FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Data Exchange of Relational Data and Beyond

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering/Internet Computing

eingereicht von

Johannes Maschler

Matrikelnummer 0725953

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Mitwirkung: Univ.Ass. Dipl.-Ing. Emanuel Sallinger

Wien, 09.10.2013

(Unterschrift Verfasserin) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Data Exchange of Relational Data and Beyond

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering/Internet Computing

by

Johannes Maschler

Registration Number 0725953

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Mag.rer.nat. Dr.techn. Reinhard Pichler
Assistance: Univ.Ass. Dipl.-Ing. Emanuel Sallinger

Vienna, 09.10.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Johannes Maschler
Spengergasse 27, 1050 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Danksagung

Ein besonderer Dank gilt meinen beiden Betreuern, Herrn Prof. Dr. Reinhard Pichler und Herrn Univ.Ass. Dipl.-Ing. Emanuel Sallinger, die mich bei der Erstellung dieses Dokuments mit sehr viel Engagement und unermüdlichem Einsatz geleitet haben. Ebenfalls bedanken möchte ich mich bei Herrn Univ.Ass. MSc. Vadim Savenkov der für mich ein Ansprechpartner war als meine Betreuer kurzeitig nicht verfügbar waren.

Desweiteren gebührt ein großer Dank meinen Eltern für ihre moralische und finanzielle Unterstützung.

Abstract

Data exchange is the problem where data conforming to a source schema is transformed into data conforming to a target schema. Although the data exchange problem was already described in the 1970s, its logical foundations have not been studied until very recently. A bit over a decade ago data exchange received new significance with the increasing importance of the internet and the various data formats that it has brought with it. Since then, data exchange has become a very active area of database research. In the first place, data exchange of relational data has been investigated. More recently, data exchange research has been extended to other data models such as knowledge bases and XML.

The goal of this master's thesis is to summarize the state of the art in data exchange. Therefore, the available literature is surveyed and a selection of the most important results in this area of research is presented. The main issues that will be discussed are finding a proper formalism for expressing expressing the relationship between the source and the target schema, validating the solutions, materializing solutions, and answering queries posed over the target schema with respect to the source data. We focus in the range of this work on data exchange over relational data, knowledge bases and XML data.

Kurzfassung

Data Exchange ist das Problem in dem Daten mit einer Struktur entsprechend eines Startschemas auf eine Art und Weise umgewandelt werden, sodass die Struktur der transformierten Daten dem eines Zielschemas entsprechen. Obwohl das Problem bereits in den 1970er Jahren beschrieben wurde, waren die formalen Grundlagen von Data Exchange bis vor kurzem unerforscht. Vor etwas mehr als einem Jahrzehnt erlangte das Data Exchange Problem, durch die schnelle Ausbreitung des Internets und den verschiedenen neuen Datenformaten welche es mit sich brachte, neue Wichtigkeit. Seitdem ist Data Exchange eines der meist diskutierten Problemen in der Datenbankforschung. In erster Linie wurde Data Exchange für das relationale Datenmodell untersucht. Vor kurzem jedoch wurde die Data Exchange Forschung auf weiter Datenmodelle erweitert, wie zum Beispiel auf Wissensdatenbanken (Knowledge Bases) und XML.

Das Ziel dieser Diplomarbeit ist den Stand der Technik von Data Exchange aufzuzeigen. Deshalb werden verfügbare wissenschaftliche Arbeiten begutachtet und eine Auswahl der wichtigsten Ergebnisse präsentiert. Die zentralen Probleme in Data Exchange sind: das Finden eines geeigneten Formalismus um die Beziehung des Startschemas mit dem Zielschema ausdrücken zu können, das Erkennen von Lösungen, die Konstruktion von Lösungen und das Beantworten von Abfragen über das Zielschema in Hinblick auf die Startdaten. Der Fokus dieser Arbeit liegt auf Data Exchange über relationale Daten, Wissensdatenbanken und XML Daten.

Contents

CHAPTER 1

Introduction

Data processing was one of the earliest applications for the computer technology. As the amount of data increased, the need to handle data in a structured way emerged. In particular, data values associated with each other should be managed in a way, such that similar data can be handled in a similar way. The key concept therefore were *schemata*, which are essentially specifications of data structures. Hence, similar data is structured such that they conform to the same schema.

Data exchange is the problem where data conforming to a source schema is transformed into data conforming to a target schema. Data exchange appears every time when data sets have to be exchanged between two applications. One of its earliest descriptions was in the 1970s in the EXPRESS system [62], which exchanged data between hierarchical schemata. Since then the data exchange problem remained recurrent. A bit over a decade ago data exchange received new significance with the increasing importance of the internet and the various data formats that it has brought with it.

For long time data exchange was mainly solved in a procedural way. The main reason for that was that although data exchange was a prevalent problem, its logical foundations had not been studied. According to [14] there are two explanations for that. On the one hand database research was concentrated on scenarios with a single database. On the other hand, there was no adequate formal model for data exchange known. Fagin, Kolaitis, Miller and Popa [37] presented at ICDT 2003 for relational data such a model, which was soon extended for other data formats.

Fagin et al. [37] based their formal model for data exchange on schema mappings. A *schema mapping* is a triple consisting of a schema for the source data, a schema for the target data, and a set of logical formulae that describe the relationship between both schemata. Schema mappings are the main building blocks for many other data management applications that involve sharing or transformation of data such as information integration [52], metadata management [19], and peer-to-peer data management systems [47].

The data integration problem [52] is the most closely related problem to data exchange. In *data integration* data is located in different sources and gets accessed in a unified way through a single virtual view. It is specified by a schema mapping consisting of a local schema, a global schema, and logical formulae that connect the local schema and the global schema. The main difference between data exchange and data integration is that in data exchange source data is actually transformed into target data, while in data integration the target is virtual and data remains in the source. Data exchange and data integration have many properties in common which recently have been studied in [43].

Data exchange was initially investigated over the relational data model [37] and was later adapted to the XML data model [9], to knowledge bases [10] and to graph databases [16]. Data exchange over relational data was extensively studied and is nowadays well understood. Data exchange over the XML data model also received substantial attention, but there remain still open questions. The research on data exchange with knowledge bases and graph databases started quite recently and accordingly there is still room for discussion.

The goal of this master's thesis is to summarize the state of the art in data exchange. The main issues that will be discussed are finding a proper formalism for expressing the dependencies, validating the solutions, materializing solutions, and answering queries posed over the target schema with respect to the source data. Therefore, the available literature is surveyed and a selection of the most important results in this area of research is presented. We focus in the range of this work on data exchange over relational data, knowledge bases and XML data.

This thesis is organized as follows. Chapter 2 fixes the notation of the relational data model and the XML data model. It further recapitulates the concepts of incomplete data and query answering. Chapter 3 discusses data exchange over relational data and further subsequent problems, like query answering or the composition of two schema mappings. Chapter 4 studies data exchange over knowledge bases. The knowledge bases used in this scenario are an extension of relational data and allow us to reuse many of the techniques from Chapter 2. Data exchange over XML trees is investigated in Chapter 5. Even though XML also generalizes the relational data model, many new formalisms have to be found to cover the tree structure of XML. Finally, Chapter 6 recapitulates the discussed results and gives some concluding remarks.

CHAