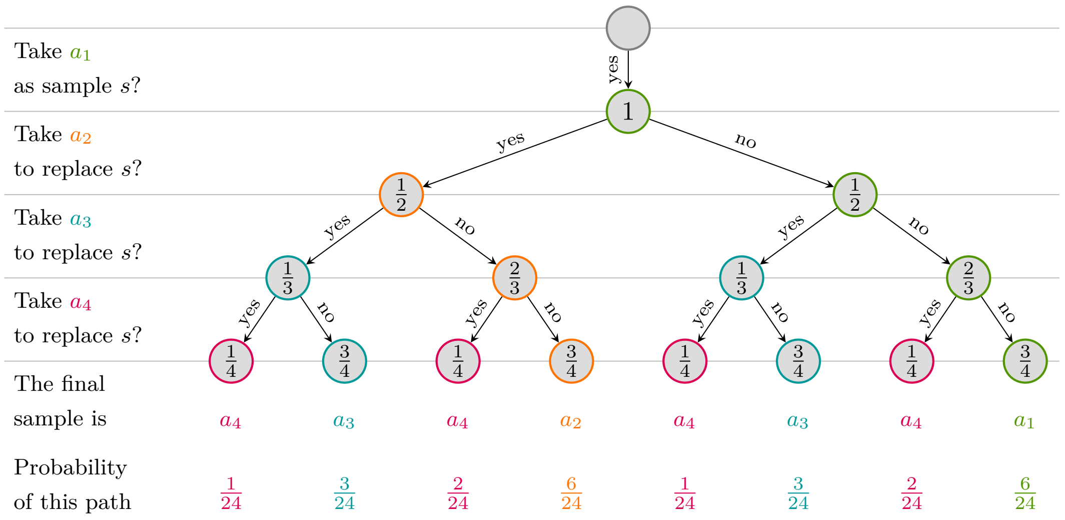

Take a_1 as sample s?
Take a_2 to replace s?
Take a_3 to replace s?
Take a_4 to replace s?
The final sample is
Probability of this path
yes
no
1
$\frac{1}{2}$
$\frac{1}{3}$
$\frac{2}{3}$
$\frac{1}{4}$
$\frac{3}{4}$
a_4 a_3 a_4 a_2 a_4 a_3 a_4 a_1
$\frac{1}{24}$ $\frac{3}{24}$ $\frac{2}{24}$ $\frac{6}{24}$ $\frac{1}{24}$ $\frac{3}{24}$ $\frac{2}{24}$ $\frac{6}{24}$

Figure 1 Decision tree for the reservoir sampling algorithm of stream a_1, a_2, a_3, a_4. The algorithm randomly decides to take or not to take ("yes" or "no") the considered input item as the actual sample s. Each node is labeled by the probability of the previous decision and its color indicates the item currently chosen as s. The probability of a specific path through this tree results from multiplying the probabilities along this path.

If we want to sample an input item uniformly at random from the stream and the length m of the stream is known in advance, this is a very simple task: Before reading the stream, the algorithm chooses a number $\ell \in \{1, 2, \dots, m\}$ uniformly at random; then it reads the stream until item a_ℓ which is picked as the sample. For the space consumption we note that the algorithm needs to generate ℓ and to memorize ℓ and a_ℓ, additionally it requires to count the number of stream items up to ℓ. Since a memory of $\mathcal{O}(\log m + \log n)$ suffices to do so and the stream is accessed sequentially, this method in fact describes a streaming algorithm using one pass.

However, the prior knowledge of m is a fairly unrealistic assumption. On the contrary, in most streaming scenarios the length of the stream is unknown beforehand or—even worse—there is no pre-defined end of the stream. Such continuous streams can arise from perpetual sensor updates; here, the unpredictable moment of a query marks the end of a stream on which the query needs to be evaluated.

It might come as a surprise that we are able to draw a uniform random sample without knowing the length of the stream. This approach is called *reservoir sampling* and is due to Vitter [63]. For every position ℓ in the stream $a_1, a_2, \dots, a_m$, it maintains an item s that is a uniform random sample over all items a_i, $i \leq \ell$, that is, over all items of the stream up to a_i. At the end of the stream, s is the final sample drawn from the whole input stream.

The algorithm starts by setting a_1 as s. In the following step, a_2 is chosen to replace a_1 as the sample with probability $1/2$; next, a_3 is picked as s with probability $1/3$. In general, for $i \geq 2$, a_i is memorized as s—and thereby replaces the previously stored item—with probability $1/i$.

Figure 1 shows the decision tree of the reservoir sampling algorithm for a stream of four input items. Each non-leaf node corresponds to a random decision of the algorithm whether or not to replace the actual sample s by the current input item. Every node is labeled by the probability of the preceding decision. Hence, the product of all labels along a path from the root to a leaf gives the probability for the specific sequence of decisions corresponding to this path.

As an example we calculate the probability for choosing the second input item a_2 as the final sample from the stream a_1, a_2, a_3, a_4. After reading the first input item, the algorithm chose a_1 as the actual sample. While reading a_2 in the next step, the algorithm picks a_2 as s with a probability of 1/2. To end up with a_2 as the final sample, the algorithm has to decide to select neither a_3 nor a_4 as the actual sample in the next two steps. The item a_3 is not picked with a probability of $(1 - \frac{1}{3}) = 2/3$; a_4 is not chosen with a probability of $(1 - \frac{1}{4}) = 3/4$. Item a_2 ends up as the final sample if and only if all mentioned events occur which happens with a probability of $\frac{1}{2} \cdot \frac{2}{3} \cdot \frac{3}{4} = \frac{1}{4}$. Similarly, one can calculate the same probability for the selection of each other input item as the final sample.

Another point of view on our small example is given by Figure 1: The probability just calculated for choosing a_2 as the final sample s corresponds to the decision sequence "yes", "yes", "no", "no" and thus to the path from the root to the orange leaf. However, while a_1 and a_2 each have a single corresponding leaf only, a_3 and a_4 correlate with several leafs, that is, decision sequences. Of course, the probability to end up with such items as the final s is the sum over the probabilities of the corresponding sequences. Eventually, we get a probability of 1/4 for every of the four input items.

The described reservoir sampling algorithm is certainly a streaming algorithm as it sequentially reads the input stream and only requires to memorize a single item. To convince ourselves that the choice for s at the end of the stream yields a uniform random sample, we look at the probability that some a_ℓ, $1 \leq \ell \leq m$, is the final s. This happens if a_ℓ is chosen as the actual s and additionally none of $a_{\ell+1}, a_{\ell+2}, \ldots, a_m$ replaces a_ℓ as the actual sample. For this probability, we have

$$\begin{aligned}
Pr[\,a_\ell \text{ is the final } s\,] &= Pr[\,a_\ell \text{ is chosen as } s\,] \cdot \prod_{i=\ell+1}^{m} Pr[\,a_i \text{ does not replace } a_\ell \text{ as } s\,] \\
&= \frac{1}{\ell} \cdot \prod_{i=\ell+1}^{m} \left(1 - \frac{1}{i}\right) \\
&= \frac{1}{\ell} \cdot \prod_{i=\ell+1}^{m} \frac{i-1}{i} \\
&= \frac{1}{m}
\end{aligned}$$

which means that any item a_ℓ ends up as the final sample with the same probability.

It is not hard to imagine a situation where we want to have a sample from the stream that is larger than only one item. Here, a natural approach is to run k parallel instances of the described procedure to get a random sample containing k items. As long as k is sublinear in m, the storage required for this method is sublinear in m as well. However, it is important to note that such a procedure results in a random sampling *with replacement* where each item in the sample is chosen from the whole stream of m items. Hence, an item from the stream can be selected more than once into the sample.

In contrast, it is often useful—as we will later see in this section—to get a random sample *without replacement* where the sampled group of k items is randomly selected from all $\binom{m}{k}$ subsets of size k that are possible over a set of m items. To get such a sample, the approach of reservoir sampling can be generalized as follows.

The first k items $a_1, a_2, \ldots, a_k$ in the stream are stored by the algorithm. For every subsequent item a_i, $k < i \leq m$, the algorithm decides to include a_i into the sample with probability k/i. If a_i is chosen for insertion into the sample, the algorithm picks an item uniformly at random among the k stored sample items that is replaced by a_i.

This generalized reservoir sampling approach reads the stream sequentially and only stores k items from the stream, hence, provided k is sublinear in m, that yields a streaming algorithm. It also achieves the promised uniformity of the random sample. To see this, we consider the probability that any k-item subset from the stream is chosen as the final sample.

For the sake of simplicity and because for different sample sizes the reasoning is along the same lines, we focus on the case where $k = 2$, that is, a sample containing two items is desired. Let a_{ℓ_1}, a_{ℓ_2} be any two items from the stream where $\ell_1 < \ell_2$. To end up with those items as the final sample, a bunch of events must occur: First, the algorithm selects a_{ℓ_1} into the sample. Second, every a_i, $\ell_1 < i < \ell_2$ is either not chosen into the sample or, if it is chosen, it does not replace a_{ℓ_1}. Third, the algorithm selects a_{ℓ_2} into the sample but does not replace a_{ℓ_1} by doing so. Finally, no a_j, $\ell_2 < j \leq m$, is chosen into the sample. We combine the probabilities for these necessary events to get the probability for obtaining a_{ℓ_1} and a_{ℓ_2} as the final sample:

$$\begin{aligned} Pr[\,a_{\ell_1}, a_{\ell_2} \text{ form the final sample}\,] &= \frac{2}{\ell_1} \cdot \prod_{\ell_1 < i < \ell_2} \left(1 - \frac{2}{i} + \frac{2}{i} \cdot \frac{1}{2}\right) \cdot \frac{2}{\ell_2} \cdot \frac{1}{2} \cdot \prod_{\ell_2 < j \leq m} \left(1 - \frac{2}{j}\right) \\ &= \frac{2}{\ell_1 \cdot \ell_2} \cdot \prod_{\ell_1 < i < \ell_2} \frac{i-1}{i} \cdot \prod_{\ell_2 < j \leq m} \frac{j-2}{j} \\ &= \frac{2}{(m-1) \cdot m} \\ &= 1/\tbinom{m}{2}\,. \end{aligned}$$

It follows that all $\binom{m}{2}$ subsets of 2 items from the stream are equally likely to show up as the final sample.

We now want to direct our attention to an application for the reservoir sampling approach. Consider a situation where we see a stream of items $a_1, a_2, \ldots, a_m$ and after the stream we are given an order-independent predicate. Every item in the stream does or does not satisfy such a predicate, but that is independent of the item's position in the stream. The task is to report the number of items in the stream that satisfy the predicate. As an example we can imagine for our stream of natural numbers the predicate of being a prime number. Note that the catch here is that the predicate is announced *after* the items passed by. Hence, we cannot simply count the number of items satisfying the predicate while reading the input stream because the subsequent predicate might as well ask for the number of items being equal to zero or being larger than twelve; we simply do not know the predicate while receiving the stream.

This problem is called *query selectivity problem* and it is of importance in the area of databases. Here, a user's query is usually unknown while an update stream is fed into the database. To select a fast evaluation method for a query, it is very useful to know the fraction of tuples that are retrieved in each evaluation step, that is, the fraction of tuples that are selected by some predicate, c. f. [59].

It is not hard to imagine that any algorithm that exactly solves the query selectivity problem for general streams and predicates requires the storage of the whole input stream. To be prepared for every possible subsequent predicate, no input item can be abandoned.

However, often we do not need an exact answer; we are instead—as for the query evaluation planning in a database—satisfied with an estimated one. For this, we can use the described random sampling approach: We draw a sample of size k uniformly at random from the input stream and get m by simply counting the number of input items. After the predicate is known, we simply determine the number k^+ of items in our sample satisfying the predicate. The number of items in the whole input stream meeting the predicate we estimate as $(m \cdot k^+)/k$. That is, we calculate the fraction k^+/k of items in our sample satisfying the predicate and estimate that the same fraction of all m items in the stream meet the predicate as well.

Using the reservoir sampling approach, we can draw a sample of arbitrary size uniformly at random. On the one hand, we want k to be large since that clearly increases the accuracy of the estimate. On the other hand, k also determines the space consumption of the sampling algorithm since every sampled item must be memorized, thus, we want k to be small. Recall that k needs to be sublinear in m to give rise to a streaming algorithm. So, what size do we need for k?

Let m^+ be the number of input items that satisfy the given predicate, i. e., the number we want to estimate, and let $m^+ = m/c$ for a constant c, $0 < c \leq 1$. Assume that we aim for a $(1 \pm \varepsilon)$ - estimate of m^+ with probability $1 - \delta$ where ε and δ are constants with $0 < \varepsilon, \delta < 1$. That is, we want to have our estimate within the interval $[(1-\varepsilon)m^+, (1+\varepsilon)m^+]$ with probability $1 - \delta$. The parameters ε and δ are chosen by the user and affect the space consumption of the algorithm.

Since a sampled item satisfies the predicate with probability $1/c$, the expected value for k^+ is k/c. To achieve a $(1 \pm \varepsilon)$ - estimate of m^+ with probability $1 - \delta$, we need k^+ to be within the $(1 \pm \varepsilon)$ - interval around its expected value with the same probability. By an application of the Chernoff Bound, we have

$$Pr\Big[\big|k^+ - Exp[k^+]\big| > \varepsilon \cdot Exp[k^+] \Big] < e^{-\Theta(\varepsilon^2 \cdot k/c)} .$$

Consequently, if we draw $k = \mathcal{O}(1/\varepsilon^2 \log(1/\delta))$ samples from the input stream, the probability that k^+ is outside the $(1 \pm \varepsilon)$ - interval around its expected value and therefore the probability that we over- or underestimate m^+ by more than an ε - fraction is at most δ.

We emphasize the fact that a constant number of samples suffices, a number that is independent of the stream's length m. The same number of samples is sufficient to estimate the median or other quantiles from a stream [53].

Since a single item from U can be memorized in $\log n$ bits and the counter for m uses $\log m$ bits, the described approach requires a memory of $\mathcal{O}(\log n + \log m)$ bits. The input items are processed in a sequential fashion, thus, the whole approach is a streaming algorithm.

However, we do not want to conceal that the quadratic dependence of the sample size k on ε makes the scheme impractical for very small values of ε. For those cases more sophisticated sampling approaches are known, e. g. [54], that reduce the dependence on ε.

3.2 AMS-Sampling

We recall that every stream $a_1, a_2, \ldots, a_m$ describes a distribution on the universe U; it conveys information about the number of occurrences for every $j \in U$. As defined in Section 2, f_j denotes the number of occurrences of the element j in the input stream, that is, the frequency of j. If we imagine the stream $2, 3, 3, 2, 3, 1, 2, 3$ as an example, we get $f_1 = 1$, $f_2 = 3$, $f_3 = 4$, and $f_b = 0$ for every $b \in U \setminus \{1, 2, 3\}$.

There are many approaches in the area of streaming algorithms to reveal the characteristics of frequencies: The average, minimum/maximum values, the median and other quantiles can be estimated [62], as well as the most frequent items [25], the fraction of rare items [28], and histograms [35].

For each $k \geq 0$ we define $F_k = \sum_{j=1}^{n} f_j^k$ to be the kth *frequency moment.* Apart from F_1, which simply equals the length m of the stream, the frequency moments provide meaningful parameters of a distribution. F_0 gives the number of distinct items in the stream which can be used to detect denial of service attacks [8]; F_k for $k \geq 2$ characterizes the skew of the distribution and is used for example by query optimizers in databases when join sizes need to be predicted [5]. For our example stream, we have $F_2 = 26$ and $F_0 = 3$ where the second equation tells us that the stream consists of three different items.

It is easy to compute all frequency moments exactly by maintaining a counter f_j for every single item j of the universe. But of course, the memory consumption of such an approach heavily depends on the distribution and can be proportional to m and/or n; no streaming algorithm can emerge from this scheme. However, there is no other method, elaborated or not, that gives rise to a streaming algorithm since it is known [6] that every algorithm that exactly computes F_k for $k \neq 1$ requires storage linear in m and n.

Consequently, we have to be content with an approximative solution. As earlier for the query selectivity problem in Section 3.1, we aim for an $(1 \pm \varepsilon)$ - estimation of F_k with probability $1 - \delta$. More formally, we want to have a solution that lies within the interval $[(1-\varepsilon)F_k, (1+\varepsilon)F_k]$ with probability $1 - \delta$. Again, it is the users choice to set the constants $0 < \varepsilon, \delta < 1$ affecting the precision and memory usage of the algorithm.

A method to estimate frequency moments in the desired accuracy using only sublinear space is the procedure of *AMS-sampling.* It originates from a celebrated paper [6] of Alon, Matias, and Szegedy, hence the name. The method works for all F_k with constant $k \geq 1$ and is a sample-and-count approach where a sample is maintained with additional data.

The core of the AMS-sampling is to pick an item a_i uniformly at random from all items $a_1, a_2, \ldots, a_m$ in the stream and to compute $r = |\{i' : i' \geq i, a_{i'} = a_i\}|$, i. e., the number of items equal to a_i occurring in the stream starting at position i. At the end of the stream a value X is calculated as $X = m(r^k - (r-1)^k)$.

We can use the reservoir sampling approach from Section 3.1 to select a_i. Whenever the reservoir sampling chooses a new item to be the actual sample s, a counter c is initialized to one; every subsequent item in the stream that is not chosen to be sampled increases c by one if it equals s. At the end of the stream, r is provided by c.

In order to compute X at the end of the stream, we need to store s, c, and a counter for m; for that, $\mathcal{O}(\log n + \log m)$ bits are sufficient which is also enough for the actual calculation of X. Since the input stream is processed sequentially, this gives rise to a streaming algorithm.

Let us assume that for our example stream $2, 3, 3, 2, 3, 1, 2, 3$ we have $k = 3$ and our randomly picked item is a_4, that is, the second occurrence of 2 in the stream. Since there are two items equal to a_4 occurring in the stream starting at a_4 (namely a_4 and a_7), it is $r = 2$. Using $m = 8$ as the length of the stream we get $X = 8(2^3 - 1^3) = 56$.

It is striking that the somewhat inscrutable value X in fact estimates the demanded F_k. Therefore, X is what we call an *unbiased estimator*, that is, a variable whose expectation equals—without any further transformations—the value in question. To see this, we compute the expected value of X. Let U' be the subset of the universe U containing the items that occur in the stream, i. e., $U' = U \cap \{a_1, a_2, \ldots, a_m\}$. For every item $j \in U'$, any of the f_j occurrences of j in the stream might be selected as the final sample of the reservoir sampling procedure. Thus, X takes the value $m((f_j - r + 1)^k - (f_j - r)^k)$ if the rth occurrence of the item j is chosen as the final sample. Every occurrence of every item in U' is selected as the final sample with uniform probability $1/m$, thus, the possible values of X corresponding to those selections emerge with the same uniform probability. For the expected value of X, that means

$$Exp[X] \;=\; \sum_{j\in U'} \sum_{r=1}^{f_j} \left(m\big((f_j - r + 1)^k - (f_j - r)^k\big) \cdot \frac{1}{m} \right) \;=\; \sum_{j\in U'} f_j^k \;=\; F_k\,.$$

Even though in expectation X equals the desired value, it is not enough to simply take X as an estimator for F_k. We cannot be sure that the probability of X lying outside the $(1 \pm \varepsilon)$ - interval around F_k is at most δ as demanded. Therefore, the probability that X lies within this interval needs to be increased, that is, we have to boost the concentration of X around its expectation. To this aim the authors of [6] make use of a technique that has become a standard by now and is presented in the following.

To increase the concentration of the random variable X around $Exp[X]$, the variance of X—which is a measure for expected deviation of X from $Exp[X]$—needs to be reduced. This can be done for independent and identically distributed random variables $v_1, v_2, \ldots, v_s$ by taking the average $v^* = \sum_{a=1}^{s} v_a/s$ of the individual v_a's. We then have $Var[v^*] = Var[v_a]/s$, that is, compared to a single random variable v_a, the variance of the average is reduced by a factor of s. Note that the expected value of the average is the same as for each individual random variable v_a.

Now the idea to enhance the quality of the estimator for F_k is obvious: Instead of taking a single X as an estimator for F_k, we run s_1 independent instances of the above approach for X in parallel to compute values $X_1, X_2, \ldots, X_{s_1}$. By taking the average Y of this values, we get an estimator for F_k which is more concentrated around its expectation, that is, around F_k, than any individual X_a, $1 \le a \le s_1$.

To get the number s_1 of parallel copies that are required, we utilize Chebyshev's inequality and can deduce that

$$Pr\big[\,|Y - F_k| \;>\; \varepsilon \cdot F_k\,\big] \;\le\; \frac{Var[\,Y\,]}{\varepsilon^2 \cdot F_k^2} \;=\; \frac{Var[\,X_a\,]}{\varepsilon^2 \cdot F_k^2 \cdot s_1} \qquad \text{for all } 1 \le a \le s_1\,.$$

The authors of [6] show that for all $1 \le a \le s_1$, it is $Var[X_a] \le kn^{1-1/k}F_k^2$. Thus, by choosing s_1 to be $8kn^{1-1/k}/\varepsilon^2$, we get the following inequality[1]:

$$Pr\big[\,|Y - F_k| \;>\; \varepsilon \cdot F_k\,\big] \;\le\; \frac{Var[\,X_a\,]}{\varepsilon^2 \cdot F_k^2 \cdot s_1} \;\le\; \frac{kn^{1-1/k}F_k^2}{\varepsilon^2 \cdot F_k^2 \cdot 8kn^{1-1/k}/\varepsilon^2} \;=\; \frac{1}{8}\,. \qquad (1)$$

Thus, the probability that Y is not a $(1 \pm \varepsilon)$ - estimation of F_k is at most $1/8$. Admittedly, we want this failure probability to be δ, not $1/8$. Of course, we could choose s_1 to be $kn^{1-1/k}/\varepsilon^2\delta$ instead of $8kn^{1-1/k}/\varepsilon^2$ to reduce the probability in (1) to δ. By doing so, s_1 would depend proportionally on $1/\delta$. But we recall that s_1 determines the memory consumption of the algorithm since every of the s_1 parallel instances of the sample-and-count approach needs to individually memorize a sample and a counter. Thus, a proportional dependence of s_1 on $1/\delta$ yields a handicap for applications where a very small failure probability is desired.

Fortunately, we can do better. Instead of taking a single Y as an estimator for F_k, we independently compute s_2 such values $Y_1, Y_2, \ldots, Y_{s_2}$ and take its median Z as our estimator for F_k. Every Y_b, $1 \le b \le s_2$ is the average of a separate group of s_1 X_a's as described above.

Let $Y^- = \{b : Y_b \notin [(1-\varepsilon)F_k, (1+\varepsilon)F_k],\ 1 \le b \le s_2\}$ be the set of indices of those Y_b's that are not an $(1 \pm \varepsilon)$ - estimate of F_k. Because of (1), we know that in expectation these

[1] The constant 8 in the value of s_1 is used in the original work [6]; any constant greater than two suffices and only slightly changes the line of argumentation in the following.

Y_b's are at most a $1/8$ - fraction of all Y_b's, thus, $Exp\big[|Y^-|\big] \leq s_2/8$. If Z as the median of all Y_b's is no $(1 \pm \varepsilon)$ - estimate of F_k, it must hold that at least half of all Y_b's are no $(1 \pm \varepsilon)$ - estimate of F_k either. That only happens if the size of Y^- exceeds $s_2/2$, that is, it exceeds its expected value by at least $3s_2/8$. As a result, we can bound the failure probability for Z as

$$\begin{aligned} Pr\Big[Z \notin [(1-\varepsilon)F_k, (1+\varepsilon)F_k]\Big] &\leq Pr\Big[|Y^-| \geq Exp\big[|Y^-|\big] + 3s_2/8\Big] \\ &< e^{-\Theta(s_2)} \end{aligned}$$

where the second inequality follows from an application of the Chernoff bound. If we choose s_2 to be $\mathcal{O}(\log(1/\delta))$, the probability of Z being no $(1 \pm \varepsilon)$ - estimate of F_k is at most δ.

Altogether, our estimator for F_k is the value Z which is the median over s_2 independent Y_b's where each of those is the average of s_1 independent X_a's. Since every X_a requires the storage of a sampled item and a counter, the overall space requirement is $s_1 \cdot s_2 \cdot \mathcal{O}(\log m + \log n)$ bits which is $\mathcal{O}(kn^{1-1/k}(\log m + \log n)\log(1/\delta)/\varepsilon^2)$. Clearly, the presented scheme is a streaming algorithm as the memory consumption is sublinear in both m and n; additionally, a sequential access to the input stream suffices to realize the sample-and-count approach for the individual X_a's.

The presented achievement has been the foundation for a lot of work enhancing it. The biggest improvement is the reduction of the $n^{1-1/k}$ - factor in the presented space bound to an $n^{1-2/k}$ - factor [44]. This dependency on n is optimal, that is, cannot be decreased any further [11]. The dependency on ε^2 is impossible to reduce either [65].

All mentioned results hold for the $(1 \pm \varepsilon)$ - estimation with probability $1 - \delta$ of F_k for general k. It is interesting to note that if k is an integer with $0 \leq k \leq 2$, F_k can be estimated by a streaming algorithm using only $\mathcal{O}((\log m + \log n)\log(1/\delta)/\varepsilon^2)$ bits of memory [6]. For the special case of F_0, that is, the determination of the number of distinct elements in a stream, the work of Kane et al. [47] gives a space optimal algorithm.

3.3 Sliding Window Sampling

So far, we viewed all items in the input stream as being equally important, no matter how far the occurrence of an item dates back. That is fine for many applications; the design of a query evaluation plan in a database often is independent of the input item's chronological order. We have seen in the previous sections that the estimation of query selectivities or join sizes uses samples that are uniformly drawn from the whole input.

However, it is easy to come up with applications where recent items are more significant than older ones. A typical task is the prediction of a system's behavior in the future based on its current state; to determine the current state, the recent input is of importance. For instance, the Random Early Detection protocol RED [34] is used within Internet routers to anticipate traffic bottlenecks by maintaining statistics over the recent queue lengths. Another example is to track calling patterns of phone company customers. To identify rapid changes in calling behavior, companies keep track of weighted averages where recent behavior is given a larger weight than older one [27].

The easiest way to model a different influence of older and newer input items is to simply define a stream's location ℓ such that all stream items a_i with $i \geq \ell$ are viewed as equally significant while all items a_i with $i < \ell$ are not considered at all. If we are interested in the recent w items, we have to increment ℓ for every incoming item. This approach is called *sliding window model*. Formally, instead of looking at the whole input stream $a_1, a_2, \ldots, a_t$, where a_t is the last item arrived, we only look at the items $a_{t-w+1}, a_{t-w+2}, \ldots, a_t$. We can visualize this scheme as a window of size w that slides over the input stream from left to right, hence the name.

There has been work in the sliding window model tackling problems that are known from the ordinary data stream model. In [23] different statistics and histograms of a sliding window are computed; [28] shows how to estimate the fraction of rare items and the similarity of different streams. We will take a look at such applications in Section 5.2. In the present section, we want to focus on the problem of maintaining a sample from the sliding window, a problem which serves as a foundation for more evolved procedures.

The memory consumption of a streaming algorithm is constrained to be sublinear in m and n. Note that under this requirement, every algorithm that completely stores the content of the sliding window is a streaming algorithm as long as the window size w is sublinear in m and n. Since for larger w such an algorithm is unsuitable, we have to tighten our requirement accordingly. To yield practicable algorithms, we demand the memory usage of a sliding window algorithm to be sublinear in n and w. As well as in the ordinary streaming model, for most functions, their exact computation is impossible in this model [29] and we strive for approximative solutions.

We want to give a sliding window algorithm that maintains a uniform random sample of all items contained in the actual window. As in Section 3.1, we emphasize on the fact that the sample is drawn uniformly from all items, that is, all positions within the window, not from the universe elements that are part of the window. For the sake of simplicity, we aim for a sample of size one and comment on larger sample sizes at the end of this section.

It is obvious that the reservoir sampling approach alone is not sufficient for the sliding window context. Of course, we might be lucky and every actual sample arises from the actual sliding window. But it is more likely that the actual sample falls out of the window as it progresses; at that moment any algorithm using memory sublinear in w cannot construct a new uniform sample.

A tempting idea to overcome this might be the following: We use the reservoir sampling approach only over the first w stream items to draw a sample a_ℓ. By the properties of the reservoir sampling, for the first sliding window $a_1, a_2, \ldots, a_w$ the item a_ℓ is chosen uniformly at random. If in the following the sliding window moves on, we keep a_ℓ as the random sample until it falls out of the window, that is, if the item $a_{\ell+w}$ appears. At that particular moment, we take $a_{\ell+w}$ as our sampled item which in turn is replaced with $a_{\ell+2w}$ and so on. That way, the item $a_{\ell+c\cdot w}$, $c \in \mathbb{N}$, is the actual random sample as soon as it occurs in the stream. If we look at each sliding window individually, this approach indeed yields a sample that is drawn uniformly at random from the window content. However, the sample for different window positions is profoundly dependent: The place of a random sample for one window position completely determines the place of the random sample in all following window positions. Clearly, that is infeasible for many applications.

There is an algorithm for sampling items uniformly at random from a sliding window by Braverman et al. [12]. In the following, we want to investigate the simple sliding window algorithm maintaining a uniform random sample that is proposed by Babcock et al. [10] and uses a *priority sampling* approach. Every incoming item a_i is given a priority $p(a_i)$, that is, a random value chosen uniformly at random in the continuous interval between 0 and 1. The actual sample a_ℓ is given by the item that has the highest priority among all items in the sliding window.

It remains to see that this method can be realized within the memory constraints of the sliding window model. To this aim, we note that it is not necessary to store all items from the actual window. We only need to memorize those items whose priority is maximal among items that arrived later. This is because an item a_r will never be the sample if there is an item a_s with $s > r$ such that $p(a_s) > p(a_r)$. Hence, a_r can be abandoned. We use a linked list L that is ordered by decreasing priority to store the items that could be the actual sample in the future. Every input item a_i is processed by the random drawing of its $p(a_i)$, its insertion into L according to $p(a_i)$, and the deletion of all descendants of a_i in L. None of these descendants can become the sample anymore. Furthermore, the item a_{i-w}, that is, the item that fell out of the window on the arrival of a_i, is erased from L if it is part of L. Note that at any time, the actual random sample is given by the head of the linked list.

Let us examine the key question here: What is the length $|L|$ of the linked list L, i. e., how many items do we need to store? Clearly, the constitution of L depends on the item's priorities and so does $|L|$. It is interesting that in fact $|L|$—and therefore the memory consumption of the algorithm—is a random variable. That includes the chance that the random choices of the priorities force the memorization of all window items in which case $|L| = w$ and the model's memory constraint is violated. However, we will see that such a violation is very unlikely by an argumentation that is inspired by [7].

To this aim, we calculate the expected value of L's length, i. e., $Exp[|L|]$. Let $a'_1, a'_2, \ldots, a'_w$ be the w recent input items, that is, the items that are in the actual window where a'_i arrived before a'_{i+1}. Since a'_w becomes the end of L, $|L|$ is given by the number of ancestors of a'_w in L (where we define every item in L to be an ancestor of itself). Let $X_1, X_2, \ldots, X_w$ be indicator random variables such that $X_i = 1$ if a'_i is an ancestor of a'_w in L and $X_i = 0$ otherwise. The crucial observation is that a'_i is an ancestor of a'_w iff among all a'_h with $i \le h \le w$ the priority $p(a'_i)$ is maximal. Since for s independent identically distributed continuous random variables a fixed variable takes the maximum with probability $1/s$, we have $Pr[X_i = 1] = 1/(w-i+1)$. The X_i's are 0-1-random variables, thus $Exp[X_i] = Pr[X_i = 1]$. Due to the fact that a'_w cannot have ancestors with a later arrival time and by the linearity of expectation,

$$Exp\big[|L|\big] \;=\; Exp\left[\sum_{i=1}^{w} X_i\right] \;=\; \sum_{i=1}^{w} Exp\big[X_i\big] \;=\; \sum_{i=1}^{w} \frac{1}{w-i+1} \;=\; H_w$$

where H_w is the wth harmonic number bounded by $\ln w < H_w \le \ln w + 1$. Hence, in expectation, only a logarithmic number of items is stored in the linked list L. To see that with high probability no significant deviation from this expectation occurs, we apply the Chernoff bound on $|L|$ as a sum of indicator random variables having different distributions. According to this, for every constant $c \ge e^2$,

$$Pr\Big[\,|L| \;\ge\; c \cdot Exp\big[|L|\big]\Big] \;<\; e^{-c \cdot Exp[|L|]} \;=\; e^{-c \cdot H_w} \;<\; w^{-c}$$

which means that with high probability the length of the linked list is $\mathcal{O}(\log w)$.

It is important to note that for the above analysis we have to assume that all priorities are distinct. From a theoretical point of view that is no issue since two samples from a continuous interval differ with probability one. But a streaming algorithm using limited storage cannot memorize arbitrary real values from a continuous interval. To overcome this, we use a technique of [7]: The random priorities in the interval $[0, 1]$ are generated piecemeal by adding more and more random bits to their binary representation when required. We only use the priorities for comparisons; if for two compared priorities one binary representation is the prefix of the other, the representations are randomly enhanced until the comparison is decided. By [7], with high probability it suffices to generate—and memorize—only a constant number of bits for every priority.

After all, the described technique yields a streaming algorithm to draw from a sliding window of size w a sample that is uniformly distributed over all items in the window. The input items can be processed in any given order. Since with high probability $\mathcal{O}(\log w)$ items are memorized in the linked list and every memorized item requires $\mathcal{O}(\log n)$ bits of storage, the memory consumption is $\mathcal{O}(\log w \cdot \log n)$ with high probability.

It remains to enhance the algorithm for drawing more than one sample. As mentioned in Section 3.1, the execution of k parallel independent runs yields a sample of size k which is drawn with replacement. We can simulate sampling without replacement by executing additional independent runs to achieve at least k distinct samples among all samples with high probability. As long as k is sublinear in w, the number of required additional runs is sublinear as well [10].

4 Sketching

The challenge for a streaming algorithm is to make a space-efficient summarization of the input that allows to answer the given query at the end of the stream. In the previous section we have examined the summarization due to sampling which is space-efficient as only a limited number of input items are memorized. In this section a different technique called sketching is considered.

Recall that for the cash register model we assumed the input stream $a_1, a_2, \ldots, a_m$ to be a sequence of items where each a_i stems from a universe U of size n. We may regard this stream as an implicit, incremental update to a vector $f = (f_1, f_2, \ldots, f_n)$ of dimension n. Initially, f is the zero vector, i. e., $f_j = 0$ for all $1 \leq j \leq n$. Each input item a_i in the stream updates f by incrementing f_{a_i} by one and leaving all other vector entries untouched. Hence, after reading input item a_t, $1 \leq t \leq m$, the vector f is the *frequency vector* of the stream $a_1, a_2, \ldots, a_t$, that is, each vector entry f_j equals the number of occurrences of element $j \in U$ in $a_1, a_2, \ldots, a_t$ as previously defined in Section 2.

In the more general turnstile model, every item in the stream is a pair $(j, z) \in U \times \mathbb{Z}$. We can imagine that after each such pair (j, z), the frequency vector f is updated by adding z to f_j. Recall that a positive z corresponds to insertions of item j, a negative z to deletions. While in the strict turnstile model we assume all vector entries f_j of f to be non-negative at all times, in the non-strict case, the f_j's can be general values in $\mathbb{Z}$. In both cases, the cardinality m of the stream is given by the sum of the absolute values of all vector entries.

Since in the streaming context we cannot assume or exclude particular orders of the input items, every reasonable function to be calculated by a streaming algorithm is order-invariant. Note that such an order-invariant function on the input items could easily be computed using the frequency vector f at the end of the stream. Admittedly, no streaming algorithm can have f at its disposal because the memorization of a general n-dimensional vector requires at least n bits which violates the memory constraints of the streaming model. However, we could, in limited space, try to sketch f in a way that allows the approximation of the demanded function at the end of the stream.

That is exactly what the *sketching* approach does. It uses pseudo-random vectors of dimension n and computes the dot product of these vectors with the frequency vector f. In particular, if x is a pseudo-random n-dimensional vector, the dot product $f \cdot x^T$ is called a *sketch* of f. Usually, several sketches are used in combination to compute a—naturally randomized—approximation of the function in question.

There are two important reasons for the utilization of sketches in the streaming context: First, the sketch of the frequency vector f can be computed gradually while the stream items, that is, the incremental updates of f are processed. For every input item $(j, z) \in U \times \mathbb{Z}$, the sketch needs to be increased by $z \cdot x_i$ where x_i is the ith entry of x. Second, the sketch can be maintained in small memory. Since we assume the entries of x to be constants, the size of the sketch $f \cdot x^T$ is $\mathcal{O}(m \cdot n)$ which can be memorized in $\mathcal{O}(\log m + \log n)$ bits.

One of the first utilizations of sketches for the streaming context can be found in the seminal paper of Alon, Matias, and Szegedy [6]. Here, the authors improve their own result of approximating F_2—which we examined in Section 3.2—by exploiting sketching techniques. A further example of a sketching algorithm is the estimation of the number of distinct elements in a data stream [24].

4.1 Count-Min Sketch

To highlight the sketching approach's efficacy, we want to tackle the *point query* problem. It asks at the end of the stream for f_j, i. e., the number of occurrences in the input of an arbitrary item $j \in U$. Note that the problem corresponds to the index problem of communication complexity [50] which means that the storage required for an exact answer is $\Omega(n)$. That is not surprising; since the algorithm does not know j before the end of the stream, it has to prepare itself for every possible point query which for general streams implies to maintain a counter for every item of the universe. It is further known [50] that even a randomized algorithm with a reasonable error probability for this problem must use a memory of size linear in n.

Hence, any streaming algorithm must be satisfied with an approximative answer. We present such an algorithm in the following that utilizes the sketching approach, in particular, the *count-min sketch* proposed in [26]. Our aim is to answer any point query by giving an estimate $\widehat{f_j}$ of the queried f_j. For this estimate we demand that $f_j \leq \widehat{f_j} \leq f_j + \varepsilon \cdot m$ with probability $1 - \delta$. As usual, the constants $0 < \varepsilon$ and $0 < \delta < 1$ are selected by the user in a trade-off between desired precision and memory consumption of the algorithm.

For the ease of presentation, we focus in the following on our canonical cash register input stream $a_1, a_2, \ldots, a_m$ where each a_i is part of the universe U.[2] However, we emphasize that the presented count-min sketch smoothly applies to the strict turnstile model.

We let $w = \lceil 2/\varepsilon \rceil$ and set up an array of w counters $c(1), c(2), \ldots, c(w)$ initialized with zero. From a family of 2-universal hash functions that map from U to $\{1, 2, \ldots, w\}$, a function h is chosen uniformly at random. We comment on the usage and shape of such hash functions in Section 4.2. While reading the input stream, the algorithm updates the counters: For every input item a_i, the counter $c(h(a_i))$ is incremented by one. After reading the whole stream, the algorithm answers any point query by utilizing the counters. In particular, if f_j is in question, the algorithm provides $f'_j = c(h(j))$ as an estimate.

It is easy to see that $f'_j \geq f_j$ because every single occurrence of j in the stream increases the counter $c(h(j))$. However, we cannot expect $f'_j = f_j$ because the number of counters is smaller than the size of U, thus, collisions occur which affect the counter for item j to be counted over with different input items. Let us examine the key question of how many excessive increasings we have to reckon with.

For a fixed point query on f_j, let $X_1, X_2, \ldots, X_n$ be indicator random variables such that $X_i = 1$ if $h(i) = h(j)$ and $i \neq j$; otherwise $X_i = 0$. Intuitively, each X_i indicates if an item i different from j is hashed to the same counter by h. Since for the 0-1-variables

[2] Notice that this cash register stream corresponds to the stream $(a_1, 1), (a_2, 1), \ldots, (a_m, 1)$ in the turnstile model.

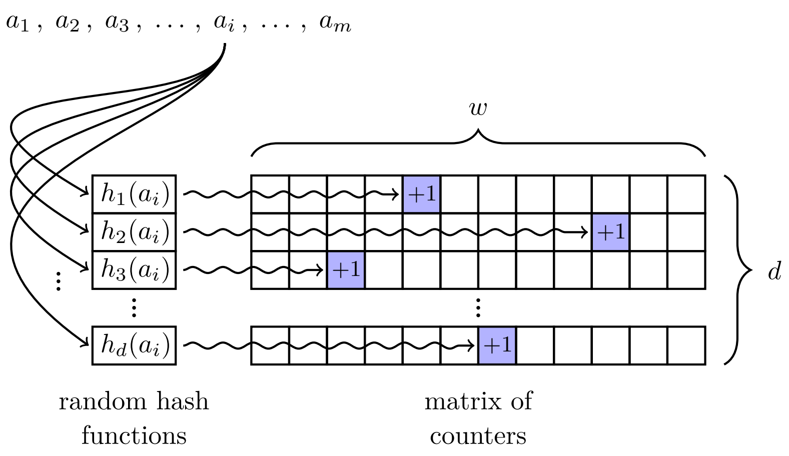


Figure 2 Update of the count-min sketch for input item a_i of the input stream. For each row $1 \leq s \leq d$, the assigned hash function h_s is evaluated for a_i. The result $h_s(a_i)$ indicates which column to increment in row s.

$Exp[X_i] = Pr[X_i = 1]$, we get

$$Exp[X_i] \;=\; Pr[\,h(i) = h(j)\,] \;=\; \frac{1}{w} \quad \text{for } i \in U \backslash \{j\} \text{ and } Exp[X_j] \;=\; 0 \tag{2}$$

where the first equality follows by the property of a function chosen randomly from a 2-universal family, see Section 4.2. Furthermore, we define $Y = \sum_{i=1}^{n} f_i X_i$ to be number of increasings to $c(h(j))$ that do not originate from j; hence, $f'_j = f_j + Y$. By linearity of expectation,

$$Exp[Y] \;=\; Exp\left[\sum_{i=1}^{n} f_i \cdot X_i\right] \;=\; \sum_{i=1}^{n} f_i \cdot Exp[X_i] \;\leq\; \frac{m}{w} \;\leq\; \frac{\varepsilon \cdot m}{2}\,.$$

Thus, in expectation the estimate f'_j exceeds the true value f_j by an amount of $\varepsilon \cdot m/2$. Using Markov's inequality, we can bound the probability that f'_j overruns f_j by a value greater than $\varepsilon \cdot m$ as

$$Pr[\,f'_j \;>\; f_j + \varepsilon \cdot m\,] \;=\; Pr[\,Y \;>\; \varepsilon \cdot m\,] \;\leq\; \frac{Exp[Y]}{\varepsilon \cdot m} \;=\; \frac{1}{2}\,. \tag{3}$$

To reduce this failure probability to the desired value δ, we run $d = \lceil \log(1/\delta) \rceil$ independent runs of the described algorithm in parallel. We can imagine this scheme as a $d \times w$ matrix of counters $c(s,t)$ with $1 \leq s \leq d$, $1 \leq t \leq w$ where each row has its own hash function h_s. It is important that each of those functions is chosen independently and uniformly at random from a family of 2-universal hash-functions. Every input item a_i causes an update in every row, i. e., for every $1 \leq s \leq d$, the counter $c(s, h_s(a_i))$ is incremented by one. The final estimate $\widehat{f_j}$ for f_j is the minimum value over the row's estimates, that is, $\widehat{f_j} = \min\{c(s, h_s(j)) : 1 \leq s \leq d\}$.

This whole scheme is called count-min sketch [26] based on its two main operations counting and minimizing. Figure 2 shows the update of the sketch for an input item a_i; the determination of the returned value $\widehat{f_j}$ is outlined in Figure 3.

It remains to certify the claimed quality of the count-min sketch's estimate. To bound the failure probability of providing an estimate $\widehat{f_j}$ that exceeds the true value f_j by more than

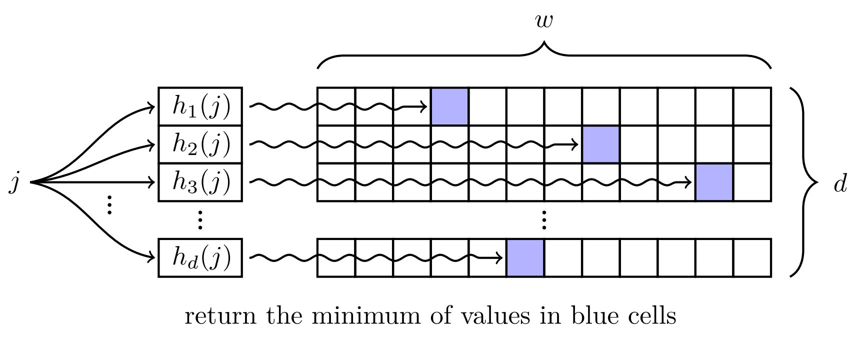


Figure 3 Count min sketch estimation of f_j. For each row $1 \leq s \leq d$, the assigned hash function h_s is evaluated for j. The result $h_s(j)$ denotes the column to consider in row s. Of all considered entries—indicated blue in this figure—the minimum is returned as $\widehat{f_j}$, that is, the estimation of f_j.

$\varepsilon \cdot m$, note that this happens iff the estimates of all rows overrun f_j by more than $\varepsilon \cdot m$ as well:

$$Pr[\,\widehat{f_j} > f_j + \varepsilon \cdot m\,] = Pr[\text{for all } s \in \{1, \dots, d\} : c(s, h_s(j)) > f_j + \varepsilon \cdot m\,] \leq 2^{-d} \leq \delta$$

where the first and second inequality is due to (3) and the choice of d, respectively. Thus, the failure probability is as desired.

Let us finally see that this procedure indeed yields a streaming algorithm. Obviously, the input stream is processed sequentially. For the memory usage of the algorithm we state that the hash functions consume a space of $\mathcal{O}(\log n \cdot \log(1/\delta))$; we postpone the reason for that to the next section. Any of the $d \cdot w$ counters can hold a value of at most m which yields an overall memory usage of $\mathcal{O}(\log m \cdot \log(1/\delta)/\varepsilon + \log n \cdot \log(1/\delta))$ bits satisfying the limits of the streaming model.

The utilization of the count-min sketch in the strict turnstile model instead of the cash register one is straightforward: For each stream item $(j, z) \in U \times \mathbb{Z}$, we add z to all counters that keep track of the occurrences of j. The estimation procedure for a query remains the same, as well as the answer guarantees and the memory consumption. However, for the non-strict turnstile model, the count-min sketch loses its ability to give an estimate $\widehat{f_j}$ that is an upper bound of f_j. This is due to the fact that the counters corresponding to j might underrun f_j because of colliding items with negative frequencies.

Finally, we want to highlight that the count-min sketch exploits its strength especially on a stream in the strict turnstile model. We can imagine such a stream as insert or delete operations to a database. As mentioned, with probability $1 - \delta$ the error of the sketch is $\varepsilon \cdot m$ where m now is the number of items currently in the database. As an example assume $\varepsilon = 0.1$ and $\delta = 0.01$, thus, the count-min sketch comprises $w \cdot d = 20 \cdot 7 = 140$ counters. We use it to track the insertion of a multiset containing a million ($< 2^{20}$) IP addresses into a database. Since each counter requires at most 20 bits, an overall of at most 2800 bits are used by the counters which is smaller than the input size by several orders of magnitudes. If now all but a multiset of nine addresses are deleted again, we can use point queries to the count-min sketch to reveal each of the remaining addresses and their frequencies exactly with a probability of 0.99 because with this probability the error for a point query is at most $9\varepsilon < 1$.

4.2 Universal Hash Functions

The sketching technique relies on projecting the frequency vector f along pseudo-random vectors to reduce the dimension of information to memorize. Often, this pseudo-random vectors are given implicitly by pseudo-random functions. We can see this in the case of the count-min sketch of the Section 4.1 where every single counter is a sketch. Here, a counter can be regarded as a dot product of f with a 0-1-vector that has a 1 at position i iff the hash function of the counter's row maps item $i \in U$ to the counter.

The reason for utilizing pseudo-random functions instead of completely random ones is the memory constraint of the streaming model; the memorization of completely random functions whose domain is the universe U is infeasible. This is due to the fact that in order to store a completely random function over the domain U we have to be prepared to store any function over U. That however requires the potential to memorize for each element in U which element of the target set is assigned to it; a memory of size $\Omega(|U|)$ is needed to do so.

In contrast, a suitable pseudo-random function combines some random-like properties with small required storage space. A basic family of such functions is the family of 2-universal hash functions.

As usual, $U = \{1, 2, \ldots, n\}$ is our universe and let $V = \{0, 1, 2, \ldots, q-1\}$ be a set with $q \leq n$. A family of hash functions $\mathcal{H}$ from U to V is said to be *2-universal* if, for all $x_1, x_2 \in U$ with $x_1 \neq x_2$, and for h chosen uniformly at random from $\mathcal{H}$ we have

$$Pr[\, h(x_1) = h(x_2) \,] \;\leq\; \frac{1}{q}\,. \tag{4}$$

This property reflects what we mean by a random-like behavior. It is something we expect a function to have that maps completely random from U to V, that is, assigns a completely random hash value to every item in U. Notice that the family of all functions from U to V satisfies this property as the random choice of any function from this family corresponds to a completely random mapping. However, since there are $|V|^{|U|} = q^n$ functions in this family, $\Omega(n \cdot \log q)$ bits are required to store such a function distinguishable from all others which exceeds the memory limitation.

However, there are families of functions that are 2-universal without being completely random. For a fixed prime $p > n$, we let $h_{a,b}(x) = (((ax + b) \bmod p) \bmod q)$ and define a family of hash functions as $\mathcal{H}' = \{h_{a,b} : 1 \leq a \leq p-1,\, 0 \leq b \leq p\}$. Each function h from this family is far from being completely random: The knowledge of two mappings $h(x_1)$ and $h(x_2)$ for $x_1 \neq x_2$ suffices to deduce $h(y)$ for every $y \in U$ while for a completely random function we can never deduce an unknown mapping from known ones. Anyway, $\mathcal{H}'$ can be shown to be 2-universal [55].

The crucial observation is that to memorize a function from $\mathcal{H}'$, we only need to store a, b, and p which can be done in $\mathcal{O}(\log n)$ bits[3]. Since the property of a 2-universal family is exactly what is needed for equality (2), we can utilize $\mathcal{H}'$ as the family of hash functions for the count-min sketch. For every row of the matrix of counters, a random function $h_{a,b}$ from $\mathcal{H}'$ is drawn by randomly selecting a and b within their respective bounds. For $\lceil \log(1/\delta) \rceil$ rows, $\mathcal{O}(\log n \cdot \log(1/\delta))$ bits are used to store the required hash functions of the count-min sketch.

In the original work presenting the count-min sketch [26], the authors choose the hash functions out of a family that is pairwise independent or strongly 2-universal. A family of

[3] Notice that by Bertrand's postulate, there is a prime p with $n < p < 2n$.

hash functions $\mathcal{H}$ from U to V is said to be *strongly 2-universal* or *pairwise independent* if, for all $x_1, x_2 \in U$ with $x_1 \neq x_2$, any $y_1, y_2 \in V$, and for h chosen uniformly at random from $\mathcal{H}$ we have

$$Pr[\,h(x_1) = y_1 \text{ and } h(x_2) = y_2\,] \;=\; \frac{1}{q^2}\,. \tag{5}$$

Note that this guarantee of pairwise independence between $h(x_1)$ and $h(x_2)$ is stronger than the one for the 2-universal family as property (4) follows from property (5). Even if the count-min sketch does not require this stronger guarantee, there are techniques used in the streaming area that do so or demand even stronger properties. The estimation of F_2 in [6] utilizes a family of strongly 4-universal hash functions where a family is strongly k-universal or k-wise independent if the hash values $h(x_1), h(x_2), \ldots, h(x_k)$ are mutually independent for all distinct $x_1, x_2, \ldots, x_k \in U$. For any constant k, there are constructions known [64] that yield a family of strongly k-universal hash functions from U to V where every function can be memorized using $\mathcal{O}(k \cdot \log n)$ bits. These small memory requirements make those functions a valuable tool for many streaming algorithms.

5 Similarity Mining

Estimating the similarity between two data streams is a basic problem in the data stream model and has many applications in mining massive streams of data, tracking changes in the network traffic, processing genetic data and query optimization. As an example, consider the problem of identifying similar entities (eg., web sites) based on the similarity between their corresponding data stream logs (IP addresses of their visitors, click-stream patterns, etc.).

An obvious solution to the problem of similarity estimation is to maintain a counter for each distinct item from the stream and compute the similarity at query time. Unfortunately, this solution requires $\Theta(n)$ words of storage, where n is the size of the universe U. As discussed previously, in the data streams scenario the dimensionality of the universe is typically very high, as well as the number of streams being analyzed. Very often we will not be able to afford the amount of memory which will be necessary in order to obtain exact answers. In such situations, one must refer to algorithms which will use bounded small amount of memory (polylogarithmic in the size of the universe), and will be able to produce high-quality approximations with high probability.

Among the most commonly used measures of similarity are the L_p distance and the Jaccard coefficient of similarity. In this section we will discuss algorithms for estimating these measures of similarity both in the *unbounded* data stream models (*time series model*, *cash register model* and *turnstile model*) and in the *windowed combinatorial* data stream model. The described algorithms build upon the basic mathematical ideas described in Section 3.3 and Section 4.

5.1 Estimating Similarity on Unbounded Data Streams

Random projections are an important mathematical idea used typically for an efficient dimensionality reduction over high cardinality domains. To this end many techniques have been proposed for computing various types of sketches, which rely on pseudo-random vectors generated by space-efficient computation of pseudo-random variables. The AMS-sampling and the count-min sketch described in Section 3.2 and Section 4 respectively are both based on the same general idea.

The idea of using multiple random projections is very general and works in the turnstile model as well. Similar powerful concept is based on generating a sequence of random variables each drawn from a stable distribution. Sketches based on different stable distributions are useful for estimating various L_p norms on the data stream, and form the basis of the algorithms presented in this section.

Another very useful mathematical tool is the family of min-wise hash functions whose properties enable a simple but efficient estimation of the Jaccard coefficient of similarity. Min-wise hashing has been used to estimate the similarity between two data sets representing various items in a market-basket analysis [22], and for estimating rarity and similarity in the *sliding window* or *combinatorial* data stream model [28].

5.1.1 L_2 and L_p Sketches

One of the most commonly used measures for data stream similarity is the L_p distance between two streams $A = (a_1, a_2, \ldots, a_m)$ and $B = (b_1, b_2, \ldots, b_m)$, where m is the length of the stream, while a_i and b_i are the actual i-th data elements of A and B. Here we consider the simplest *time-series* data stream model. For a real number $p \geq 1$ the L_p distance is defined by:

$$L_p = \sum_{i=1}^{m} |a_i^p - b_i^p|^{1/p}. \tag{6}$$

The the same definition applies to the *cash-register* and the *turnstile* data stream models with the difference that a_i and b_i would now represent updates on the counts of the corresponding stream elements $A[j]$ and $B[j]$, for $j \in \{1, .., n\}$.

The special cases of the L_p distance for $p = 0$ and $p \to \infty$ are defined as follows. The L_0 distance (also known as the Hamming distance) is the number of i's such that $a_i \neq b_i$, and measures the dissimilarity between two data streams. The L_∞ distance is the limit of L_p for $p \to \infty$ and is equivalent to the maximal difference at any time between any two items for the given data streams:

$$L_\infty = max_{i \in \{1,m\}} |a_i - b_i|. \tag{7}$$

There is a substantial amount of work done on estimating the L_1 [32, 24], the L_2 (Euclidean norm) [6, 43] and the L_p norm [43]. Feigenbaum et al. [32] were the first to produce a data stream algorithm for estimating the L_1 distance. Their technique relied on construction of pseudo-randomly generated "range-summable" variables which are *four-wise independent*[4]. The L_2 norm has been mostly used for estimating join and self-join sizes for the task of query selectivity estimation using only a limited storage. The earliest work for estimating the L_2 norm is the paper of Alon et al. [6], where they consider the simpler *cash-register* model. Their algorithms have been later extended in the work of [5] for handling the general sequence of insertions and deletions in the *turnstile* data stream model.

Alon et. al.'s technique for estimating the L_2 norm is based on the same concept described in Section 3.2. The main idea is to define a random variable which can be computed under the given space constraint, whose expected value is exactly the quantity we wish to estimate, and whose variance is relatively small. The final result is then obtained by considering

[4] The *four-wise independence* is defined as: the probability that a group of four random variables $\{\epsilon_1, \epsilon_2, \epsilon_3, \epsilon_4\}$ will map into a given combination of -1, +1 values, e.g., $\{-1, +1, +1, -1\}$, is equal to $1/2^4$=1/16.

sufficiently many such estimators, whose average is more concentrated around the expectation of a single estimator. By putting them into several groups, computing the average within each group, and taking the median of the group averages we get an estimator of the desired quantity whose variance is bounded by a user-defined parameter ε with a tunable probability of success 1 - δ, where δ is also specified by the user. Having the basic technique already described, here we will briefly outline only the main points of the algorithm.

Let $Z_{i,j}$ for $i = 1, 2, \ldots, s_1$ and $j = 1, 2, \ldots, s_2$ denote independent random variables defined as $Z_{i,j} = \sum_{v=1}^{m} \epsilon_v(a_v - b_v)$, where ϵ_v are 4-wise independent random variables that take on the values +1 or -1 with equal probability. Let $X_{i,j} = Z^2_{i,j}$. The interesting result is that, the expected value of the square of this quantity is the square of the L_2 distance we wish to estimate:

$$\begin{aligned} Exp[X_{i,j}] &= Exp\big[(\sum_{v=1}^{m} \epsilon_v(a_v - b_v))^2\big] \\ &= Exp\big[\sum_{v=1}^{m} \epsilon_v^2(a_v - b_v)^2 + \sum_{v \neq u} \epsilon_v\epsilon_u(a_v - b_v)(a_u - b_u)\big] \\ &= \sum_{v=1}^{m}(a_v - b_v)^2. \end{aligned}$$

The last equality follows from the fact that $Exp[\epsilon_v] = 0$ which will cancel out the second term, and $\epsilon_v^2 = 1$, as well as the independence of the random variables.

We now explain how to compute the variables $Z_{i,j}$, and hence $X_{i,j}$. Each $Z_{i,j}$ is initialized to 0. The resulting algorithm simply maintains the values of the random variables $Z_{i,j}$ after every update. Maintaining this value under continuous updates of the items a_v and b_v is straightforward: when an item with a value v arrives from stream A, we add ϵ_v to $Z_{i,j}$ for all i and j. If an item with a value v arrives from stream B, we subtract ϵ_v from $Z_{i,j}$ for all i and j. Practically, for each data item with value v we generate a mapping $h_{i,j}(v)$ from $\{1, 2, \ldots, m\}$ to $\{-1, +1\}$ which is being added/subtracted to the random variables $Z_{i,j}$.

The authors refer to this algorithm as *tug-of-war*, because each member of the sequence with a value mapping to +1 associates to pulling the rope in one direction, while each member with a value mapping to -1 associates to pulling the rope in the other direction. Note that this algorithm does not require a priori knowledge about the length of the sequence, or the number of distinct items seen from U at query time.

Let further Y_j be the average of $\{X_{1,j}, X_{2,j}, \ldots, X_{s_1,j}\}$ for all $j = 1, 2, \ldots, s_2$. Our final estimate is given with the value of Y which is the median of $\{Y_1, Y_2, \ldots Y_{s_2}\}$. As previously, parameter s_1 determines the accuracy of the results, i.e., the variance of the estimation, and parameter s_2 determines the confidence. By taking $s_1 = 1/\varepsilon^2$ and $s_2 = \log(1/\delta)$ the algorithm will need $\mathcal{O}(s_1 s_2) = \mathcal{O}(1/\varepsilon^2 \log(1/\delta))$ memory words.

Building upon the ideas in [6, 32], Indyk extended the previous results providing a unified framework for approximating the L_p distance between two data streams in small space, for any $p \in (0, 2]$ [43]. The method relies on the notion of *p-stable distributions*.

▶ **Definition 1.** A distribution D over $\Re$ is called *p-stable*, if there exist $p \geq 0$ such that for any n real numbers a_1, a_2, ..., a_n and i.i.d.[5] variables X_1, X_2, ..., X_n with distribution D, the random variable $\sum_i a_i X_i$ has the same distribution as the variable $(\sum_i |a_i|^p)^{1/p} X$ where X is a random variable with distribution D.

[5] independent and identically distributed

It is known that stable distributions exist for any $p \in (0, 2]$. In particular, the *Cauchy distribution* is 1-stable, and the *Gaussian (normal) distribution* is 2-stable, while for the general case $p > 2$, random variable X from a *p-stable* distribution can be generated by using the method of Chambers et al. [45].

The idea of using stable distributions enables us to use the previous approach for estimating the quantity $(\sum_i |a_i|^p)^{1/p}$ for any $p \geq 0$. The algorithm proceeds as previously by generating a number of i.i.d. random variables $X_{i,j}$, only this time drawn from a p-stable distribution D. The resulting random variables $Z_{i,j}$ will have "magnitudes" proportional to the "magnitudes" of the corresponding random variables $X_{i,j}$, which implies that the dot product can be used to approximate the value of the L_p distance. As previously one needs to repeat the procedure multiple times in parallel. The final estimate will be within a multiplicative factor $1 \pm \varepsilon$ of the true value, with probability of at least $1 - \delta$.

5.1.2 Min-wise Hashing

Another very popular measure of similarity is the Jaccard coefficient of similarity. Given two data streams A and B, let S_A denote the set of distinct items appearing in stream A, and let S_B denote the corresponding set for stream B. The Jaccard similarity between these two data streams is defined as:

$$\sigma(S_A, S_B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|}.$$

Since we are restricted on the space we can use the simplest approach of memorizing all the distinct items observed till the moment is not viable. Thus, we would have to do some sort of sampling, choosing not to memorize the appearance of some items. This equals to creating signatures of size $k \ll n$ bits for each set of distinct items, $Sig(S_A)$ and $Sig(S_B)$, which will be then used to compute an estimate of the similarity. Of course, the simplest way would be to sample the sets uniformly at random k times, using some of the techniques described above. However, due to sparsity this approach can miss important information, and as a result we would obtain a biased similarity estimate. This becomes obvious from the formula of the Jaccard coefficient, which shows that we are interested in the items that appear in both of the streams, while random sampling will not take into account this important fact.

Having in mind that streams are typically characterized with domains of a high cardinality, creating a space-efficient signature for each stream is not an easy task. Here we will present a very efficient way (in terms of memory and time) based on the concept of min-wise hashing [21]. Before explaining how it is possible to construct small signatures from large sets, it is helpful to visualize the collection of two sets S_A and S_B as their *characteristic matrix*.

Element	S_A	S_B
1	1	1
2	0	1
3	1	0
4	1	0
5	0	1

Figure 4 A matrix representing the sets S_A and S_B.

▶ **Example 2.** In Fig. 4 is an example of a matrix representing sets S_A and S_B chosen from the universal set $U = \{1, 2, 3, 4, 5\}$. Here, $S_A = \{1, 3, 4\}$, and $S_B = \{1, 2, 5\}$. The columns of the characteristic matrix correspond to the sets, and the rows correspond to elements of the universal set U from which elements of the sets are drawn. There is a 1 in row r and column c if the element for row r is a member of the set for column c. Otherwise the value in position (r, c) is 0. The top row and leftmost columns are not part of the matrix.

▶ **Definition 3.** Let π be a randomly chosen permutation over $[n] = \{1, 2, \ldots, n\}$. For a subset $S_A \subseteq [n]$ the *min-hash* of S_A for the given permutation π, i.e., $(h_\pi(S_A))$, is a mapping of the set S_A to the element $a \in S_A$ with $\pi(a) = min\{\pi(a')|a' \in S_A\}$

In other words, the min-hash of any subset is the is the number of the first row, in the permuted order, in which the column has a 1.

π_1	S_A	S_B
1	1	1
4	1	0
5	0	1
2	0	1
3	1	0

π_2	S_A	S_B
5	0	1
4	1	0
3	1	0
2	0	1
1	1	1

π_3	S_A	S_B
3	1	0
4	1	0
5	0	1
1	1	1
2	0	1

Figure 5 The matrices representing the sets S_A and S_B after the permutations π_1, π_2, and π_3 given in the corresponding order.

▶ **Example 4.** Consider the following 3 permutations for a universe of size $n = 5$, $U = \{1,2,3,4,5\}$: $k = 1$, $\pi_1 = (1\ 2\ 3\ 4\ 5)$; $k = 2$, $\pi_2 = (5\ 4\ 3\ 2\ 1)$; $k = 3$, $\pi_3 = (3\ 4\ 5\ 1\ 2)$, where k represents the index of the permutation. Although it is not physically possible to permute very large characteristic matrices, the min-hash function h implicitly reorders the rows of the matrix of Fig. 4 so it becomes one of the other matrices given in Fig. 5.

Given the sets $S_A = \{1, 3, 4\}$ and $S_B = \{1, 2, 5\}$ the min-hashes for each permutation are as follows: $k = 1$: $h_{\pi_1}(S_A) = 1$, $h_{\pi_1}(S_B) = 1$; $k = 2$: $h_{\pi_2}(S_A) = 4$, $h_{\pi_2}(S_B) = 5$; $k = 3$: $h_{\pi_3}(S_A) = 3$, $h_{\pi_3}(S_B) = 5$. The expectation of the fraction of permutations for which the min-hashes agree is an estimation of the Jaccard similarity between the sets S_A and S_B. In this case the fraction equals to $1/3 = 0.33$ which is not a very good estimation of the true value $1/5 = 0.2$. The quality of the estimate depends on the amount of memory we are willing to use, i.e., the size of the signature, that is, the value of k.

The wonderful and simple to prove property of min-hash functions is given with the following proposition:

▶ Proposition 1. *For any pair of subsets $S_A, S_B \subseteq [n]$*

$$Pr[h_\pi(S_A) = h_\pi(S_B)] = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} \pm \epsilon,$$

where the probability is defined over the random choice of the permutation π. The proof is given in [13, 14].

The on-line algorithm for estimating the similarity works in the following way: choose at random k permutations corresponding to k min-hash functions $h_1, h_2, \ldots, h_k$. Then at any time t maintain: $h_i^*(S_A) = min_{j \le t} h_i(a_j)$, and $h_i^*(S_B) = min_{j \le t} h_i(b_j)$ for $i = 1, \ldots, k$. This is simple to do in a streaming fashion: as each new element a_{t+1} from the first stream appears, the algorithm computes the min-hash $h_i(a_{t+1})$ for $i = 1, \ldots, k$, i.e., for all the initially chosen permutations $\pi_1, \pi_2, \ldots, \pi_k$. Then, it compares the computed min-hashes with their current minimum $h_i^*(S_A)$. The minimum is updated only if $h_i(a_{t+1}) < h_i^*(S_A)$.

In order to maintain the $h_i^*(S_B)$ values for the second stream, the same procedure is performed simultaneously in a similar manner using the same randomly chosen k permutations. The fraction of the min-hash values that they agree on, i.e., $\hat{\sigma}(S_A, S_B) = |\{i : h_i^*(S_A) = h_i^*(S_B)\}|/k$ can be easily computed at any time.

To obtain an unbiased estimate we need to repeat the calculation multiple times, each time choosing at random k permutations. In [33] it is shown that, for $k = O(\varepsilon^{-1}\log(1/\delta))$ the value $\hat{\sigma}(S_A, S_B)$ approximates the true Jaccard similarity with probability of at least $(1-\delta)$ within a multiplicative factor of $(1 \pm \varepsilon)$. More precisely, for $0 < \varepsilon < 1$, $0 < \delta < 1$ and $k \geq 2\varepsilon^{-3}\log\delta^{-1}$, with probability of at least $1-\delta$ holds:

$$\hat{\sigma}(S_A, S_B) \in (1 \pm \varepsilon)\frac{|S_A \cap S_B|}{|S_A \cup S_B|}.$$

The ideal family of min-hash functions is defined by the set of all permutations over $[n]$. Storing a single permutation from this family requires $\mathcal{O}(n \log n)$ bits; hence, they are not suitable for data stream applications. In [56] a family of *approximate* min-hash functions is presented such that any function from this family can be represented using $\mathcal{O}(\log n \log(1/\epsilon'))$ bits (each hash function being computed efficiently in $\mathcal{O}(\log(1/\epsilon'))$ time). This induces an additional error to the approximation of ϵ', for which only the k value will have to be appropriately adjusted. The advantage is in the savings of space and time.

5.2 Estimating Similarity on Windowed Data Streams

In many real-life scenarios the users are most interested on the most recent statistics or models gathered over the "recently observed" data elements. Despite the exponential growth in the storage capacity of the available systems, it is not common for such streams to be stored even partially. For example, consider high-speed, backbone Internet routers that route several Gbit/s and process tens of millions of packets per second on average. Storing the log of these packets locally even for an hour will require several MB of fast memory. Alternatively, moving it to a central warehouse would consume a sizable portion of the network bandwidth [28]. The windowed data stream model was formalized as a framework for designing algorithms, addressing the need of reasoning in this context.

Let us briefly recall the definition of the *combinatorial* [58] or *sliding window* data stream model defined previously in Section 3.3: At any time t consider the window of the last w observations $a_{t-(w-1)}, a_{t-(w-2)}, \ldots, a_t$, where each item a_i is a member of the universe of n items U. In this model we are allowed to ask queries about the data in the window using only $o(w)$ (often polylogarithmic in w) storage space.

Using sliding windows causes additional complications since maintaining simple statistics like minimum or maximum over a window of most recent data requires storing the most recent t items in the window, i.e., when a new item comes in, an old item is removed. Despite these difficulties, there are algorithms for estimating both the L_p distance [29] and the Jaccard similarity [28] over sliding windows of streaming data.

5.2.1 Approximating the L_p Norm

The work of Datar et al. [29] provides a general method for translating a wide range of data stream algorithms into the windowed data stream models, such as maintaining histograms, hash tables, distinct values and statistics or aggregates such as averages/sums.

Their technique is based on a special type of a histogram called *exponential histogram*, which is used to partition the window of w items into buckets. The main property of their algorithm is to maintain buckets with exponentially increasing sizes such that: there are at most $\frac{k}{2}+1$ and at least $\frac{k}{2}$ buckets of each bucket size, where $k = \lceil \frac{1}{\epsilon} \rceil$. Thus, whenever there are $\frac{k}{2}+2$ buckets of same size, the oldest two buckets are merged, which may occasionally lead to a cascade of such mergers.

Each bucket maintains the L_p sketches computed over the items it contains, but not the actual values of the items. Additionally, for each bucket a time-stamp is associated that marks the oldest *active* element in the bucket, and is used to indicate expiry. The expired buckets are deleted, and new ones are created for the newly encountered items. The absolute error of their estimate is due to the fact that the last bucket may contain items older than the last observation seen at time $t-(w-1)$. However, this error is bounded with an additive $(1+\epsilon)$ factor loss in accuracy for a multiplicative overhead of $\mathcal{O}(\frac{1}{\epsilon}\log w)$ in memory. The query time for the exponential histogram is $\mathcal{O}(1)$, and the worst-case processing time is $\mathcal{O}(w)$. For more details the reader is referred to [29].

5.2.2 Approximating the Jaccard Similarity

Computing an approximation of the Jaccard similarity in the windowed data stream model is not an easy task, primarily due to the problem of maintaining the minimum over a sliding window. The algorithm for approximating the Jaccard similarity described in section 5.1.2 requires at any time the correct minimal values for each hash function $h_i^*(A)$ $(h_i^*(B))$, for $i = 1, \ldots, k$ computed after every new observation a_t (b_t). To be able to maintain these minimums, one needs to store the outcome of all $h_i(a_j)$ $(h_i(b_j))$, where $j = t-(w-1), \ldots, t$. Thus, the problem boils down to maintaining the minimum hash values $h_i^*(t)$ for each random permutation at any time t for both of the streams. Datar and Muthukrishnan [28] provide a simple solution to this problem based on the idea of maintaining a linked list of hash values h_i and their timestamps only for the *dominant* items. Note, that we are interested only in the items which are appearing in the current window, i.e., with timestamps greater than or equal to $t-(w-1)$.

The property of *dominance* is defined as follows: Consider at time t two items d_1 and d_2 from the window of most recent w items with arrival times t_1, t_2 such that $t_1 < t_2 < t$. If $h_i(d_1) \geq h_i(d_2)$ then we say that item d_2 dominates item d_1. Thus, as long as there is an item d_2 that dominates an item d_1 both appearing in the window, we need not to store the hash value $h_i(d_1)$. It is easy to see that, at any time t if d_1 is in the window then d_2 is also present and has a hash value no greater than $h_i(d_1)$. Hence, the minimum h_i^* at time t will not be affected by the hash value of an item that is dominated.

Based on the observation above, the authors propose to maintain at any time t a linked list $L_i(t)$ for all $i = 1, \ldots, k$ permutations. Every element of this list will represent a pair of a hash value and its time-stamps $(h_i(a_j), j)$ for some data item a_j at time t, where j is the arrival time of the item and satisfies the property $t-(w-1) \leq j \leq t$. The list would look like:

$$\{(h_i(a_{j_1}), j_1), (h_i(a_{j_2}), j_2), \ldots, (h_i(a_{j_l}), j_l)\}, \text{ where } l \leq w.$$

The list satisfies the property that both the hash values and the arrival times are strictly increasing from left to right, i.e.,

$$j_1 < j_2 < j_3 < \ldots < j_l \text{ and } h_i(a_{j_1}) < h_i(a_{j_2}) < h_i(a_{j_3}) < \ldots < h_i(a_{j_l}),$$

where clearly $h_i^*(t) = h_i(a_{j_1})$.

Maintaining this list is simple. When a new data item arrives a_{t+1}, we compute its hash value $h_i(a_{t+1})$ and traverse the list $L_i(t)$ looking for the largest index j' (eg., a binary search over a special data structure) such that $h_i(a_{j'}) \leq h_i(a_{t+1})$. Then, we remove all the pairs from the list which appear after the pair with index j'. Now, $(h_i(a_{j'}),j')$ will be the rightmost item in the list $L_i(t)$:

$$j_1 < j_2 < \ldots < j' \text{ and } h_i(a_{j_1}) < h_i(a_{j_2}) < \ldots < h_i(a_{j'}),$$

If its hash value is different from $h_i(a_{t+1})$ then we insert the pair $(h_i(a_{t+1}),\ t+1)$ at the end of the list $L_i(t)$. Otherwise, we only need to update $(h_i(a_{j'}),j')$ into $(h_i(a_{j'}),t+1)$. The last step is to check if the leftmost pair corresponds to an item which is still present in the window. If $j_1 \notin \{(t+1)-(w-1),\ldots,t+1\}$ than the leftmost pair is deleted, and we get an updated list $L_i(t+1)$.

In the worst case this procedure will require a memory of $\mathcal{O}(w)$. However, with high probability, over the random choice of min-hash functions h_i the size of the list is proportional to the Harmonic number H_w, given by $1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{w}=\Theta(\log w)$. A proof of this bound is given in Section 3.3. Hence, the standard algorithm can be adapted to the windowed data stream model, using $\mathcal{O}(\log w+\log n)$ words of space and taking $\mathcal{O}(\log\log w)$ processing time per data item, with high probability. Note that, the success of the result above is predicated on the random choice of the random min-hash functions, and not over the distribution of the input, meaning that, it holds for an arbitrary (worst case) input.

6 Group Testing for Tracking Frequent Items

Tracking the *hot* items (those that occur frequently) on an underlying database relation or a data stream is a fundamental issue for the task of continuous query selectivity estimation, *iceberg* query computation or simple outliers detection in stream data mining. Hot items influence caching, load balancing, network traffic management, market-basket analysis and are crucial for a successful anomaly detection. As such, maintaining the set of hot items at any time is an interesting and an important problem. Formally, the question is how to dynamically maintain a set of hot items under the presence of delete and insert operations in the general *turnstile* data stream model.

Imagine that you observe a sequence (or a stream) of m operations on items, each member of a universe U of size n. Without loss of generality, we can assume that the item identifiers are integers in the range 1 to n. The net occurrence of any item x at time t, denoted $c_x(t)$, is the number of times the item x has been inserted minus the number of times it has been deleted. The current frequency of any item is thus given by:

$$f_x(t) = c_x(t)/\sum_{i=1}^{n} c_i(t).$$

The k most frequent items at time t are those with the k largest $f_x(t)$'s, where k is a parameter. On the other hand, an item x is called *hot item* if $f_x(t) > 1/(k+1)$, i.e., it represents a significant fraction of the entire dataset. Clearly, there can be at most k hot items, and there may be *none*.

▶ **Example 5.** Observe the following sequence of items: 1,2,1,3,4,5,1,2,2,3,1,1,3,5,2,6,1,2 each of them being a member of the set [1, 6]. Their corresponding frequencies are: $f_1 = 6/18$, $f_2 = 5/18$, $f_3 = 3/18$, $f_4 = 1/18$, $f_5 = 2/18$ and $f_6 = 1/18$. For $k = 3$, hot items are only 1 and 2 ($f_1 = 6/18 = 1/3 > 1/4$ and $f_2 = 5/18 > 1/4$).

6.1 Preliminaries

Determining the set of hot items is an easy problem if we are allowed a memory of $\mathcal{O}(n)$ words. Using a simple heap structure, we can process each *insert* or *delete* operation in $\mathcal{O}(\log n)$ time and find the hot items in $\mathcal{O}(k \log n)$ time in the worst case [4]. As discussed in several occasions for many streaming applications it is important to use sub-linear space $o(n)$ on the cardinality of the data stream. However, Alon et al. [6] proved that estimating $f^*(t) = max_x f_x(t)$ is impossible with $o(n)$ space. Thus, estimating the k most frequent items or the k hot items is at least as hard. A simple argument from information theory can help us show that solving this problem *exactly*, i.e., finding *all* and *only* items which have frequency greater than $1/(k+1)$ requires the storage of at least n bits.

This also applies to randomized algorithms. Any algorithm which guarantees to output all hot items with probability at least $1-\delta$, for some constant parameter δ, must also use $\Omega(n)$ space. This follows by observing that the above statement corresponds to the Index problem in communication complexity [50]. However, if we are willing to accept approximate answers it is possible to guarantee with high success probability at least $1-\delta$, that *all* hot items will be found and no item which has frequency less than approximately $\frac{1}{k+1} - \epsilon$, for some user-specified parameter ϵ and any user-specified probability δ [25].

6.2 Background

The problem of finding the most frequent items in one-pass with *limited* storage has gained a lot of interest in the last two decades. There is a large body of one-pass algorithms for finding the k most frequent items in the simpler data stream model in which only insert operations are allowed [30, 48, 52]. The general idea is to hold a number of counters (polylogarithmic in n), each associated with a single item seen in the sequence. The counters are incremented whenever their corresponding item is observed, but are decremented or deallocated only under certain circumstances. Therefore, they cannot be easily adapted to the dynamic case. As before, the algorithms guarantee that all hot items will be found, including items about which *no guarantees* of frequency can be made.

Another approach is to use filters: as each item arrives, the filter is updated to account for this arrival. Items which are above the threshold are retained as possible candidates for hot items. At output time all the retained items are rechecked with the filter, and those that pass the filter are output. Filter methods can only discover items when they become *hot* but cannot retrieve items from past which have since become *frequent* [25]. An important result that is of Charikar et al. [20], who gave an algorithm to approximate the count of any item correct up to ϵn in $\mathcal{O}(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ space and $\mathcal{O}(\log \frac{1}{\delta})$ time per update.

In the general turnstile model only the algorithms proposed by Cormode and Muthukrishnan [25] give theoretical space and time guarantees, which are outperformed in practice, as claimed. The algorithms use $\mathcal{O}(k \log k \log n)$ space for a summary data structure, and are able to process each transaction in $\mathcal{O}(\log k \log n)$ time. Querying the summary for finding the hot items takes $\mathcal{O}(k \log k \log n)$ time, which is independent on the size of the stream.

The approach of Cormode and Muthukrishnan is based on two different "group testing" procedures which can be categorized as *adaptive* and *nonadaptive*. Thus the methods are different in nature and give slightly different time and space guarantees. In the following subsection we will discuss the nonadaptive group testing method which is more efficient for the case of high transaction rates. For the adaptive group testing based method the interested reader is referred to [25].

6.3 Nonadaptive Group Testing

The general idea behind the algorithm is to randomly create $\mathcal{O}(k \log k)$ groups or sets of items, which are further deterministically divided or grouped into $\mathcal{O}(\log n)$ subgroups using error-correcting codes. Each group is associated with a counter which is incremented whenever an item that belongs to that group is inserted, or decremented when the item gets deleted. If a group contains a hot item then its corresponding counter will exceed a certain threshold. Thus, discovering the hot items requires an assembling of all the results from the tests performed over the different groups.

To ease the exposition of the algorithm, we will first describe a solution to the simpler problem of finding the *majority* (occurs more than half of the time) item. The algorithm for solving the latter problem will then be extended for the problem of finding k hot items.

6.3.1 Finding the Majority Item

While finding the majority item in the *cash-register* data stream model (where only insert operations are allowed) is easy, this problem looks less trivial for the *turnstile* data stream model. The reason is that an item which was found to be frequent, can become infrequent due to a sequence of delete operations. However, there exist a deterministic algorithm to solve this problem using $\lceil \log_2 n \rceil + 1$ counters. The algorithm maintains the set of counters at any time trough increment and decrement operations. To identify the majority item at output time a binary search procedure of $\log(n)$ steps is used.

The first counter $d_0 = c(t) = \sum_x c_x(t)$ keeps track of the number of items in total, i.e., we increment d_0 for every insert and decrement it for every delete operation. The remaining counters d_1, d_2, …, d_j are associated each with its corresponding jth bit of the binary representation for a given item identifier x in the range 1 to n. Thus, j goes from 1 to $\log n$. Let $bit(x, j)$ is a function that returns the value of the jth bit of the binary representation of the integer x. The update procedure is as follows:

On *insert(x)*: increment d_0 and update all counters c_j with $+bit(x, j)$, for $j = 1, \dots, \log n$
On *delete(x)*: decrement d_0 and update all counters c_j with $-bit(x, j)$, for $j = 1, \dots, \log n$

At output time the algorithm does a binary search over the set of counters. The logic is simple: if there is an item whose count is greater than $d_0/2$ (majority item), then for any way of dividing the elements into two sets, the set containing the majority item will have weight greater than $d_0/2$, and the other will have weight less than $d_0/2$. For example, if $c_1 > c_0/2$ (the least significant bit) means that the majority item is an item from the group of odd numbers. Next we will need to examine if it belongs to the group of items divisible with 4 or not, i.e., we need to test the value of c_2. In this way we proceed with examining the values of all the counters using the following procedure:

```
Initialize $x \leftarrow 0$
For $j = 1, \dots, \log n$, if $c_j > c_0/2$ then $x \leftarrow x + 2^{j-1}$
Output $x$
```

The algorithm described above guarantees always to find the majority item if there is one. If there is none such item, it will still return some item. Note, that in that case it will not be possible to distinguish the difference based only on the information stored.

6.3.2 Finding k Hot Items

Suppose we have selected a group of items to monitor which happened to contain only one hot item. Then we can apply the algorithm from the previous section to this group by dividing it further into $\log n$ buckets and associating a counter with each bucket. Determining the hot item in the group would require simple "weighting" of all the buckets. Provided that the total weight of other items in the group is not *too much* the hot item will always be in the heavier of the two buckets.

The idea is to divide each group into $\log n$ subgroups in which we will not hold an exact count for each separate item that is mapped to the group. This enables us to use space polylogarithmic in the size of the universe, albeit on the cost of an approximate answer. However, as we shall see this approximation can be very close to the correct answer. The choice of items belonging to each group can be done completely randomly, in which case we would have to store a list of members for every group explicitly (space at least linear in n). Instead, to create a concise description of each group, one may use hash functions which will provide a mapping of items to the groups. Each group will consist of all the items which are mapped to the same value by a particular hash function. The advantage of this approach comes from the possibility to store a concise representation for each hash function, using space $\mathcal{O}(\log n)$.

Let assume that we need W groups each divided further into $\log n$ subgroups. We need a hash function which will provide a mapping from the set of item identifiers $[1, n]$ to the set of group identifiers $[1, W]$. The hash functions used in the algorithm of Cormode and Muthukrishnan [25] are universal hash functions derived from those given by Carter and Wegman [16], and were discussed in Section 4.2. Briefly, each hash function is defined by a and b, which are integers smaller than P (initially chosen to be $\mathcal{O}(n)$) as: $f_{a,b}(x) = (((ax+b) \mod P) \mod W)$, where $P > n > W$ is a fixed prime, and a and b are drawn uniformly at random in the range $[0, P-1]$. As a result, the space required to store each hash-function representation is $\mathcal{O}(\log n)$ bits.

When a hash function is used to provide a mapping into a number of groups, there is a probability that two different items will be mapped to the same group. The authors make use of the following fact which comes from *Proposition 7* of [16]:

Over all choices of a and b, for $x \neq y$, $\Pr[f_{a,b}(x) = f_{a,b}(y)] \leq \frac{1}{W}$.

This probability is directly connected with the success probability of the algorithm for determining all k hot items. Therefore, we need to maximize it by using not one but several hash functions of this type. Lets assume that we will use T such hash functions. As a result we get $T \times W$ overlapping groups. For storing the representation of each hash function h_i we will need two arrays $a[1 \dots T]$ and $b[1 \dots T]$ whose values are chosen at random, having $h_i = f_{a[i],b[i]}$ for $i = 1, \dots T$.

The data structure which will be maintained at all times is a three-dimensional array of counters d, of size $T \times W \times (\log n + 1)$. In addition to that, we need a counter for the current total number of items seen m. The counters $d[1][0][0]$ to $d[T][W-1][\log n]$ are all initialized to zero. The counter $d[0][0][0]$ is used to keep count of the total number of items. Let $G_{i,j} = \{x | h_i(x) = j\}$ be the set of item identifiers which will be mapped to group $G_{i,j}$ by the hash function h_i, for $i = 1, \dots T$ and $j = 1, \dots W$. To keep the count of the current number of items within each group $G_{i,j}$ we will use the counters $d[i][j][0]$. For each such group we will need $\log n$ counters for $\log n$ subgroups defined as $G_{i,j,l} = \{x | x \in G_{i,j} \wedge bit(x, l) = 1\}$. These correspond to the groups used for finding the majority item. We will use $d[i][j][l]$ to keep count of the current number of items within subgroup $G_{i,j,l}$.

The update procedure is simple and very similar to the update procedure for finding the majority item, i.e., update the $\log n$ counters for each of the groups where an item x belongs to based on its bit representation in exactly the same way as in section 6.3.1. The time to perform an update $\mathcal{O}(T \log n)$ is the time taken to compute the T hash functions, and to modify $\log n$ counters for each of those T mappings. To output the hot items the structure can be searched at any time. The basic test wold be whether the count for a group or a subgroup exceeds the threshold needed for an item to be hot, which is $m/(k+1)$. A group containing a hot item will always pass this test, but the same is possible for a group which does not contain a hot item. Although this probability is very small various checks need to be made in order to reduce the number of items output which are not hot.

The search procedure consist of examining all of the groups and testing if they contain a hot item. That is, for a given group $G_{i,j}$, if $d[i][j][0] \leq m/(k+1)$ then there cannot be a hot item in that group, and the group is rejected. For the groups which are not rejected we need to examine the counts of their subgroups. If a group is not rejected, then there is enough information to discover the identity of the hot item x contained. At the end, the discovered hot item need to be further verified if it belongs to the group it was found in, and if all the groups where the item belongs are above the threshold, i.e., $d[i][h_i(x)][0] > m/(k+1)$ for all i. The total time to find all hot items is $\mathcal{O}(T^2 W \log n)$.

Cormode and Muthukrishnan [25] gave the following final result: *Choosing $W \geq 2/\epsilon$ and $T = \log_2(k/\delta)$ for a user-specified parameter δ ensures that, with probability at least $1-\delta$ we can find all hot items whose frequency is more than $\frac{1}{k+1}$, and for a given $\epsilon \leq \frac{1}{k+1}$, with probability at least $1-\delta/k$ each item which is output has frequency at least $\frac{1}{k+1} - \epsilon$.*

The proof is simple and is based on the property of the hash functions used and the Markov inequality, combined with some simple observations on the testing procedure. The interested reader is referred to [25] for more details. If we substitute the values for T and W from the above result into the previously given time and space bounds we will obtain the following bounds: the upper bound on the space required is $\mathcal{O}(\frac{1}{\epsilon} \log n \log(k/\delta))$, the update time takes $\mathcal{O}(\log n \log(k/\delta))$, and the query time is no more than $\mathcal{O}(\frac{1}{\epsilon} \log 2(k/\delta) \log n)$.

One of the drawbacks of the method is that the update time depends on the product of T and $\log n$, which can be slow for streams with high cardinality, i.e., large item identifiers. To reduce the time dependency on T each of the hash functions can be applied in parallel, and the relevant counts can be modified separately. The dependency on $\log n$ can be addressed by increasing the space usage. The observation is that, if instead of using the function $bit(x, i)$ one can use a function $dig(x, i, b)$ which gives the ith digit in the integer x when it is written in base $b \geq 2$. Then, within each group one will need to keep $(b-1) \times \log_b n$ subgroups: the i,j group now counting how many items have $dig(x, i, b) = j$ for $i = 1, \ldots, \log_b n$ and $j = 1, \ldots, b-1$. Setting b to m will correspond to keeping a count for every item.

7 Clustering and Summarizing Data Streams

In this last section we will discuss some more advanced algorithms for solving basic summarization problems in the singe-pass data stream scenario. We will place the focus on the *maximum error* histogram construction problem, although the techniques discussed here can be applied to other summarization problems like K-center, K-median clustering and VOPT histogram construction [39].

Histograms and related synopsis structures are popular techniques for approximating data distributions, and may serve as basic building blocks in the design of more sophisticated summary data structures for maintaining other statistics of interest. They have been extensively used in database query optimization for estimating selectivity factors [57] and access path selection in a relational database management system [61], in approximate query answering [1], mining time series [18], and many other areas.

With the increased interest into mining data streams several streaming algorithms for histogram construction problems have been proposed [38, 39, 41, 15, 49]. However, all of them have space bounds dependent on the size of the input m, the magnitude of the optimum solution ε^* or the machine precision M. A new result given by Guha [37] improves all previous algorithms on the problems of histogram construction and K-center clustering in either the space bound, the approximation factor or the running time. It represents the best algorithm applicable to streaming scenarios with *tunable* guarantees, linear running time and memory requirements independent of the input size.

In the following sections we will describe three main ideas used in the framework proposed by Guha [37]: (1) the notion of "thresholded approximation", (2) the idea of running multiple copies of the algorithm corresponding to different estimates of the final error and, (3) "streamstrapping" as a way to bootstrap the estimation procedure by using the summaries of the prefixes of the data to choose the correct granularity required to further inspect the data.

7.1 Maximum Error Histograms

Let $X = x_1, \dots, x_m$ be a finite data sequence of m real-valued numbers. The histogram construction problem is defined as follows: given some space constraint B, create and store a piecewise constant representation H_B of the data sequence using at most B storage (pieces), such that H_B is optimal under some notion of error $E_X(H_B)$ defined between the data sequence and H_B. The representation is a grouping of the values of consecutive points x_i, where $i \in [l_j, r_j]$ into a single value h_j, thus forming a bucket b_j, defined with the smallest l_j and the greatest r_j index of the data points belonging to the bucket, and their representative value h_j. In other words, for $l_j \le i \le r_j$ we estimate x_i by h_j. The histogram uses at most B buckets which cover the entire interval $[1, m]$, and saves space by storing only $\mathcal{O}(B)$ numbers instead of m.

Since h_j is an estimate for the values in bucket b_j, for the query at point i, where $l_j \le i \le r_j$ we incur an error $x_i - h_j$. The error $E_X(H_B)$ of the histogram H_B is defined as a function of these point errors. Since the interval corresponding to the buckets do not overlap and every point belongs to exactly one bucket, we can express the total error of a histogram H_B with buckets $b_1, \dots, b_B$ as a sum over all bucket errors: $\sum_j Err(b_j)$. In the case of the maximum (absolute) error histogram construction problem, the error Err for the bucket b_j defined by the interval $[l_j, r_j]$ and representative h_j is defined as follows:

$$Err(b_j) = Err(l_j, r_j) = \max_{i \in [l_j, r_j]} |x_i - h_j|.$$

The error of the histogram is given with:

$$E_X(H_B) = \sum_j Err(b_j) = \sum_j \max_{i \in [l_j, r_j]} |x_i - h_j|.$$

In order to minimize the error of the histogram we need to minimize the error of each bucket. For the case of the maximum **absolute** error histogram construction problem the representative values which minimize the maximum absolute error are computed using the formula $h_j = \frac{x_{l_j} + x_{r_j}}{2}$. Replacing the formula for h_j in $x_i - h_j$ the error of a bucket is given with $Err(b_j) = \frac{x_{r_j} - x_{l_j}}{2}$.

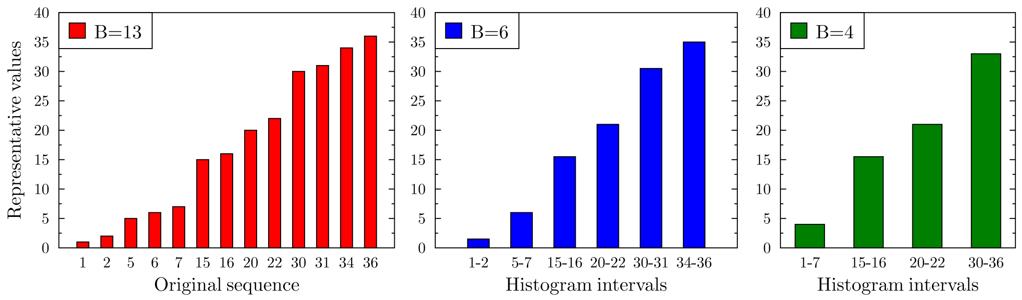


Figure 6 Illustration of the histogram construction problem.

▶ **Example 6.** Let us consider the following data sequence $X = \{20, 1, 5, 15, 5, 2, 16, 22, 36, 30, 34, 7, 31\}$. When constructing histograms it is usually useful to order the data $X = \{1, 2, 5, 6, 7, 15, 16, 20, 22, 30, 31, 34, 36\}$ to ease the representation. Figure 6 illustrates the original sequence and the resulting histograms for the given sequence X and two possible storage constraints: $B = 6$ and $B = 4$. As we can see, each histogram tries to approximate the data sequence using fewer data points. Thus, the histogram construction algorithm has to examine all of the possible divisions of this sequence in order to find the one that minimizes the total error $E_X(H_B)$ of the histogram.

We would first like to construct a maximum absolute error histogram using at most B=6 storage, which means that our histogram will have not more than 6 buckets. Let assume that we have such an algorithm which is able to find the optimal histogram for the given sequence X and space constraints B. The resulting optimal histogram for B=6 divides the data elements in the following sequence of buckets $B_1 = \{1, 2\}$, $B_2 = \{5, 6, 7\}$, $B_3 = \{15, 16\}$, $B_4 = \{20, 22\}$, $B_5 = \{30, 31\}$ and $B_6 = \{34, 36\}$, represented with the corresponding sequence of values $h_1 = 1.5$, $h_2 = 6$, $h_3 = 15.5$, $h_4 = 21$, $h_5 = 30.5$ and $h_6 = 35$, with a cumulative error $E_X(H_B) = 4$.

However, if we wish to reduce the storage to $B = 4$ pieces, then there are two possible solutions with equal errors and one of them comprises the following sequence of buckets $B_1 = \{1, 2, 5, 6, 7\}$, $B_2 = \{15, 16\}$, $B_3 = \{20, 22\}$ and $B_4 = \{30, 31, 34, 36\}$, represented with the corresponding sequence of values $h_1 = 4$, $h_2 = 15.5$, $h_3 = 21$ and $h_4 = 33$. The total error under these space constraints is $E_X(H_B) = 8$.

Another variant of the maximum error histogram construction problem is to use the relative error instead:

$$Err(l_j, r_j) = \min_{h_j} \max_{i \in [l_j, r_j]} \frac{|x_i - h_j|}{\max\{c, |x_i|\}},$$

where c is an absolute constant that works as a sanity bound, used to reduce excessive domination of the relative error by small data values. By setting c to be larger than all numbers in the input, the relative error is reduced to an absolute error multiplied by $\frac{1}{c}$ which allows to discuss both errors at the same time. The maximum absolute or relative error metrics enable to approximate the data with uniform fidelity throughout the domain, unlike sum-based measures.

Jagadish et al. [46] first gave a general technique for computing the optimum histogram in $\mathcal{O}(m^2B)$ time and $\mathcal{O}(mB)$ space for several measures. However, the quadratic running time showed is undesirable for large data sets, not to mention for streaming applications. Besides, having in mind that the histogram is already an approximation of the data, it came natural to think of near optimal solutions which can be constructed in time linear in the size of the input data. This line of thinking went even further, considering "tunable" approximations which would allow faster running times if a less accurate histogram suffices for the application at hand. As a result, many solutions have been developed for a slightly different problem formulation known as constructing $(1+\epsilon)$-*approximate* histograms [39]: *Given a sequence X of length m, a number of buckets B, and a precision parameter $\epsilon > 0$, find H_B with $E_X(H_B)$ at most $(1+\epsilon)\min_H E_X(H)$ where the minimization is taken over all histograms H with B buckets.* Thus, if we desire a 1% approximation to the optimal histogram, we would set $\epsilon = 0.01$. However, all of the solutions proposed have running times dependent on the size of the input, although polylogarithmic.

An interesting and important result is given by Guha and Shim [40], where they present a linear time *optimal* algorithm for the maximum absolute and relative error measures for computing the optimum histogram in $\mathcal{O}(m + B^2 \log^3 m)$ time and $\mathcal{O}(m)$ space. Note that the improvement in the running time is on the cost of an increased memory usage. Although the algorithm is linear, due to its memory dependence on the size of the input it cannot be used in streaming applications.

Only recently Guha [37] gave the first tight results for linear time algorithms whose space requirements do not depend on the size of the input, i.e., the data stream. The *StreamStrap* algorithm [37] uses the concept of thresholded approximation plugged into a framework in which multiple repetitions of the thresholded algorithm are run in a sequential manner, where each new run is enhanced and bootstrapped with the results from the previous run, an idea called "streamstrapping". The StreamStrap algorithm is discussed in more detail in the following section.

7.2 The StreamStrap Algorithm

To be able to apply the StreamStrap algorithm there are two basic requirements which have to be fulfilled in our summarization scenario:

1) *Thresholded small space approximations exists.*
2) *The error measure is a Metric error.*

For a given summarization problem P a *thresholded approximation* (as given in the work of [37]) is defined to be an algorithm which simultaneously guarantees that: 1) if there is a solution with summarization size B' and error ε (where ε is known), then in small space we can construct a summary of size at most B' such that the error of our summary is at most $\alpha\varepsilon$ for some $\alpha \geq 1$ and, 2) otherwise declare that no solution with error ε exists. An important point of the first requirement is that we use the knowledge of ε.

The second requirement translates into a property of the error measure which enables us to use the following inequality for any X, Y, H, H':

$$Err(X(H) \circ Y, H') - Err(X, H) \leq Err(X \circ Y, H') \leq Err(X(H) \circ Y, H') + Err(X, H),$$

where $Err(X, H)$ is the summarization error of X using the summary H (eg., maximum error histogram), $X \circ Y$ denotes a concatenation of input X followed by Y, and $X(H)$ is the summarized input in which every point x is replaced by the corresponding representative h from H.

A *thresholded* version of the optimal algorithm for the maximum error problem [40] can be easily derived by using the knowledge of ε in the computations [37]. For a given error ε the algorithm can produce a summary of the input in linear time, using $\mathcal{O}(B')$ space, if such summary exists. Further, both the maximum error and the square root of the VOPT error satisfy the second requirement. Thus, we can fulfill all the conditions required for the StreamStrap algorithm to be applied.

On a high level the algorithm proceeds as follows: First, it reads B points from the input. Since all the input values are stored at this point of time, the summarization error is 0. The algorithms continues with reading as long as the error remains zero. When the first input which causes a non-zero error is observed (say this error is ε_0), the algorithm initializes J copies of the *thresholded* summarization algorithm and runs the algorithms. Each copy is run for a different error value which is exponentially increasing with a factor of $(1+\epsilon)$, i.e., ε_0, $(1+\epsilon)\varepsilon_0$, …, $(1+\epsilon)^J\varepsilon_0$. The value of J is chosen such that $(1+\epsilon)^J > \alpha/\epsilon$, giving us $\mathcal{O}(\frac{1}{\epsilon}\log\frac{\alpha}{\epsilon})$ different algorithms.

A property of the thresholded algorithm is that it will succeed only if a summary exists for the given error ε and the available storage space. Therefore, when at some point in time some of the copies of the thresholded algorithm will declare a "fail" for some ε', we know that there exist no such solution for an error smaller or equal to ε', that is, $\varepsilon^* > \varepsilon'$. Now, we terminate all the algorithms run for error estimates $\le \varepsilon'$ and, start running a thresholded algorithm for $(1+\epsilon)^J\varepsilon'$ using the summary from the "failing" algorithm as the initial input. Whenever a running copy of the thresholded algorithm will declare a "fail" the same procedure is applied. As a result, we will always have the same number of running algorithms, but for different error estimates. At query time the algorithm returns the answer for the lowest error estimate for which a thresholded algorithm is still running, i.e., have not declared a "fail".

The general idea is to start with the smallest possible estimate of the error and raise it a number of times until we find a solution under the constraints on the memory storage and the approximation factor ϵ. Using the summaries from the previous runs and the property of a Metric error, it can be shown that for a given summarization problem that fulfills the requirements of the framework, for any $\epsilon \le 1/10$ the StreamStrap algorithm provides a $\alpha/(1-3\epsilon)^2$ approximation. The proof is given in [37].

7.3 Applications

If we apply the StreamStrap algorithm for the maximum error histogram construction problem with B buckets, we can have a single pass $1+\epsilon$ *streaming* approximation using $\mathcal{O}(\frac{B}{\epsilon}\log\frac{1}{\epsilon})$ space and $\mathcal{O}(m+\frac{B}{\epsilon}(\log^2\frac{B}{\epsilon})\log M\varepsilon^*)$ time. The error of any bucket will be additively within $\epsilon\varepsilon^*$ of the true error of that bucket.

The StreamStrap algorithm has been applied also to the the problem of K-center clustering, K-median clustering and the VOPT histogram construction problem, for which upper bounds on the space and running time are given [37]. Guha further proved the first lower space bounds for maximum error histograms and for the K-center problem in the *Oracle Distance Model*, where an oracle is assumed which given two input points and an additional small space determines their distance. In particular, for the maximum error histogram construction problem he proved that: *for all $\epsilon \le 1/(40B)$, any $1+\epsilon$ approximation for B bucket maximum error histogram, which also approximates the error of each bucket within additive ϵ times the optimum error must use $\Omega(\frac{B}{\epsilon\log(B/\epsilon)})$ bits of space.* This result is proved by using a reduction of the Indexing problem to the problem of constructing a histogram.

The importance of Guha's results lies in the fact that the StreamStrap algorithm can run indefinitely using a bounded amount of space and a constant processing time for each data item from the stream, while still providing an approximation or a summary which is according to the user's specifications (given the approximation factor ϵ and the size of the summary B). Thus, it can be easily applied in many data stream applications in the *cash-register* model, and as a building block in more sophisticated summary data structures for tracking various statistics of interest. To the best of our knowledge, similar results have not yet been achieved for the more general *turnstile* data stream model.

References

1 Swarup Acharya, Phillip B. Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. The aqua approximate query answering system. *SIGMOD Rec.*, 28:574–576, June 1999.

2 Gagan Aggarwal, Mayur Datar, Sridhar Rajagopalan, and Matthias Ruhl. On the streaming model augmented with a sorting primitive. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 540–549, 2004.

3 Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, 1975.

4 Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data structures and algorithms.* Addison-Wesley, Reading, Mass., 1983.

5 Noga Alon, Phillip B. Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'99, pages 10–20, New York, NY, USA, 1999.

6 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

7 C.R. Aragon and R.G. Seidel. Randomized search trees. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:540–545, 1989.

8 Aditya Akella Ashwin, Ashwin Bharambe, Mike Reiter, and Srinivasan Seshan. Detecting ddos attacks on isp networks. In *Proceedings of the Workshop on Management and Processing of Data Streams*, 2003.

9 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *PODS'02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16, New York, NY, USA, 2002.

10 Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA'02, pages 633–634, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

11 Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702 – 732, 2004. Special Issue on FOCS 2002.

12 Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. Optimal sampling from sliding windows. *J. Comput. Syst. Sci.*, 78(1):260–272, 2012.

13 A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES'97, pages 21–29, Washington, DC, USA, 1997. IEEE Computer Society.

14 Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of 13th Annual ACM Symposium on Theory of Computing*, STOC'98, pages 327–336, New York, NY, USA, 1998. ACM.

15 C. Buragohain, N. Shrivastava, and S. Suri. Space efficient streaming algorithms for the maximum error histogram. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1026 –1035, april 2007.

16 J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions (extended abstract). In *Proceedings of the ninth annual ACM symposium on Theory of computing*, STOC'77, pages 106–112, New York, NY, USA, 1977. ACM.

17 CERN: European Organisation for Nuclear Research. `http://public.web.cern.ch/`.

18 Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27:188–228, June 2002.

19 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37:79–102, 2007.

20 Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, ICALP'02, pages 693–703, London, UK, UK, 2002. Springer-Verlag.

21 Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55:441–453, December 1997.

22 Edith Cohen, Mayur Datar, Shinji Fujiwara, Aristides Gionis, Piotr Indyk, Rajeev Motwani, Jeffrey D. Ullman, and Cheng Yang. Finding interesting associations without support pruning. *IEEE Trans. on Knowl. and Data Eng.*, 13:64–78, January 2001.

23 Edith Cohen and Martin Strauss. Maintaining time-decaying stream aggregates. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'03, pages 223–233, New York, NY, USA, 2003. ACM.

24 Graham Cormode, Mayur Datar, Piotr Indyk, and S. Muthukrishnan. Comparing data streams using hamming norms (how to zero in). In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB'02, pages 335–345. VLDB Endowment, 2002.

25 Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'03, pages 296–306, New York, NY, USA, 2003. ACM.

26 Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58 – 75, 2005.

27 Corinna Cortes and Daryl Pregibon. Giga-mining. In *Knowledge Discovery and Data Mining*, pages 174–178, 1998.

28 M. Datar and S. Muthukrishnan. Estimating rarity and similarity over data stream windows. In *Proceedings of the 10th Annual European Symposium on Algorithms*, ESA'02, pages 323–334, London, UK, UK, 2002. Springer-Verlag.

29 Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM J. Comput.*, 31:1794–1813, June 2002.

30 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In Rolf H. Möhring and Rajeev Raman, editors, *ESA*, volume 2461 of *Lecture Notes in Computer Science*, pages 348–360. Springer, 2002.

31 Earth System Research Laboratory. `http://www.esrl.noaa.gov/psd/`.

32 Joan Feigenbaum, Sampath Kannan, Martin J. Strauss, and Mahesh Viswanathan. An approximate l1-difference algorithm for massive data streams. *SIAM J. Comput.*, 32:131–151, January 2003.

33 Philippe Flajolet and G. Nigel Martin. Probabilistic counting. In *FOCS*, pages 76–82. IEEE, 1983.

34 Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.*, 1:397–413, August 1993.

35 Anna C. Gilbert, Sudipto Guha, Piotr Indyk, Yannis Kotidis, S. Muthukrishnan, and Martin J. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In

Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, STOC'02, pages 389–398, New York, NY, USA, 2002.

36 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, SIGMOD'01, pages 58–66, New York, NY, USA, 2001.

37 Sudipto Guha. Tight results for clustering and summarizing data streams. In *Proceedings of the 12th International Conference on Database Theory*, ICDT'09, pages 268–275, New York, NY, USA, 2009. ACM.

38 Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC'01, pages 471–475, New York, NY, USA, 2001. ACM.

39 Sudipto Guha, Nick Koudas, and Kyuseok Shim. Approximation and streaming algorithms for histogram construction problems. *ACM Trans. Database Syst.*, 31:396–438, March 2006.

40 Sudipto Guha and Kyuseok Shim. A note on linear time algorithms for maximum error histograms. *IEEE Trans. on Knowl. and Data Eng.*, 19:993–997, July 2007.

41 Sudipto Guha, Kyuseok Shim, and Jungchul Woo. Rehist: Relative error histogram construction algorithms. In *Very Large Data Bases*, pages 300–311, 2004.

42 Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

43 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53:307–323, May 2006.

44 Piotr Indyk and David Woodruff. Optimal approximations of the frequency moments of data streams. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC'05, pages 202–208, New York, NY, USA, 2005.

45 C. L. Mallows J. M. Chambers and B. W. Stuck. A method for simulating stable random variables. *Journal of the American Statistical Association*, 71:340–344, June 1976.

46 H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB'98, pages 275–286, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

47 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the twenty-ninth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS'10, pages 41–52, New York, NY, USA, 2010. ACM.

48 Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, March 2003.

49 Panagiotis Karras, Dimitris Sacharidis, and Nikos Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD'07, pages 380–389, New York, NY, USA, 2007. ACM.

50 Eyal Kushilevitz and Noam Nisan. *Communication complexity.* Cambridge University Press, 1997.

51 LHC Computing Grid. `http://lcg.web.cern.ch/LCG/`.

52 Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB'02, pages 346–357. VLDB Endowment, 2002.

53 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *Proceedings of the 1998 ACM*

SIGMOD international conference on Management of data, SIGMOD'98, pages 426–435, New York, NY, USA, 1998.

54 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *Proceedings of the 1999 ACM SIGMOD international conference on Management of data*, SIGMOD'99, pages 251–262, New York, NY, USA, 1999.

55 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, New York, NY, USA, 2005.

56 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, 1993.

57 M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In Haran Boral and Per-Åke Larson, editors, *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, June 1-3, 1988*, pages 28–36. ACM Press, 1988.

58 S. Muthukrishnan. Data streams: algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 1(2):117–236, 2005.

59 Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD'84, pages 256–276, New York, NY, USA, 1984.

60 Nicole Schweikardt. One-pass algorithm. In Ling Liu and M. Tamer Zsu, editors, *Encyclopedia of Database Systems*, pages 1948–1949. Springer Publishing Company, Incorporated, 1st edition, 2009.

61 P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, SIGMOD'79, pages 23–34, New York, NY, USA, 1979. ACM.

62 Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, SenSys'04, pages 239–249, New York, NY, USA, 2004.

63 Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11:37–57, March 1985.

64 Mark N. Wegman and J. Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265 – 279, 1981.

65 David Woodruff. Optimal space lower bounds for all frequency moments. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA'04, pages 167–175, Philadelphia, PA, USA, 2004.

Data Stream Management Systems *

Sandra Geisler

RWTH Aachen University
Ahornstrasse 55, 52056 Aachen, Germany
`geisler@dbis.rwth-aachen.de`

Abstract

In many application fields, such as production lines or stock analysis, it is substantial to create and process high amounts of data at high rates. Such continuous data flows with unknown size and end are also called data streams. The processing and analysis of data streams are a challenge for common data management systems as they have to operate and deliver results in real time. Data Stream Management Systems (DSMS), as an advancement of database management systems, have been implemented to deal with these issues. DSMS have to adapt to the notion of data streams on various levels, such as query languages, processing or optimization. In this chapter we give an overview of the basics of data streams, architecture principles of DSMS and the used query languages. Furthermore, we specifically detail data quality aspects in DSMS as these play an important role for various applications based on data streams. Finally, the chapter also includes a list of research and commercial DSMS and their key properties.

1998 ACM Subject Classification H.2.4 Systems

Keywords and phrases Data Streams, Data Stream Management, Data Quality, Query Languages

Digital Object Identifier 10.4230/DFU.Vol5.10452.275

1 Introduction

Today, sensors are ubiquitous devices and they are crucial for a multitude of applications. Especially, tasks like condition monitoring or object tracking often require sensors [76]. Important examples for monitoring applications are weather observation and environment monitoring in general, health monitoring, monitoring of assembly lines in factories, RFID monitoring, or road monitoring. These applications share characteristic properties which are especially challenging for a data management system processing this data. First of all, the sensed data is produced at a very high frequency, often in a bursty manner, which may pose real-time requirements on processing applications and may allow them only one pass over the data. ECG signals, for example, are created at a frequency of usually 250 Hz. Second, sensor data is not only produced rapidly, but also continuously forming a *data stream*. Data streams can be unbounded, i.e., it is not clear, when the stream will end. Data from sensors furthermore can be defective, i.e., it is likely to include errors introduced by the imprecision of measurement techniques (e.g., a vehicle speedometer has an error tolerance of 10% [29]), data may be lost due to transmission failure or failure of the sensor.

Also the mapping of recorded sensor data to a time domain is important to rate the timeliness of the data and to make it interpretable in that dimension. In monitoring

* This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the grant 01BU0915 (Project Cooperative Cars eXtended, `http://www.aktiv-online.org/english/aktiv-cocar.html`) and by the Research Cluster on Ultra High-Speed Mobile Information and Communication UMIC (`http://www.umic.rwth-aachen.de`).

applications the most recent data is apparently the most interesting data. This reveals another source of defectiveness of the sensor data – namely disorder of data, which is likely to happen when the protocol used for transmission cannot guarantee to sustain order or network latencies occur [64]. In monitoring applications, data from multiple sources have also to be integrated and analyzed to build a comprehensive picture of a situation. For example, integrating data from several health sensors (such as ECG, temperature, blood pressure) has to be integrated to derive that a patient is in a critical situation. Another challenge is caused by the pure mass of data. Due to limited system resources in terms of space and CPU time, algorithms analyzing the data, e.g. data mining algorithms, cannot store and process the entire data, but they can process the data only once (*the one-pass property*) with the available resources. In some applications, it is also required to combine streaming data with historical or static data from a common database, e.g., when for a monitored road section we want to look up, if it currently contains a construction site. All of these properties are especially common to sensor data on which we will focus in this chapter. There are also other typical applications producing data streams (further termed *stream applications*). Popular examples are stock price analysis or network traffic monitoring, which can produce even millions of samples per second.

To tackle the aforementioned challenges in data stream processing, a specific type of systems called Data Stream Management Systems (DSMS) has evolved. The properties of stream applications discussed before lead to a long list of requirements for DSMS. In contrast to common data management systems and based on the nature of stream applications, DSMS have to react to incoming data and deliver results to listening users frequently [65]. This is also termed as the *DBMS-Active, Human-Passive model* by Carney et al. [19], while the *DBMS-Passive, Human-Active model* is implemented by common Database Management Systems (DBMS). In DSMS a reactive behaviour is realized by continuous queries which are registered by a user in the system once and after that the queries are executed incessantly. But some applications may at the same time require to allow ad hoc user queries [2] or views [39] also. Data in a DSMS must not only be processed and forgotten, but the system also has to react to changes of data items, which may lead to recalculation of already produced results. This requires an appropriate change management in the system [1, 8].

Handling unbounded data streams while having only a limited amount of memory available and being restricted in CPU time for processing the data is one of the main challenges for a data stream management system. The creation of incremental results for critical data processing operations and the application of window operators are only two examples of how these issues are solved in DSMS. As already mentioned, most of the applications pose real-time requirements to the data processing system [65]. This implicitly comprises the requirement to be scalable in terms of data rates, which in turn demands techniques for load balancing and load shedding to be incorporated in a stream system. But the system must be also scalable in terms of queries as some application contexts can get complex and require the introduction of several queries at the same time. Therefore, the demand for and the adaptability to newly registered, updated, or removed queries is obvious [20]. This also brings up the need for multi-query optimization, e.g., by sharing results of operators in an overall query plan. Query plan modification during query processing is not only desirable for query optimization, but also to serve Quality of Service demands under varying system resource availabilities [59].

The imperfectness of data has also to be addressed by a DSMS. Unordered data has to be handled adequately, and may be tolerated in controlled bounds. Means to recognize and rate the quality of the data processed are also crucial to make assumptions about the produced answers or results [73]. Finally, as Stonebreaker et al. [65] demand, a DSMS also has to have a deterministic behaviour, which outputs predictable and repeatable results to implement fault tolerance and recovery mechanisms. This contrasts with non-deterministic components in a DSMS, such as a component randomly dropping tuples to compensate a high system load [78].

Parallel to DSMS, the terms Event Processing, Complex Event Processing (CEP) or Event Stream Processing have evolved. These terms are referring to a concept which is closely related to the notion of data streams. The corresponding systems are specialized on the processing and analysis of events to identify higher level events, such as predicting a political development from Twitter tweets or detecting a serious condition from vital parameter readings. DSMS have a broader application scope, but CEP applications can also be rebuild with DSMS [28]. Cugola and Margara distinguish DSMS and CEP systems as data processing and event detection systems denoting the different focuses. The focus disparities result, among others, in differences in architectures, data models and languages between CEP systems and DSMS [25]. The focus of this chapter is put on DSMS. For a comparison of CEPs and DSMS we refer to the survey by Cugola and Margara [25].

In this chapter, we will give a brief overview of DSMS and of some of the solutions which address the above requirements. As there are already excellent surveys on the concepts of DSMS, e.g., [11, 38, 22, 78, 39], we will focus on two main aspects and discuss them in more detail: query languages for DSMS and data quality in DSMS.

1.1 Running Example – A Traffic Application Scenario

Throughout this chapter, we will use a real-time traffic management application as a running example, namely traffic state estimation based on data from multiple mobile traffic sources. In Car-to-X (C2X) communication vehicles can communicate with other vehicles (Car-to-Car) or with the road infrastructure (Car-to-Infrastructure). The data collected and sent is called *Floating Car Data* (FCD). For example, a vehicle can warn other vehicles behind it when it brakes very hard. We consider two kinds of event-based messages sent out by the vehicles: Emergency Braking Light (EBL) messages, which are created when a vehicle has a high negative acceleration, and Warning Light Announcement messages (WLA). The latter are produced when a vehicle turns on its warning light flashers. These messages contain general information about the current state of the vehicle, such as the speed, the location, or the acceleration.

Example 1 shows the schema of a C2X message stream, which is an example for FCD. Another source of a data stream may be anonymously collected mobile phone position data (also called Floating Phone Data (FPD) analogously to Floating Car Data (FCD)). This can be used to derive further traffic information, such as the speed or the traffic state [33].

▶ **Example 1.** In our case study we receive the CoCar messages sent by the equipped vehicles as a data stream. The simplified schema of the stream `CoCarMessage` is as follows,

$$\mathit{C2XMessage}(\mathit{TS}, \mathit{AppID}, \mathit{Speed}, \mathit{Acceleration}, \mathit{Latitude}, \mathit{Longitude})$$

where `AppID` represents the message type, such as an emergency brake message, `Longitude` and `Latitude` represent the position of the vehicle and `Speed` and `Acceleration` represent the current values for these attributes of the vehicle. Additionally, the data stream elements contain a timestamp `TS` from a discrete and monotonic time domain.

The idea of traffic state estimation based on C2X messages is simple. We divide roads in a given road network into equal-sized sections, e.g., of 100 meters length, as depicted in

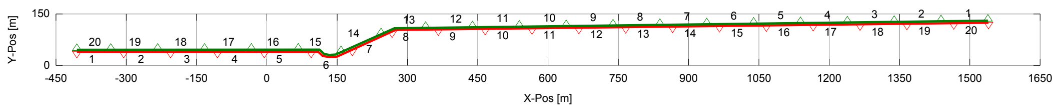


Figure 1 A road with two directions divided up into sections [35].

Figure 1. We collect the messages and data produced, assign the data to a section based on the position in the message, and aggregate the data for each section and a certain time period (e.g., the last minute). Based on the aggregated data for each section we can then determine the traffic state either based on simple rules (if speed is higher than x and the number of messages is higher than y) or using data stream mining techniques.

We realized the described application in the course of the CoCar project[1] and its successor CoCarX. We implemented an architecture based on a DSMS and data stream mining algorithms. The architecture is supposed to derive further traffic data from raw data, such as hazard warnings or the traffic state, and to send it out as traffic information to end-users [35, 34]. The CoCar project investigates the feasibility of traffic applications based on Car-to-X communication via cellular networks (UMTS and its successor LTE) and pWLAN.

2 DSMS Architectures

Due to the system requirements for DSMS their architectures differ in several aspects from the traditional relational DBMS architecture. Querying has to be viewed from a different angle as data is pushed into the system and not pulled from the system [20]. In Data Stream Management Systems queries are executed continuously over the data passed to the system, also called *continuous* or *standing queries*. These queries are registered in the system once. Depending on the system, a query can be formulated mainly in two ways: as a declarative expression, mostly done in an SQL-dialect, or as a sequence or graph of data processing operators. Some of the systems provide both possibilities. A declarative query is parsed to a logical query plan, which can be optimized. Similar to DBMS this logical query is afterwards translated into a physical query execution plan (QEP). The query execution plan contains the calls to the implementation of the operators. Besides of the actual physical operators, query execution plans include also *queues* for buffering input and output for the operators. A further support element in QEPs are *synopsis structures*. DSMS may provide specific synopsis algorithms and data structures which are required, when an operator, e.g., a join, has to store some state to produce results. A synopsis summarizes the stream or a part of the stream. It realizes the trade-off between memory usage and accuracy of the approximation of the stream. Additionally, *load shedders* can be integrated in the plan, which drop tuples on high system loads. In most systems, execution plans of registered queries are combined into one big plan to reuse results of common operators for multiple queries. The physical query plan may be constantly optimized based, e.g., on performance statistics. In Figure 2 the query processing chain is depicted.

In Figure 3 a generic architecture of a DSMS based on [5, 39] is shown. First of all a DSMS typically gets data streams as input. Wrappers are provided, which can receive raw data from its source, buffer and order it by timestamp (as e.g. implemented by the *Input Manager* in the STREAM system [64]) and convert it to the format of the data stream

[1] `http://www.aktiv-online.org/english/aktiv-cocar.html`

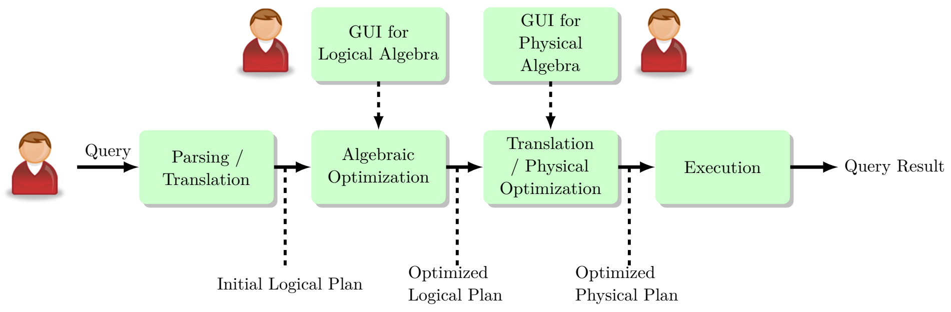


Figure 2 Illustration of the Query Processing Chain in DSMS as given in [50].

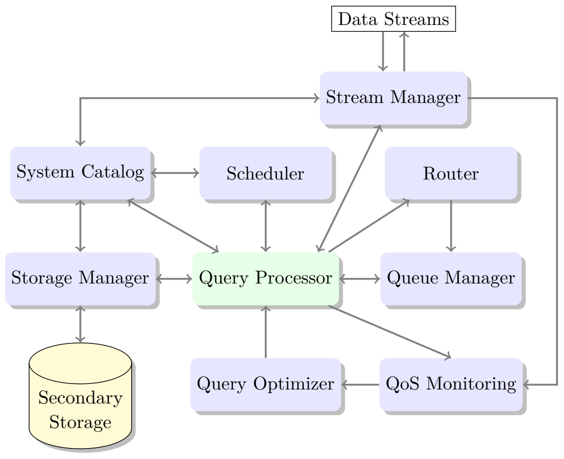


Figure 3 An generic architecture of a DSMS based on [5, 39].

management system (the task of the *Stream Manager*). As most systems adopt a relational data model, data stream elements are represented as tuples, which adhere to a relational schema with attributes and values. After reception the tuples are added to the queue of the next operator according to the query execution plan. This can be done e.g. by a *Router* component as implemented in the Aurora system [3]. The management of queues and their corresponding buffers is handled by a *Queue Manager*. The Queue Manager can also be used to swap data from the queues to a secondary storage, if memory resources get scarce. To enable access to data stored on disk many systems employ a *Storage Manager* which handles access to *secondary storage*. This is used, when persistent data is combined with data from stream sources, when data is archived, or swapped to disk. Also it is required when loading meta-information about, inter alia, queries, query plans, streams, inputs, and outputs. These are held in a system catalog in secondary storage.

While the queue implementation decides which element is processed next, a *Scheduler* determines which operator is executed next. The Scheduler interacts closely with the *Query Processor* which finally executes the operators. Many systems also include some kind of *Monitor* which gathers statistics about performance, operator output rate, or output delay. These statistics can be used to optimize the system execution in several ways. The scheduler strategy can be influenced, e.g., prioritizing specific subplans. Furthermore, the throughput of a system can be increased by a *Load Shedder*, i.e., stream elements selected by a *sampling* method are dropped. The Load Shedder can be a part of a Query Optimizer, a single component, or part of the query execution plan. Furthermore, the statistics can be used to reoptimize the current query execution plan and reorder the operators. For this purpose a *Query Optimizer* can be included. In the Telegraph system [20] subsets of operators are build which are commutative. The operators of each subset are connected to a component called Eddy. Each Eddy routes the incoming tuples to the operators connected to it based on optimization statistics. Each tuple has to log, which operator has already processed it successfully and when all attached operators processed it, it is routed to the next part of the plan or output to recipients [20]. DBMS and DSMS share some of their query optimization goals – both try to minimize computational costs, memory usage and size of intermediate results stored in main memory. But obviously, the priorities for these goals are different for the two system types. DBMS mainly try to reduce the costs of disc accesses [27], while DSMS mainly have to reduce memory usage and computation time to be fast enough. Of course, these different goals stem from the different data handling strategies (permanent storage vs. real-time processing). In a DSMS a query is also not only optimized before execution, but it is adaptively optimized during its run time. This enables the system to react to changes of input streams and system and network conditions.

An interesting work to research the different architectural components of a DSMS is the Odysseus [17, 32] framework. Odysseus allows to create customized DSMS for which it provides basic architectural components while offering variation points for each of the components. At these variation points custom modifications such as the integration of a new data model or the inclusion of new algebraic and physical operators or new optimization rules for logical query plans are possible.

Now that general concepts of DSMS architectures for query management and processing have been introduced, the user interface to access the data, namely the principles of query languages in DSMS are discussed in the next section.

3 Query Languages in DSMS

In principle, two main types of query languages for DSMS can be distinguished: declarative languages (mostly relational, based on SQL) and imperative languages which offer a set of operators (also called box operators) to be assembled to a data-flow graph using a graphical user interface. The imperative languages often also include operators which represent SQL operations. SQL-based languages are widely used, though SQL has many limitations for querying streams [52]. Systems which include a declarative SQL-based language are, for example, STREAM (Continuous Query Language, CQL) [10], Oracle Event Processing[2], PIPES [50], SASE (Complex Event Language) [43], or StreamMill (Expressive Stream Language, ESL) [72]. Imperative languages are supported for example by the Aurora/Borealis system (SQuAl) [3], or System S/InfoSphere Stream[3] (SPADE/SPL) [31].

Because a stream is potentially unbounded in size, it is neither feasible nor desirable to store the entire stream and analyze it. Some of the operations known from traditional query languages, such as SQL, might wait infinitely long to produce a result, as the operation would have to see the entire stream to generate a result (defined as *blocking operations*) [11]. The missing support of sequence queries, i.e., retrieving sequential data, is one crucial limitation known from relational databases and SQL [52]. One simple yet powerful way is to first extract only a desired portion of the stream and use this portion in the remainder of the query. Therefore, a very important requirement for a DSMS query language is the provision

[2] `http://oracle.com/technetwork/middleware/complex-event-processing/overview/index.html`
[3] `http://ibm.com/software/data/infosphere/streams`

of windows [22, 9, 57]. *Windows* are operators that only select a part of the stream according to fixed parameters, such as the size and bounds of the window. Hence, they provide an approximation of the stream, but are at the same time implementing the desired query semantics [11]. A window is updated based on fixed parameters [57] and internal matters of the system (e.g., in principle, a result can be updated whenever a new element arrives or whenever time proceeds) [45]. We will detail the different types and parameters of windows in Section 3.1.5.

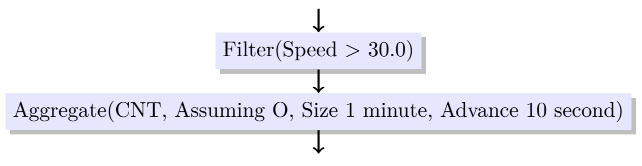
Besides the definition of windows in the language, also an approved set of query operations accompanied by established semantics is beneficial for a DSMS query language [9]. This can be accomplished, e.g., by using a common query language based on relational algebra and its operators for operations on finite tuple sets (commonly called *relations*). The advantage is the reuse of operator implementations and transformations for query optimization [9]. But such an approach also risks to be complicated, because the closure under a consistent mathematical structure, such as bags in relational algebra, is crucial to enable nested queries and algebraic optimization. Cherniack and Zdonik [22] investigated the property of DSMS query languages to be closed under streams. A language is closed under streams if its operators get streams as an input and if the output of the operators are streams as well. They state that most languages provide stream-to-stream operators by either implicitly using operators for windowing and conversion to streams or special stream-to-stream operators. Only CQL provides the possibility to explicitly formulate the conversion of relations to streams by specific relation-to-stream operators. These operators can either add output elements to a result stream when a new element compared to the last time step is in the result set (Istream operator), or when an element has been removed from the query result relation compared to the last time step (Dstream operator), or all elements which are present in the result set in the current time step (Rstream) are added to the stream [10, 9]. But to avoid an inconsistent query formulation producing results not closed under streams, CQL also offers many default query transformations. In fact, Arasu et al. showed in [9] that a stream-only query language (only using stream-to-stream operators) can be build based on the set of CQL operators. We will detail the types of operators used in continuous query languages in Section 3.1.4.

▶ **Example 2.** Suppose we want to monitor the number of speeders in a reduced speed area. We would like to retrieve the number of CoCar messages which have been sent in the last minute and which contain a speed greater than $30km/h$. Additionally, we also want to retrieve an update every 10 seconds. In CQL the following query could be formulated to fulfil our information requirement:

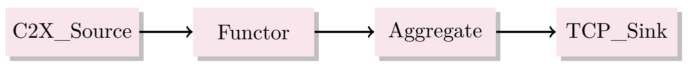
```
SELECT Istream Count(*) FROM
  C2XMessage[Range 1 Minute Slide 10s]
WHERE
  Speed > 30.0
```

Listing 1 Query 1

The same query can be formulated in an imperative query language by assembling box operators. In Figure 4a a query formulated in Aurora's SQuAl is depicted. The Filter operator is similar to the selection in relational algebra – it retrieves all tuples with speed greater than 30.0. Furthermore, an Aggregate operator is connected to the Filter operator, which can be parametrized with the aggregate function and details for an implicit window integrated in the operator. The SPADE query in Figure 4b is similar – the parametrized Functor operator selects the tuples according to the speed restriction and the Aggregate operator executes the Count operation.


Filter(Speed > 30.0)
Aggregate(CNT, Assuming O, Size 1 minute, Advance 10 second)

(a) Query 1 in SQuAl



(b) Query 1 in SPADE

Figure 4 Query 1 formulated in imperative languages.

To include more powerful and custom functionality, another requirement to DSMS query languages is the extension of the language by custom functions and operators to include more complex actions, such as data mining or custom aggregates [52]. Most of the languages provide a possibility for custom extensions. In the System S by IBM [31] a code generator enables users to create skeleton code for user-defined operators in C++ and add custom code to the operator. The operator is treated the same way as the other operators, e.g., during query optimization. The Aurora system provides a specific operator *Map* in which user-defined functions can be integrated and are executed whenever a new element is received [3]. In the Stream Mill ESL [52, 72] User-defined Aggregates (UDAs) based on concepts from the SQL:99 standard (INITIALIZE, ITERATE, INSERT INTO RETURN, TERMINATE) and the stream are used to implement new operators such as non-blocking aggregates or windows. The UDAs process one tuple at a time and allow for keeping a state over the so far seen stream elements. They can output a result when a specified condition is fulfilled and not only when all the input elements have been seen. In [72] the authors also show how they integrate stream mining algorithms into Stream Mill using UDAs.

3.1 Data Models and Semantics of Data Streams

Before we go into detail on continuous queries and operators used in the queries, we have to understand what is exactly meant by the term "data stream". In this section we will also break data streams down to their indivisible components and examine the data models applied in DSMS.

3.1.1 Data Models

Depending on the desired application realized with the DSMS and the data sources to support there are different demands on the adopted data model and the semantics of the query language. In the literature, the relational model is very dominant, presumably due to the well-defined semantics, the established set of relational algebra operators, and the very well studied principles of DBMS. Also, from a user's perspective, the step from SQL towards its streaming extensions seems to be quite small. But there are also data sources which do not conform to the flat table concept for discrete data offered by the relational data model [56]. XML data received from, e.g., web services or RSS feeds as well as continuous signal data, object streams, or spatio-temporal data streams demand special treatment. Therefore, a variety of specialized DSMS with differing data models and corresponding query languages have been proposed.

The maintenance and analysis of massive and dynamic graphs is also a challenging problem which can be tackled by data stream technology. There are multiple forms of this data model possible. The stream can consist of, amongst others, a sequence of edges [30], a stream of graphs where each graph is an adjacency matrix or list [67], or weight updates of edges [6]. It may be used in event detection and forecasting e.g. of Twitter streams, for web page linkage or social network analysis. To achieve a stable logical foundation for DSMS Zaniolo presents a data model and query language called Streamlog, which is based on Datalog [77]. While stream sources are represented by fact streams and sinks (or output streams) by the goal of a datalog query, the operators in between are defined by rules.

An interesting generic approach, supporting multiple query languages and data models, is implemented by the Odysseus framework [17]. A logical algebra component converts queries from various language parsers to a logical query plan with generic algebra operators. The semantics of the PIPES system [50] also allows for implementing multiple data models. In this chapter we will concentrate mostly on the classical relational model, though there are also many works on XML stream processing, e.g., using XML-QL [21], XPath [58] or XQuery [18]. In the following we will discuss various representations and semantics of data streams and concepts related to them.

3.1.2 Representations and Semantics of Data Streams

A data stream S can be understood as an unbounded multiset of elements, tuples, or events [52, 11, 26, 50]. This means, each tuple can occur more than once in the stream which is denoted by a multiplicity value. Each tuple $(s, \tau) \in U$, where U is the support of the multiset S, has to adhere to the schema of S. The *support* U of a multiset S is a set which consists of the elements which occur in the multiset S at least once. Multiset and support can be defined as follows [68, 63]:

▶ **Definition 3.** (Multiset and Support of a Multiset) A *multiset* is a tuple $\mathcal{A} = (B, f)$, where B is a set and f is a function $f : B \to \mathbb{N}$, assigning a multiplicity to the tuple (number of occurrencies of the tuple in the stream). The *support* of $\mathcal{A}$ is a set U which is defined as follows:

$$U = \{x \in B | f(x) > 0\},$$

i.e., $U \subseteq B$.

The schema of a stream is constituted of attributes $A_1, \ldots, A_n$. Each tuple contains also an additional timestamp τ from a discrete and monotonic time domain. Most definitions of data stream semantics do not consider the timestamp as a part of the stream schema [10, 50, 3] and therefore, it is always separately listed in the tuple notation.

A data stream can be formally defined as follows, based on the definition of Arasu et al. [10]:

▶ **Definition 4.** (Data Stream) A data stream S is an unbounded multiset of data stream elements (s, τ), where $\tau \in T$ is a timestamp attribute with values from a monotonic, infinite time domain T with discrete time units. s is a set of attribute values of attributes $A_1, A_2, \ldots, A_n$ with domains $dom(A_i), 1 \leq i \leq n$, constituting the schema of S. A stream starts at a time τ_0. $S(\tau_i)$ denotes the content of the stream S at time τ_i, being
$S(\tau_i) = \{< (s_0, \tau_0), m_0 >, < (s_1, \tau_1), m_1 >, \ldots, < (s_i, \tau_i), m_i >\}$.

▶ **Example 5.** The schema of the C2XMsgs stream from Example 1 would be written according to this definition as:

$$C2XMessage(Timestamp, (TS, AppID, Speed, Acceleration, Latitude, Longitude))$$

In this example the timestamp `Timestamp` has been generated by the DSMS and `TS` is the creation timestamp defined by the application. The different kinds of timestamps are detailed in Section 3.2.

Depending on the data model, the attributes $A_1, \ldots A_n$ can contain values of primitive data types, objects or XML data. For example, Krämer and Seeger consider tuples to be drawn from a composite type (in the relational case this is the schema) and its attributes can contain objects [50]. This generic definition allows them to be open to different data models. Gürgen et al. propose a generic relational schema for streams of sensor data of all kinds. Each tuple in a stream consists of several general attributes also denoted as *properties* containing the meta-data of the sensor, an attribute *measurement* carrying the values measured by that sensor and a timestamp, when the measurement has taken place [41]. Their semantics of a stream considers a temporal aspect, i.e., the present, past, and future of a stream is defined. The life time of present data is limited by the size of the corresponding queue in the DSMS and becomes past data when the queue is full [41].

Data streams commonly adhere to an append-only principle, i.e., data once inserted in the stream will not be removed or updated [14]. But to enable updates in the streams, there are also systems which do not only support insertion of tuples, but also updates and deletions. In the Borealis system [1], which is the distributed successor system of Aurora, an extended data stream model has been implemented. The schema of the stream has one or more attributes which have been designated as the key of the stream. This allows for the identification of data stream elements in the stream system for future changes. Furthermore, the tuples include a revision flag which indicates if the tuple is either an insertion, an update, or a deletion. The revision flag is processed by operators in the query plan and if an update or deletion has been detected, former buffered results are reevaluated and also tagged with a corresponding revision flag. Additionally, each tuple can also have information about Quality of Service attached, which will be detailed in Section 4. The internal (physical) representation of streams in STREAM uses the concept of revision flags, too, to denote inserted and outdated tuples, e.g. for a window [10].

In the PIPES [50] and STREAM systems, streams are separated into *base streams* and *derived streams* to denote the origin of the stream. Base streams are produced by an external source and derived streams are produced by internal system operators. Furthermore, in PIPES stream notations are distinguished based on the level in the query processing chain. Streams from external sources are termed *raw streams* and adhere to the attributes plus timestamp notation described above (according which the tuples are ordered in the stream). Streams on the logical or algebraic level are termed *logical streams.* The tuples of a logical stream contain attributes corresponding to the stream schema, a timestamp τ and a multiplicity value. The multiplicity value indicates how often the tuple occurs at time τ in the stream, which implements the bag semantics explicitly. Finally, the PIPES system also introduces a *physical stream* notation. The notation is used to represent streams in query execution plans, which include in addition to the attributes a validity time interval with start and end timestamp, similar to the CESAR language [26], which indicates when a tuple is outdated.

While above we have described the representations of streams and tuples, the semantics or denotation of a stream (i.e., the underlying mathematical concept) can be separated from these representations [56]. So far we used the rough semantics of an unbounded multiset for a stream. This rough semantics raises the questions of how tuples are organized in the stream, which tuples are included, and how a stream evolves with the addition of tuples. A stream denoted as a *multiset* or *bag* of elements, allows duplicate tuples in the stream [10, 15, 50].

The denotation as a *set*, i.e., without duplicates, is also possible. Furthermore, a stream could also be interpreted as a sequence of states [56], where a relation transitions from one state to another (on arrival of tuples or progression of time). Maier et al. propose reconstitution functions to formally describe the denotations of streams and operations on these denotations [56]. For example, the insertion of a new tuple in a stream denoted as a bag can be recursively defined by describing what will be the following state, i.e., the "result bag". The reconstitution functions allow to write down the semantics in a more formal way and can then be used to prove properties like the correctness of operators for the corresponding denotation.

3.1.3 Inclusion of Persistent Relations

Solely querying of streams is often not sufficient to implement certain applications. Take the traffic state estimation as an example. To improve our assertion about the traffic state we could also integrate information from other streams, such as Floating Phone Data (positions from anonymously tracked phones in vehicles), or from persistent sources such as historical data about the traffic state at the same time last week or last year on the same street. This would involve to join data of streams with other streams and to join streams with persistent data from "static" data sources. In [22] this ability is called *correlation*. Most languages support joins between streams and relations, because it is easy to implement. In principle, each time new tuples arrive in the stream these are joined with the data in the relation and the operator outputs the resulting join tuples. The join between streams is a little bit more complicated. Most languages require at least one stream to be windowed [22], which results in the former situation of joining a stream with a finite relation. Some offer also specific operators for joins without windows.

The use of persistent relations in queries requires them to be represented in the query language. In SQL-based languages these are noted in the same way as streams (it is maybe more correct to say that the streams are represented in the same way as relations). Representations for relations can also be related to the notion of time. A relation at time τ consists then of a finite, unbounded bag of tuples which is stored in the relation at time τ [10]. In CQL, a time-related relation is called an *instantaneous relation*, and analogously to streams, *base relations* and *derived relations* are distinguished.

▶ **Example 6.** In this example we want to know for a certain road and section on this road, if and how many road works it currently contains. The schemas of the stream with aggregated information from the C2X messages about the road section and the persistent relation are as follows:

$$AvgC2XMessage(Timestamp, (AvgSpeed, AvgAcceleration, RoadID, SectionID))$$

$$ConstructionSite(SiteID, StartDate, RoadID, SectionID)$$

In CQL we would formulate the query in the following way:

```
SELECT Rstream m.* , Count(c.SiteID) As SiteNo
FROM AvgC2XMessage[Now] AS m, ConstructionSite AS c
WHERE c.RoadID = m.RoadID AND c.SectionID = m.SectionID
```

Note, that CQL is limited in its ability to join streams and relations. Only a `NOW` window on the stream and the Rstream operator can safely be used for joins between streams and relations. Queries with different windows or relation-to-stream operators would usually deliver semantically incorrect results [10].

3.1.4 Continuous Queries and Algebraic Operators

Now that we have clarified the main constituents of continuous queries in DSMS, namely, streams, tuples, and relations, we will proceed to explain continuous queries and operators for languages based on the relational algebra. So what is the difference between a continuous query and an ad-hoc query? One goal of Tapestry [70], one of the first data management systems processing continuous data, was to give the user the impression that a query is continuously executed (which is not possible). To achieve this in DSMS, the result of a continuous query at time τ is equal to the result of the query executed at every time instant before and equal to τ [70, 10]. That means it takes into account all tuples arrived up to τ. Depending on the language the result of the query can be a stream or a finite set of tuples, e.g., in CQL the result can be either or, while other languages only support streams.

There are three main models how data is processed in a DSMS, i.e., when continuous queries are executed. When a *time-driven model* [45] is used, a query will be updated with progression of time (on every time step of the system). In a *tuple-driven model* a query is evaluated on the arrival of each tuple, unless the query includes some temporal restriction, such as a time-based window [45]. The *event-driven model* allows to define events or triggers on whose firing the query is executed, e.g. in OpenCQ [55]. Of course these could be also temporal events or an amount of tuples seen so far, but these could be also user-defined events, such as a fired alert or an incoming e-mail.

One main problem for operators processing streams is the fact that streams are unbounded. Especially, *blocking operators*, i.e., operators which do not produce a result tuple before they have seen all tuples on their inputs [11], are problematic. The set of these operators comprises aggregations, groupings, but also set operations, such as `NOT IN` or `NOT EXISTS` [52, 40]. For example, if we would like to calculate the average over the speed of the incoming C2X messages, a classical average operation has to wait until the stream of messages ends (but we do not know when the stream ends) to produce the desired result. In contrast, *non-blocking operators* produce results periodically or on arrival of new tuples, i.e., incrementally [51]. *Partially blocking operators* [52] are operators which can produce intermediate results but also a final result in the end. Obviously, a language for continuous queries can then only be based on non-blocking operators [52, 11]. But the set of non-blocking operators is neither in relational algebra nor in SQL sufficient for all expressible relational queries [52]. We discuss the completeness of languages in Section 3.3. Another type of operators which are harmful to continuous query processing are *stateful operators* [56] (in contrast to *stateless operators*). These operators, e.g., joins, require to store a state for their operation, which for streams is unbounded in size. Hence, the remedy to these problems is to approximate processing the stream as a whole as good as possible. One simple yet powerful approach is the partitioning of the stream into small portions, so-called *windows*. Each window is a finite bag of tuples and can be processed also by the common relational blocking operators. A second possibility is to provide incremental implementations of these operators, which are able to update the result with new tuples and output "intermediate" results. Finally, an approximation of the stream in form of a summary or *synopsis* can be used to operate on. In the following we will detail window operators and their semantics.

3.1.5 Windows

In continuous query languages based on SQL, windows are a crucial extension to the algebraic set of operators. It depends on the language which types of windows are supported. A window is always built according to some ordered *windowing attribute* which determines the order of elements included in the window [57, 56]. The type of window, i.e., how it is determined which elements are valid in the current window, according to [57] can be described by its measurement unit, the edge shift, and the progression step. The *measurement unit* can be either a number of x time units (*time-based window*) or tuples (*tuple-based window*) declaring that the elements with timestamps within the last x time units or the last x elements are valid for the window at the point in time of the query. We define a time-based window of size l similar to the definition in [57] as follows:

▶ **Definition 7.** (Time-based Window) A *time-based window* W_{l_T} (with window size $l_T \in \mathbb{T}$) over a stream S at time $\tau_i \in T$ is a finite multiset of stream elements with

$$W_{l_T}(S(\tau_i)) = \{< (s_k, \tau_k), m_k > \mid (s_k, \tau_k) \in U, \tau_i - l_T \leq \tau_k \leq \tau_i, \tau_k \geq \tau_0\},$$

where m_k is the multiplicity of the tuple in the subset and U is the support of stream $S(\tau_i)$.

We assume, that the stream elements are ordered by timestamp τ before a window operator is applied. The definition for a tuple-based window of size l_N is similar:

▶ **Definition 8.** (Tuple-based Window) Let $S(\tau_i) = \{< (s_0, \tau_0), m_0 >, \ldots, < (s_n, \tau_n), m_n >\}$, $\tau_i \geq \tau_n \geq \tau_0 \wedge \tau_j \geq \tau_{j-1}\ \forall j \in \{1, \ldots, n\}$, be the content of stream S at time τ_i. Then a *tuple-based window* W_{l_N} (with window size $l_N \in \mathbb{N}$) over stream S at time $\tau_i \in T$ is a finite multiset of stream elements with

$$\begin{aligned} W_{l_N}(S(\tau_i)) = \{< (s_k, \tau_k), m_k > \quad \mid \quad & (s_k, \tau_k) \in U, j \leq k \leq i, \\ & \exists m'_j, m''_j.\ m'_j \geq 0, m''_j > 0, \\ & m_j = m'_j + m''_j, \quad m''_j + \sum_{\ell=j+1}^{i} m_\ell = l_N\}. \end{aligned}$$

where $(s_n, \tau_n) \in U,\ \forall \tau_j \in U\ \tau_n > \tau_j$, and U is the support of stream $S(\tau_i)$.

This means, that we go backwards in the stream from time τ_i on and search for the last point time τ_n at which elements arrived. We now collect exactly N tuples going backwards in the stream. We assume, that the last tuple which (partially) fits into the window is $< (s_j, \tau_j), m_j >$. Then only the multiplicity portion m''_j which still fits into the window will be added to the window.

In the Aurora system [78, 3] value-based windows as a form of generalization of time-based windows are presented. These windows implement the idea of having a different windowing attribute instead of a timestamp – the attribute just has to be ordered. Furthermore, the value-based window should return only those tuples whose value of the particular attribute is within a specific interval (hence, value-based windows). A similar concept are predicate-windows [37] suited for systems which work with negative and positive tuples. A correlation attribute identifying the data in the tuple and a predicate condition are defined for the window, which enables to filter the tuples according to that predicate. The filtering may result in negative tuples, when the predicate has been fulfilled by tuples for the same correlation attribute value before but is not for the present query evaluation. *Partitioned windows* [53, 10] are applicable to time- or tuple-based windows and follow the idea of dividing the stream into substreams based on filter conditions and of windowing them separately. Afterwards, the windows of the substreams are unioned to one result stream [3].

The *edge shift* of a window describes the motion of the upper and lower bounds of the window. Each of them can either be fixed or moving with the stream. For example, in the most common variant, the *sliding window*, both bounds move, while for a *landmark window* one bound is fixed and one is moving.

Finally, the *progression step* or *periodicity* defines the intervals between two subsequent movements of a window. This again can either be *time-based* or *tuple-based*, e.g., the window can move every 10 seconds or after every 100 arrived tuples. When the contents of windows in each progression step are non-overlapping, this is termed a *tumbling window*, i.e., size and sliding step have an equal number of units. Windows can also be *punctuation-based.* A notification tuple sent with the stream indicates the window operator that it should evaluate. We will explain punctuations in Section 3.2.

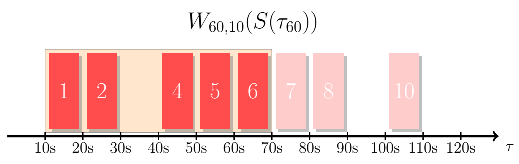
In the following, one of the most commonly used forms of a moving window, a sliding time-based window, is defined as:

▶ **Definition 9.** A *sliding time-based window* with window size $l_T \in \mathbb{T}$ and slide value $v \in \mathbb{T}$ over a stream S at time $\tau_i \in T$ is a finite multiset of stream elements with

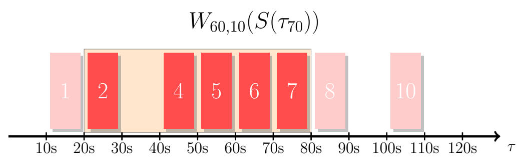
$$\begin{aligned} W_{l_T,v}(S(\tau_i)) = \{< (s_k,\tau_k), m_k > \quad | \quad & (s_k,\tau_k) \in U, \exists j \in \mathbb{N} : \tau_0 + j \cdot v \leq \tau_i, \\ & \tau_0 + (j+1) \cdot v > \tau_i, \tau_i \geq \tau_0 + l_T, \\ & \tau_0 + j \cdot v - l_T \leq \tau_k \leq \tau_0 + j \cdot v, \tau_k \geq \tau_0\}. \end{aligned}$$

where U is the support of stream $S(\tau_i)$

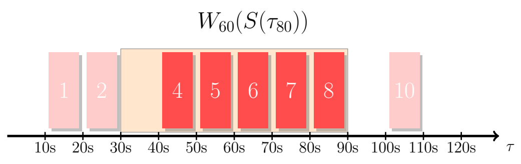
▶ **Example 10.** In our traffic example we would like to retrieve the number of C2X messages with `speed` greater than $30km/h$ from the last minute every 10 seconds. We realize this by defining a time-based sliding window W_{l_T}, v of size $l_T = 60s$ and with a sliding step of $v = 10s$ over the `CoCarMessage` stream. In Figure 5a the content of the window after 1 minute is shown. This is the first point in time, where the query with the window delivers results (hence, $\tau_i \geq \tau_0 + l_T$). The window contains the elements 1,2,4,5,6. After 10 more seconds the window slides, element 1 is dropped from the window and element 7 is added (Figure 5b). After another slide after 10 seconds element 2 is dropped and element 8 is added to the window (Figure 5c).


$W_{60,10}(S(\tau_{60}))$
1
2
4
5
6
7
8
10
10s 20s 30s 40s 50s 60s 70s 80s 90s 100s 110s 120s
τ

(a) Window after 60s


$W_{60,10}(S(\tau_{70}))$
1
2
4
5
6
7
8
10
10s 20s 30s 40s 50s 60s 70s 80s 90s 100s 110s 120s
τ

(b) Window after 70s



(c) Window after 80s

Figure 5 Example of a sliding window with size of $1min$ and slide step of $10s$.

Depending on the language, windows can either be implicitly included in an operator (see the definition of the Aggregate operator in Example 2, Figure 4a) or defined as a separate operator in the query language. An example of the latter is given in the data and query model SStream [41] of the system SStreaMWare [42]. Gürgen et al. enable to define windows a little bit different to provide a high flexibility and variety of window types. A window creation operator produces a set of windows on a stream according to a window description configuring the operator. The description is constituted of initial start and end parameters (time or tuples are possible), parameters for the advancement of start and end window edges, and a rate parameter which is analog to the slide parameter. The windows serve as input for windowed operators, such as aggregation operators.

In SQL-based declarative languages the window construct included in the SQL:2003 standard is extended, e.g., by a `SLIDE` keyword to enable the definition of a sliding step. Example 11 shows three different types of windows in different languages.

▶ **Example 11.**

```
CREATE STREAM C2XSpeeder
SELECT COUNT(*) As speederNo
OVER (RANGE 1 MINUTE PRECEDING SLIDE 10 SECOND]
FROM C2XMessage WHERE Speed > 30.0
```

Listing 2 A Time-based Sliding Window in ESL (Stream Mill)

```
SELECT Istream COUNT(*)
FROM C2XMessage[RANGE 100 SLIDE 100]
WHERE Speed > 30.0
```

Listing 3 A Tuple-based Tumbling Window in CQL (STREAM)

```
SELECT Count(*)
FROM C2XMessage <LANDMARK RESET AFTER 600 ROWS ADVANCE 20 ROWS>
WHERE Speed > 30.0
```

Listing 4 A Tuple-based Landmark Window in TruSQL (TruSQLEngine)

3.2 The Notions of Time and Order

We already mentioned before, that time plays an important role in DSMS. In all DSMS systems the processed tuples have some kind of timestamp assigned from a discrete and monotonic time domain. The timestamps allow then to determine if a tuple is in order or not and enable the definition of time-based windows [64].

3.2.1 Time

The prominent status of timestamps can already be seen from several semantics' definitions of streams. The timestamp is always handled as a specific attribute which is not part of the stream schema [57, 10, 50]. A monotonic time domain T can be defined as an ordered, infinite set of discrete time instants $\tau \in T$ [57, 9]. For each timestamp exists a finite number of tuples (but it can also be zero).

In the literature, there exist several ways to distinguish where, when, and how timestamps are assigned. First of all, the temporal domain from which the timestamps are drawn can be either a *logical time domain* or *physical clock-time domain*. Logical timestamps can be simple consecutive integers, which do not contain any date or time information, but serve just for ordering. In contrast, physical clock-time includes time information (e.g., using UNIX timestamps). Furthermore, systems differ in which timestamps they accept and use for internal processing (ordering and windowing). In most of the systems *implicit timestamps* [11, 78], also called *internal timestamps* or *system timestamps* are supported. Implicit timestamps are assigned to a tuple, when it arrives at the DSMS. This guarantees that tuples are already ordered by arrival time, when they are pipelined through the system. Implicit timestamps assigned at arrival in the system also allow for estimating the timeliness of the tuple when it is output [3]. Besides a global implicit timestamp (assigned on arrival), there exists also the concept of new (*local*) timestamps assigned at the input or output queue of each operator (time of tuple creation). This could also be implicitly expressed by the tuple's position in the queue [78]. In contrast, *explicit timestamps* [11, 78], *external timestamps* [15] or *application timestamps* [64] are created by the data stream sources and an attribute of the stream schema is determined to be the timestamp attribute. The authors of the Stream Mill language ESL [72, 15] use the term *explicit timestamp* in another way to discriminate between their concept of *latent timestamps* and internal and external timestamps. Latent timestamps are assigned on demand (lazily), i.e., only for operations dependent on a timestamp such as windowed aggregates [15], while explicit timestamps are assigned to every tuple. Example 12 shows, how the three types of timestamps are defined on creation of our C2XMsgs stream. In Listing 5 the creation timestamp `ts` is used as an explicit timestamp, designated by the `ORDER BY` clause. The query in Listing 6 uses an implicit timestamp by applying the function `current_time` and also designating it as order criterion. Listing 7 does not contain any information about an ordering attribute (no `ORDER BY` clause). Thus a timestamp will be assigned on demand.

▶ **Example 12** (Usage of the different timestamp types in ESL).

```
    CREATE STREAM C2XMgs(
            ts timestamp, msgID char(10), lng real,
            lat real, speed real, accel real)
    ORDER BY ts;
    SOURCE 'port5678';
```

Listing 5 Explicit Timestamp

```
    CREATE STREAM C2XMgs(
            ts timestamp, msgID char(10), lng real, lat real,
            speed real, accel real, current_time
            timestamp)
    ORDER BY current_time;
    SOURCE 'port5678';
```

Listing 6 Implicit Timestamp

```
    CREATE STREAM C2XMgs(
            ts timestamp, msgID char(10), lng real, lat real,
            speed real, accel real);
  SOURCE 'port5678';
```

Listing 7 Latent Timestamp

Depending on the semantics of a data stream, tuples can include more than one timestamp. For example, in the data stream definition of the CESAR language [26], which is an event-based stream algebra, a tuple contains two timestamps, τ_0 and τ_1, denoting the start and end of an event, respectively.

There are also systems (e.g., StreamBase) which additionally order the tuples for the same timestamp, for instance, by arrival order. Jain et al. define the order of tuples with respect to the timestamp even stricter – an additional ordering between batches of tuples with the same timestamp is specified [45]. This order was proposed because the authors noticed semantic inconsistencies for query results in systems which use a *time-driven model* (i.e., windows are evaluated on each new time instant) as well as in systems, which use a *tuple-driven model* (i.e., a window is evaluated on each new tuple).

An interesting question is how timestamps should be assigned to results of binary operators and aggregates to ensure semantic correctness. Babcock et al. propose two solutions to assign a timestamp to results of a join [11]. The first option is to use the creation time of a join output tuple when using an implicit timestamp model. The second option is to use the timestamp of the first table involved in the join in the FROM clause of the query can be used, which is suited for explicit and implicit timestamp models. For aggregates similar considerations can be made. For example, if a continuous or windowed minimum or maximum is calculated, the timestamp of the maximal or minimal tuple, respectively, could be used. When a continuous sum or count is calculated, the creation time of the result tuple or the timestamp of the latest element included in the result can be used. If an aggregate is windowed there exist additional possibilities. The smallest or the highest timestamp of the elements in the window can be used as they reflect the oldest timestamp or most recent timestamp in the window, respectively. Both maybe interesting, when timeliness for an output tuple is calculated, but which one to use depends obviously on the desired outcome. Another possibility would be to take the median timestamp of the window.

3.2.2 Order

Many of the systems and their operators rely on (and assume) the ordered arrival of tuples in increasing timestamp order to be semantically correct [64]. For example, in the STREAM system (using a time-driven execution model) time can only advance to the next time instant, when all elements in the current time instant have been processed [10]. This has been coined as the *ordering requirement* [64]. But as already pointed out, this can not be guaranteed especially for explicit timestamps and data from multiple sources. In the various DSMS basically two main approaches to the problem of disorder have been proposed.

One approach is to tolerate disorder in controlled bounds. The Aurora system, for example, does not assume tuples to be ordered by timestamp [3]. The system divides operators into *order-agnostic* and *order-sensitive* operators. The first group of operators does not rely on an ordering of elements, for instance, the Filter operation we already introduced in Example 2, which is a unary operation processing one tuple at a time. The order-sensitive operators are parametrized with a definition how unordered tuples should be handled. The definition contains the attribute which indicates the order of the tuples and a *slack* parameter. The slack parameter denotes, how many out-of-order tuples may arrive between the last and next in-order tuple. All further out-of-order tuples will be discarded. The order can also be checked for partitions of tuples, specified by an additional GROUP BY clause in the order definition. A general concept of a slack parameter, called *adherence parameter* has been presented for the STREAM system [7, 13]. The adherence parameter is a measure for how well a stream "adheres" to a defined constraint. The authors define a set of *k-constraints* one of which is the ordered-arrival-k-constraint. This constraint conforms to the slack parameter's ordering semantics.

The second way to handle disorder is to dictate the order of tuples and reorder them if necessary. While the use of implicit timestamps is a simple way of ordering tuples on arrival [64], the application semantics often requires the use of explicit timestamps though. *Heartbeats* [64] are tuples sent with the stream including at least a timestamp. These markers indicate to the processing operators, that all following tuples have to have a timestamp greater than the timestamp in the punctuation. As already mentioned in Section 2 the STREAM system includes an Input Manager which buffers stream elements based on heartbeats. The buffered elements are output in ascending order as soon as a heartbeat is received, i.e., they are locally sorted. The heartbeats can be created by the stream sources or by the Input Manager itself. Srivastava and Widom propose approaches to create heartbeats either periodically or based on properties of the stream sources and the network, such as transmission delay [64]. Heartbeats are only one possible form of *punctuation* [74]. Punctuations, in general, can contain arbitrary patterns which have to be evaluated by operators to true or false [74]. Therefore, punctuations can also be used for approximation. They can limit the evaluation time or the number of tuples which are processed by an otherwise blocking or stateful operator. For example, when a punctuation-aware join operator receives a punctuation it can match all elements received on the streams to join since the last punctuation. Other methods for reordering tuples in limited bounds use specific operators. Aurora's SQuAl language provides a sorting operator called BSort [3]. It orders tuples according to some attribute by applying a buffered bubble sort algorithm. Finally, the Stream Mill system leaves the handling of out-of-order tuples to the user [15]. It detects these tuples and adds them to a separate stream.

3.3 Completeness of Stream Query Languages

It has already been mentioned that blocking operators are not an option for query languages in DSMS [11]. Hence, SQL as is is not suited for DSMS, because sequence queries cannot be expressed [52, 51]. An interesting question therefore is, which queries can be formulated using only non-blocking operators of SQL. Continuous query semantics is based on the append-only principle. Therefore, the class of monotonic queries is different from the query classes which allow deletions and updates [70]. For a monotonic query Q holds, that the result of a query over the ordered stream S at time τ_i is included in the results of the query at time τ_{i+1} [70, 52, 51, 39] or formally expressed:

$$Q(S(\tau_i)) \subseteq Q(S(\tau_{i+1})), \forall \tau_i \in \mathbb{T}$$

Law et al. have proven that the class of monotonic queries over data streams can be expressed by queries using only non-blocking operators [52]. Non-blocking and monotonic operators in the relational algebra are obviously Selection and Projection. Law et al. also showed that a query which is monotonic for ordered streams is also monotonic for relations with respect to set containment and can therefore be expressed only with non-blocking operators [51]. It can be followed from this, that Union and Cartesian Product or Join also can be calculated by non-blocking operators as these are monotonic wrt. set containment [51], while Set Difference (which is non-monotonic) cannot. The intersection is a monotonic operator and can be either expressed in the relational algebra by non-blocking or blocking operators. In SQL also continuous forms of Count and Sum are non-blocking [52].

4 Data Quality in DSMS

We have motivated the demand for a new concept of data management systems by monitoring and tracking applications. These applications usually rely on sensors which create data streams by measuring values or recording multimedia. But the use of sensors also reveals problems. The produced data can be incorrect, unordered, and incomplete. Another source of inaccuracies in data streams can be the use of classification, prediction, ranking or any other sort of approximation algorithms [47, 48]. The unreliability of the data processed again leads to unreliable results of applications realized with a DSMS. Consider our example of traffic state estimation. We rely on positions determined by GPS devices. These devices usually introduce a measurement error in positioning and messages may also contain no position, when the GPS signal is lost. These errors are propagated through the entire data processing chain. For example, when the road and the section on which the message has been created is determined, the error can lead to an assignment to a wrong section. Inevitably, this will lead to inexact results when estimating the traffic state for the road. Hence, means to take inaccuracies into account and to rate the result of a query have to be considered. Not only the quality of the data values (we will call this *application-based data quality* in the following), but also the data stream management system performance has to be considered to assess the quality of query results. When the output delay in the DSMS is too high, e.g., the estimated traffic state will identify a traffic situation which is no longer present. Furthermore, if too many tuples are dropped by a load shedder to gain performance, statements based on a low number of data may also harm the result. Therefore, also the Quality of Service (QoS) for multiple performance aspects of a system has to be tracked and taken into account in query processing. Hence, we define data quality related to the system performance as *system-based data quality*. In the following, we will discuss data quality dimensions for application-based and system-based data quality. We will discuss solutions which enable to rate and to track the corresponding data quality dimensions. We will also briefly introduce our own ontology-based approach.

4.1 Application-based and System-based Data Quality Dimensions

As mentioned before, we distinguish data quality which is inherent in the data itself, i.e., it describes how reliable the data we process is, and data quality which represents the system performance of a DSMS. The type of the measured data quality is expressed by a *data quality dimension*. For example, the timeliness of data is a data quality dimension. For each data quality dimension a *data quality metric* defines a possible way to measure the quality within that dimension. There exists a plethora of classifications which structure and describe data quality dimensions, such as the Total Data Quality Management (TDQM) classification [75, 66], the Redman classification [60], or the Data Warehouse Quality (DWQ) classification [46]. For data streams, Klein and Lehner propose a dimension classification [49]. In Table 1 we list a non-exhaustive set of data quality dimensions, which we think are of importance for data quality rating in a traffic state estimation application realized by a DSMS (based on [49, 16]) and rate if they are application-based or system-based. Data quality for a dimension can be measured on different levels. It can be measured system-wide, e.g., the output rate, on operator level, e.g., the selectivity of an operator, or on window, tuple, or attribute level.

In the following, we will discuss some approaches of measuring and rating data quality in DSMS.

4.2 Quality of Service Monitoring in DSMS

In Section 2 we described that DSMS can implement means to monitor the system performance during query processing. Aspects which describe the performance of a DSMS are also termed *Quality of Service*. Quality of Service in general rates how good the component or system at

Table 1 Example Data Quality Dimensions.

Data Quality Dimension	**Informal Description**	**Example Metric**	**Application-based/ System-based**
Completeness	Ratio of missing values or tuples to the number of received values/tuples	The number of non-null values divided by all values including null values in a window	Application-based
Data Volume	The number of tuples or values a result is based on, e.g., the number of tuples used to calculate an aggregation	Quantity of tuples in a window	System-based and Application-based
Timeliness	The age of a tuple or value	Difference between creation time and current system time	System-based and Application-based
Accuracy	Indicates the accuracy of the data, e.g., a constant measurement error or the result of a data mining algorithm	An externally calculated or set value	Application-based
Consistency	Indicates the degree to which a value of an attribute adheres to defined constraints, e.g., if a value lies in certain bounds	Rule evaluation	Application-based
Confidence	Reliability of a value or tuple, e.g., the confidence to have estimated the correct traffic state	A weighted formula which is calculated from values for other data quality dimensions	Application-based

hand fulfils the constraints and requirements posed to it. QoS oriented systems then try to give guarantees for the QoS aspects and try to keep QoS within defined bounds by applying countermeasures. QoS dimensions (used here analogously to quality dimensions) have been classified by Schmidt in [62] into time-based and content-based dimensions. Schmidt identified the following time-based dimensions for DSMS: throughput (or data rate), output delay (or latency) and the following content-based dimensions: sampling (or drop rate), sliding window size, approximation quality and data mining quality. He defined two new time-based dimensions, called signal frequency (amount of information in a stream) and inconsistency (maximal difference between creation timestamp and system timestamp, which is similar to our example metric of timeliness in Section 4.1). In the Aurora system additionally a value-based QoS dimension is defined, which rates if important values have been output, i.e., they prioritize results and can therefore also prioritize values and adapt operators of the corresponding queries for these values [3]. To guarantee to stay in given bounds for the above dimensions, DSMS have several countermeasures, depending on the QoS dimension at hand.

For example, if the system is overloaded and the output delay or the throughput are low, a DSMS can drop tuples with sampling techniques, which can be quite simple (random) or very sophisticated, e.g., based on information about other QoS parameters as in the case of the value-based QoS [3]. The dropping of tuples is also called *load shedding*. Load shedding is done by placing load shedding operators into the query execution plan [12, 69] where required. Other possibilities are adaptive load distribution or admission control [69]. In the Aurora system for each QoS the administrator of the system is required to model a two-dimensional function with a QoS rating between 0 and 1 on the y-axis and the QoS dimension values on the x-axis for each output stream [3]. Additionally, a threshold for the QoS dimension has to be defined to indicate in which bounds QoS is acceptable. A similar approach is followed by the QStream system [61, 62]. In the QStream system two descriptors for each output stream are defined – a content quality descriptor which includes the dimension's inconsistency and signal frequency defined by Schmidt [62] and a time quality descriptor consisting of values for data rate and delay. The quality dimensions are calculated throughout the whole query process. All operators in the query execution plan comprise functions used to calculate the value of each quality dimension when new tuples are processed. At the end of the processing chain quality values are output for the result streams of each continuous query. A user can pose requirements on the result stream's quality by formulating a request describing thresholds for the dimensions. If the descriptors meet the quality request, this is reported to the user as a successful negotiation.

4.3 Inclusion of Data Quality in Data Streams

In the systems reviewed in the previous section, QoS information is handled separately from the stream data and can only be retrieved for the output streams. Furthermore, data quality dimensions are mostly system-based, i.e., the content of the data in the stream and corresponding application-based quality dimensions are not taken into account. In the successor system of Aurora, Borealis [1], the QoS model of Aurora has been refined to rate QoS not at the output streams of the system, but also in each operator in between. To this end, each tuple includes a vector with quality dimensions, which can be content-related (e.g., the importance of a message) or performance-related (e.g., processing time for a message up to this operator). The vectors can be updated by operators in the query execution plan and a score function is provided which can rate the influence of a tuple on the QoS based on a vector [1]. A crucial limitation of the previous approach is, that the quality dimensions in the vector are equal for each stream, which does not allow for an application-based data quality rating of stream contents. To achieve a more fine-granular rating of data quality on attribute, tuple and window level and to also include application-based data quality dimensions, Klein and Lehner [49] propose a different approach. They extended the PIPES system [50] by Krämer and Seeger with modified operators to include data quality dimensions as a part of the stream schema. Krämer and Seeger distinguish four different types of operators based on the operator's influence on the stream data (modifying, generating, reducing or merging operators). Changes on the data can in turn result in updates of the data quality of an attribute, tuple, or window. For each data quality dimension and each operator the influence of the operator is discussed and a function to calculate the new data quality is provided. The approach introduces so called *jumping data quality windows* which include a set of data quality dimensions and where for each attribute and window the size of the window can be defined independently [49]. In addition, Klein and Lehner make the size of the data quality windows dynamically adaptable based on an *interestingness factor*. The interestingness factor is dependent on the application realized and can for example shrink the window size when there are interesting peaks in the stream data to refine the granularity of quality information in this stream portion.

A drawback of Klein and Lehner is the deep integration of data quality dimensions and corresponding metrics into the operators. The implementation of operators has to be changed substantially to include the quality information. Therefore, we propose a more flexible solution, which allows to define custom quality dimensions and metrics based on an abstract data quality ontology. The ontology consists of two parts – a first part including data quality related concepts, such as Quality Metric, Quality Dimension and Quality Factor, and a second part comprising application specific concepts with relationships to the data quality concepts (e.g., an attribute is related to a certain quality dimension). The data quality meta information is loaded once when a continuous query is registered in the system and the data quality is then calculated according to the meta information throughout query processing. Similar to [49] we add the data quality information as separate attributes to the tuples in the stream. We identified three main tasks in the data stream management process, which can influence the data quality. First, in line with Klein and Lehner we parse the continuous query and identify relational operators in the query which modify the data stream and the data in it and hence, also the data quality. Second, in the ontology we allow to define rules, which rate the semantic consistency of attribute values or values of multiple related attributes in a tuple. For example, if we receive the current temperature and an indicator for snow, we can define a rule to check if the value for snow (yes/no) is consistent with the temperature given (no snow possible when above $3°C$). Third, also application specific and user-defined operations, such as data mining operations, are considered and the addition of data quality values can be easily integrated into the application code. The data quality metrics can be arbitrarily complex and can include several data quality factors and can also combine values of other data quality dimensions. This allows for flexible and application-dependent estimation of data quality. For example, in the traffic state estimation example we calculate a confidence value for the estimated traffic state based on multiple weighted data quality dimensions, where the weights allow to reflect the individual contribution to the overall quality. Finally, we made the calculation of data quality in the system optional – if not required, data quality information is not calculated at all. Our approach is described in detail in [36].

4.4 Probabilistic Data Streams

A completely different way of dealing with defective data in DSMS are probabilistic data streams. Uncertainty has been studied intensively for Database Management Systems and several systems (mainly research prototypes) implementing probabilistic query processing have been proposed. In data stream management this is a very recent topic which has been addressed only scarcely in literature compared to other research questions. Kanagal and Desphande [48] distinguished two main types of uncertainties in data streams (analogously to databases). First, the existence of a tuple in the stream can be uncertain (how probable is it for this tuple to be present at the current time instant?), which is termed by Kanagal and Despande *tuple existence uncertainty.* Second, the value of an attribute in a stream tuple can be uncertain, which is called *attribute value uncertainty* (what is the probability of attribute X to have a certain value?). In literature, mainly the tuple existence uncertainty is addressed. To model uncertainty for attribute values, for each attribute of a stream a random variable with a corresponding distribution function has to be introduced [48]. The probability of a tuple to consist of a certain configuration of values is then modeled by a joint distribution function, which multiplies the probabilities of the single attribute values. The tuple existence uncertainty can be modeled by a binary random variable, which can either be zero (is not included) or one (is included) and a probability distribution function [48].

▶ **Definition 13.** A *probabilistic stream* or *probabilistic sequence* is a sequence of tuples $S = < t_1, p_1 >, \ldots, < t_n, p_n >$, where p_i is the probability of the existence of a tuple t_i at position i modeled by a probability distribution function (pdf) [48, 47, 23].

The probability for a composition of the stream of a certain multiset of tuples at a certain point in time is again modeled by a joint probability distribution calculated from the pdfs of the tuples. Analogously to the possible worlds in probabilistic databases, from the random variables and the probability distributions, all *possible streams* [48, 47], i.e., deterministic instances of the stream, at a time τ and their probability can be determined.

One can easily see that the number of possible streams grows exponentially with the size of the stream and the number of possible tuples (#P-hard data complexity) [23]. For certain query operations, such as aggregates, on probabilistic streams, which are again probability distributions, it is therefore undesirable to calculate all the possible streams. The remedy to this problem are operators, which approximate the pdfs for the corresponding query operations. In [23] aggregates are computed using sketches, i.e., estimating frequency moments for the probabilistic stream. The data complexity of possible streams is even worse when the random variables are not independent of each other, i.e., if there is a correlation between tuples [48]. Kanagal and Desphande [48] propose to use directed graphical models (this can be, for example, Bayesian Networks) to describe correlated random variables, intermediate query results, and the dependencies between them. In the graphical model, the nodes depict the random variables and the edges are the dependencies. To limit the size of the graph and hence the computational complexity for a query, marginal distributions are used, which neglect unnecessary dependencies and infer new dependencies. Kanagal and Desphande extended the set of SQL operators by some probabilistic operators. For example, two new `SELECT` operators are provided, which can return a deterministic result for a query. The `SELECT-MAP` retrieves the stream with the highest probability of all possible streams, while `SELECT-ML` returns the possible stream which includes for each attribute of a tuple the value with the highest probability for that attribute. Their language also supports sliding windows and the set of common aggregates over the probabilistic data streams.

While the approaches before assumed possible stream semantics for streams with discrete data, these semantics is not valid for streams with uncertain continuous data (a problem of attribute-value uncertainty). The PODS system [73] addresses this issue by modeling each continuous-valued attribute as a continuous random variable with a corresponding pdf. To model pdfs for continuous random variables Tran et al. use Gaussian Mixture Models, which include several distributions for one variable. A bimodal distribution function, for example, can be approximated by two different Gaussian distribution functions. A probabilistic or uncertain data stream then consists of a sequence of tuples with discrete and continuous random variables for each attribute [73].

5 Conclusion

DSMS have demonstrated to be effective and efficient solutions dealing with huge amounts of rapidly incoming data which has to be processed in real-time. The new data management requirements of data stream applications have not only led to new concepts for data management architectures, but also to new and adapted query languages and semantics suited for streaming systems. Now, after almost two decades of research on continuous queries and data streams, many research prototypes have evolved and already some of them have turned into mature industry solutions. Many big players in the field of data management, such as Microsoft, IBM, or Oracle, have their own data stream management solution. However, despite some efforts, there is neither a common standard for query languages nor an agreement on a common set of operators and their semantics until today. Finally, data quality has turned out to be a crucial aspect in DSMS. Incoming data streams may be delivered by unreliable sources and results of a DSMS application can be falsified by data inaccuracies. Hence, data quality measurement, monitoring, and improvement solutions have to be integrated into DSMS and receive increasing attention. Since many interesting questions still remain open further research in the field is to be done and will follow.

Acknowledgments. I thank my colleagues Stefan Schiffer and Christoph Quix and the reviewers for their valuable comments.

References

1 Daniel J. Abadi, Yanif Ahmad, Magdalena Balazinska, Ugur Çetintemel, Mitch Cherniack, Jeong-Hyon Hwang, Wolfgang Lindner, Anurag Maskey, Alex Rasin, Esther Ryvkina, Nesime Tatbul, Ying Xing, and Stanley B. Zdonik. The Design of the Borealis Stream Processing Engine. In *Proc. 2nd Biennal Conference on Innovative Data Systems Research (CIDR)*, pages 277–289, Asilomar, CA, USA, 2005.

2 Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo F. Galvez, M. Hatoun, Anurag Maskey, Alex Rasin, A. Singer, Michael Stonebraker, Nesime Tatbul, Ying Xing, R. Yan, and Stanley B. Zdonik. Aurora: A data stream management system. In Halevy et al. [44], page 666.

3 Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.

4 Karl Aberer, Manfred Hauswirth, and Ali Salehi. A middleware for fast and flexible sensor network deployment. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *Proc. 32nd Intl. Conference on Very Large Data Bases (VLDB)*, pages 1199–1202. ACM Press, 2006.

5 Yanif Ahmad and Ugur Çetintemel. Data Stream Management Architectures and Prototypes. In Liu and Özsu [54], chapter D, pages 639–643.

6 A. Angel, N. Sarkas, N. Koudas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *Proceedings of the VLDB Endowment*, 5(6):574–585, 2012.

7 Arvind Arasu, Brian Babcock, Shivnath Babu, John Cieslewicz, Mayur Datar, Keith Ito, Rajeev Motwani, Utkarsh Srivastava, and Jennifer Widom. STREAM: The Stanford Data Stream Management System. Technical report, Stanford InfoLab, 2004.

8 Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. STREAM: The Stanford Stream Data Manager. In Halevy et al. [44], page 665.

9 Arvind Arasu, Shivnath Babu, and Jennifer Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. In *Proc. Intl. Conf. on Data Base Programming Languages*, 2003.

10 Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, 2006.

11 Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In Lucian Popa, editor, *Proc. 21st ACM Symposium on Principles of Database Systems (PODS)*, pages 1–16, Madison, Wisconsin, 2002. ACM Press.

12 Brian Babcock, Mayur Datar, and Rajeev Motwani. Load Shedding in Data Stream Systems. In Charu Aggarwal, editor, *Data Streams - Models and Algorithms*, chapter 7, pages 127–147. Springer, 2007.

13 S. Babu, U. Srivastava, and J. Widom. Exploiting k-constraints to reduce memory overhead in continuous queries over data streams. *ACM Transactions on Database Systems (TODS)*, 29(3):545–580, 2004.

14 Shivntah Babu and Jennifer Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–119, 2001.

15 Yijian Bai, Richard Chang Luo, Hetal Thakkar, Haixun Wang, and Carlo Zaniolo. *An Introduction to the Expressive Stream Language (ESL)*. UCLA, CS Department, 2004.

16 Norbert Baumgartner, Wolfgang Gottesheim, Stefan Mitsch, Werner Retschitzegger, and Wieland Schwinger. Improving situation awareness in traffic management. In *Proc. of the 30th Intl. Conf. on Very Large Databases*, 2010.

17 A. Bolles. A flexible framework for multisensor data fusion using data stream management technologies. In *Proceedings of the 2009 EDBT/ICDT PhD Workshop*, pages 193–200. ACM, 2009.

18 I. Botan, D. Kossmann, P.M. Fischer, T. Kraska, D. Florescu, and R. Tamosevicius. Extending XQuery with window functions. In *Proceedings of the 33rd international conference on Very large data bases*, pages 75–86. VLDB Endowment, 2007.

19 Don Carney, Ugur Centintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring streams - a new class of data management applications. In *Proc. 28th Intl. Conference on Very Large Data Bases (VLDB)*, Hong Kong, China, 2002. Morgan Kaufmann.

20 Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: continuous dataflow processing for an uncertain world. In *Proc. 1st Biennal Conference on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, 2003.

21 Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In Weidong Chen, Jeffrey F. Naughton, and Philip A. Bernstein, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 379–390, Dallas, Texas, 2000. ACM.

22 Mitch Cherniack and Stan Zdonik. Stream-oriented query languages and architectures. In Liu and Özsu [54], chapter S, pages 2848–2854.

23 Graham Cormode and Minos Garofalakis. Sketching probabilistic data streams. In Lizhu Zhou, Tok Wang Ling, and Beng Chin Ooi, editors, *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, Beijing, China, 2007. ACM Press.

24 Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In Halevy et al. [44], pages 647–651.

25 G. Cugola and A. Margara. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys (CSUR)*, 44(3):15, 2012.

26 Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker White. A general algebra and implementation for monitoring event streams. Technical report, Cornell University, 2005. http://hdl.handle.net/1813/5697.

27 Ramez A. Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 4th edition, 2004.

28 Opher Etzion and Peter Niblett. *Event Processing in Action*. Manning Publications Co., 2011.

29 European Union. Uniform provisions concerning the approval of vehicles with regard to the speedometer equipment including its installation. *Official Journal of the European Union*, 53:40–48, May 2010.

30 J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

31 Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. Spade: the system s declarative stream processing engine. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1123–1134, New York, NY, USA, 2008. ACM.

32 D. Geesen and M. Grawunder. Odysseus as platform to solve grand challenges: Debs grand challenge. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 359–364. ACM, 2012.

33 Sandra Geisler, Yuan Chen, Christoph Quix, and Guido G. Gehlen. Accuracy Assessment for Traffic Information Derived from Floating Phone Data. In *Proc. of the 17th World Congress on Intelligent Transportation Systems and Services*, 2010.

34 Sandra Geisler, Christoph Quix, and Stefan Schiffer. A data stream-based evaluation framework for traffic information systems. In Mohamed Ali, Erik Hoel, and Cyrus Shahabi, editors, *Proc. of the 1st ACM SIGSPATIAL International Workshop on GeoStreaming*, pages 11–18, November 2010.

35 Sandra Geisler, Christoph Quix, Stefan Schiffer, and Matthias Jarke. An evaluation framework for traffic information systems based on data streams. *Transportation Research Part C*, 23:29–55, August 2012.

36 Sandra Geisler, Sven Weber, and Christoph Quix. An ontology-based data quality framework for data stream applications. In *Proc. 16th Intl. Conf. on Information Quality (ICIQ)*, Adelaide, Australia, 2011.

37 T.M. Ghanem, W.G. Aref, and A.K. Elmagarmid. Exploiting predicate-window semantics over data streams. *ACM SIGMOD Record*, 35(1):3–8, 2006.

38 Lukasz Golab and M. Tamer Özsu. Issues in stream management. *SIGMOD Record*, 32:5–14, 2003.

39 Lukasz Golab and M. Tamer Özsu. *Data Stream Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

40 Y. Gurevich, D. Leinders, and J. Van den Bussche. A theory of stream queries. In *Proceedings of the 11th international conference on Database programming languages*, pages 153–168. Springer-Verlag, 2007.

41 L. Gürgen, C. Labbé, C. Roncancio, and V. Olive. Sstream: A model for representing sensor data and sensor queries. In *Int. Conf. on Intelligent Systems And Computing: Theory And Applications (ISYC)*, 2006.

42 L. Gürgen, C. Roncancio, C. Labbé, A. Bottaro, and V. Olive. Sstreamware: a service oriented middleware for heterogeneous sensor data management. In *Proceedings of the 5th international conference on Pervasive services*, pages 121–130. ACM, 2008.

43 Daniel Gyllstrom, Eugene Wu, Hee-Jin Chae, Yanlei Diao, Patrick Stahlberg, and Gordon Anderson. SASE: Complex Event Processing Over Streams. In *Proc. of 3rd Biennal Conference on Innovative Data Systems Research (CIDR)*, 2007.

44 Alon Y. Halevy, Zachary G. Ives, and AnHai Doan, editors. *Proc. ACM SIGMOD Intl. Conference on Management of Data*, San Diego, California, USA, 2003. ACM.

45 Namit Jain, Shailendra Mishra, Anand Srinivasan, Johannes Gehrke, Jennifer Widom, Hari Balakrishnan, Ugur Çetintemel, Mitch Cherniack, Richard Tibbetts, and Stanley B. Zdonik. Towards a streaming SQL standard. *Proceedings of the VLDB Endowment (PVLDB)*, 1(2):1379–1390, 2008.

46 Matthias Jarke, Manfred A. Jeusfeld, C. Quix, and Panos Vassiliadis. Architecture and Quality in Data Warehouses: An Extended Repository Approach. *Information Systems*, 24(3):229–253, 1999.

47 TS Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 346–355. Society for Industrial and Applied Mathematics, 2007.

48 Bhargav Kanagal and Amol Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *Proc. of IEEE Intl. Conf. on Data Engineering*, 2009.

49 A. Klein and W. Lehner. Representing data quality in sensor data streaming environments. *ACM Journal of Data and Information Quality*, 1(2):1–28, September 2009.

50 Jürgen Krämer and Bernhard Seeger. Semantics and implementation of continous sliding window queries over data streams. *ACM Trans. on Database Systems*, 34:1–49, 2009.

51 Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Relational Languages and Data Models for Continuous Queries on Sequences and Data Streams. *ACM Transactions on Embedded Computing Systems*, 36(3):1–31, March 2011.

52 Y.N. Law, H. Wang, and C. Zaniolo. Query languages and data models for database sequences and data streams. In *Proceedings of the 30th Intl. Conf. on Very large data bases*, pages 492–503. VLDB Endowment, 2004.

53 J. Li, D. Maier, K. Tufte, V. Papadimos, and P.A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 311–322. ACM, 2005.

54 Ling Liu and M. Tamer Özsu, editors. *Encyclopedia of Database Systems*. Springer, 2009.

55 Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Trans. on Knowl. and Data Eng.*, 11(4):610–628, 1999.

56 David Maier, Jin Li, Peter Tucker, Kristin Tufte, and Vassilis Papadimos. Semantics of data streams and operators. In *ICDT 2005*, number 3363 in LNCS, pages 37–52. Springer, 2005.

57 Kostas Patroumpas and Timos K. Sellis. Window specification over data streams. In *Current Trends in Database Technology - EDBT 2006 Workshops*, pages 445–464, 2006.

58 Feng Peng and Sudarshan S. Chawathe. Xsq: A streaming xpath engine. Technical report, Computer Science Department, University of Maryland, 2003.

59 Vijayshankar Raman, Amol Deshpande, and Joseph M. Hellerstein. Using state modules for adaptive query processing. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *Proceedings of the 19th International Conference on Data Engineering (ICDE)*, pages 353–366, Bangalore, India, 2003. IEEE Computer Society.

60 Thomas C. Redman. *Data Quality for the Information Age*. Artech House, Boston, 1996.

61 S. Schmidt, H. Berthold, and W. Lehner. Qstream: Deterministic querying of data streams. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *Proc. 30th Intl. Conference on Very Large Data Bases (VLDB)*, pages 1365–1368, Toronto, Canada, 2004. Morgan Kaufmann.

62 Sven Schmidt. *Quality-of-Service-Aware Data Stream Processing*. PhD thesis, Technischen Universität Dresden, 2006.

63 D. Singh, A. Ibrahim, T. Yohanna, and J. Singh. An overview of the applications of multisets. *Novi Sad Journal of Mathematics*, 37(3):73–92, 2007.

64 Utkarsh Srivastava and Jennifer Widom. Flexible time management in data stream systems. In Alin Deutsch, editor, *Proc. 23rd ACM Symposium on Principles of Database Systems (PODS)*, pages 263–274, Paris, France, 2004. ACM.

65 Michael Stonebraker, Ugur Çetintemel, and Stanley B. Zdonik. The 8 requirements of real-time stream processing. *SIGMOD Record*, 34(4):42–47, 2005.

66 D.M. Strong, Y.W. Lee, and R.Y. Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

67 J. Sun, C. Faloutsos, S. Papadimitriou, and P.S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696. ACM, 2007.

68 Apostolos Syropoulos. Mathematics of multisets. In *Proceedings of the Workshop on Multiset Processing*, pages 286–295, 2000.

69 N. Tatbul, U. Çetintemel, and S. Zdonik. Staying fit: Efficient load shedding techniques for distributed stream processing. In *Proceedings of the 33rd international conference on Very large data bases*, pages 159–170. VLDB Endowment, 2007.

70 Douglas Terry, David Goldberg, David Nichols, and Brian Oki. Continuous queries over append-only databases. In *ACM SIGMOD 1992*, 1992.

71 Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. Continuous queries over append-only databases. In Michael Stonebraker, editor, *Proc. ACM SIGMOD International Conference on Management of Data*, pages 321–330, San Diego, CA, 1992. ACM Press.

72 H. Thakkar, B. Mozafari, and C. Zaniolo. Designing an inductive data stream management system: the stream mill experience. In *Proceedings of the 2nd international workshop on Scalable stream processing system*, pages 79–88. ACM, 2008.

73 T.T.L. Tran, L. Peng, B. Li, Y. Diao, and A. Liu. PODS: a new model and processing algorithms for uncertain data streams. In *Proceedings of the 2010 international conference on Management of data*, pages 159–170. ACM, 2010.

74 P.A. Tucker, D. Maier, T. Sheard, and L. Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering*, pages 555–568, 2003.

75 R.Y. Wang and D.M. Strong. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems*, 12(4):5–33, 1996.

76 J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

77 C. Zaniolo. Logical foundations of continuous query languages for data streams. *Datalog in Academia and Industry*, pages 177–189, 2012.

78 Stan Zdonik, Peter Sibley, Alexander Rasin, Victoria Sweetser, Philip Montgomery, Jenine Turner, John Wicks, Alexander Zgolinski, Derek Snyder, Mark Humphrey, and Charles Williamson. Streaming for dummies. http://list.cs.brown.edu/courses/csci2270/archives/2004/papers/paper.pdf, 2004.

A Scientific and Commercial DSMS

In the following tables scientific and commercial DSMS are listed with their predominant properties.

A.1 Research Projects

Table 2 Data Stream Management Systems – Research Projects.

Project	Research Group	Runtime	Description
Tapestry [71]	Xerox Parc (Terry, Goldberg et al.)	1992–?	Uses a commercial append-only database, cont. querying by using stored procedures
NiagaraCQ [21] (`http://research.cs.wisc.edu/niagara`)	University of Wisconsion-Madison (Chen, DeWitt et al.)	2000–2002	Distributed ystem for continuous queries over web resources in XML format. Queries are executed periodically after a fixed amount of time or on arrival of new data. Windows are not supported. Language: XML-QL
Gigascope [24]	AT&T Labs and CMU (Cranor, Johnson, Srivastava et al.)	Ongoing	Specifically conceptualized for network monitoring applications. Language: GSQL
OpenCQ [55] (`http://www.cc.gatech.edu/projects/disl/CQ`)	College of Computing at the Georgia Institute of Technology (Liu, Pu et al.)	1999–2002	System to execute continuous queries on a trigger basis, i.e., when a trigger condition is fulfilled (database modifications, time events, or user-defined events), a query is executed.
TelegraphCQ [20] (`http://telegraph.cs.berkeley.edu`)	UC Berkeley (Hellerstein, Franklin et al.)	2000–2007	Reuses components from DBMS PostgreSQL, dataflows composed of set of operators (e.g., Eddy, Join) connected by Fjords, Language: SQL, scripts
STREAM [7] (`http://infolab.stanford.edu/stream`)	Stanford University (A. Arasu, J. Widom, B. Babcock, S. Babu et al.)	2000–2006	Probably the most famous one, comprehensible abstract semantics description; Language: CQL
Aurora/Borealis [2, 1] (`http://www.cs.brown.edu/research/borealis`)	Brown Univ., Brandeis Univ., MIT (Abadi, Cherniack, Madden, Zdonik, Stonebraker et al.)	2003–2008	Distributed system, uses notions of arrows, boxes and connection points for operator networks; Commercial: StreamBase; Language SQuAl
PIPES [50] (`http://dbs.mathematik.uni-marburg.de/Home/Research/Projects/PIPES`)	Universität Marburg (Seeger, Krämer et al.)	2003–2007	Commercial: RTM Analyzer; Language: PIPES, define logical and physical query algebra on multisets, use algebraic optimizations
System S/SPC/SPADE [31] (`http://researcher.watson.ibm.com/researcher/view_project.php?id=2531`)	IBM T.J. Watson Research	Started 2006; Ongoing	Distributed System, notion of operator network, Commercial: InfoSphere Streams; Language: SPADE/SPL
StreamMill [72] (`http://wis.cs.ucla.edu/wis/stream-mill/index.php`)	UCLA (H. Takkhar, C. Zaniolo)	Ongoing	Inductive DSMS (mining implementable with SQL and UDAs), support for XML data;Language: ESL
Global Sensor Networks [4] (`http://sourceforge.net/apps/trac/gsn/`)	EPF Lausanne, Digital Enterprise Research Insitute (DERI) (Salehi, Aberer et al.)	Ongoing	Wraps existing rel. DBMS with stream functionality; language: SQL
Odysseus [32] (`http://odysseus.offis.uni-oldenburg.de:8090/display/ODYSSEUS/Odysseus+Home`)	Carl von Ossietzky Universität Oldenburg (A. Bolles, D. Geesen, M. Grawunder, J. Jacobi, D. Nicklas)	Ongoing	Framework to create custom DSMS by extending and adapting architectural components.
SStreaMWare [41, 42]	France Telecom and LIG Laboratory (L. Gürgen et al.)	2006-2008	Service-oriented framework focusing on sensor data processing; Relational with a generic schema for sensor measurements and meta-data. Language: SQL-based

A.2 Commercial Systems

Table 3 Data Stream Management Systems – Commercial Products.

System	Company	Based on	Description
InfoSphere Streams `http://www-01.ibm.com/software/data/infosphere/streams/`	IBM	System S / SPADE / SPC	Current version: 3.0; Stand-alone product, supports only Linux, queries over structured and unstructured data sources; Language: SPL (successor of SPADE)
Oracle Event Processing (OEP) (`http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/index.html`)	Oracle	–	Standalone product. Current version: 11gR1; Available for various OS; based on Java and XML, Eclipse development plugin available (also for visual stream graph composition); Languages: CQL/EPL (Event processing language)
StreamInsight `http://msdn.microsoft.com/de-de/library/ee362541.aspx`	Microsoft	–	Standalone product. Needs SQL Server 2012 product key. Current version: 2.0 Languages: .NET, LINQ
StreamBase `http://www.streambase.com`	StreamBase	Aurora/Borealis	Stand-alone products (Server, Studio, Adapters, LiveView,...); Language: StreamSQL
TruSQL Engine	Truviso	TelegraphCQ	has been acquired by Cisco and integrated into Cisco Prime, a network management software bundle; Language: StreaQL
Esper (Open Source) `http://esper.codehaus.org`	EsperTech	–	Available for .NET and Java, Standalone product; Language: EPL
SAP Sybase Event Stream Processor (`http://www.sybase.com/products/financialservicessolutions/complex-event-processing`)	Sybase	–	Current Version: 5.1;Language: CCL (Continuous Computation Language)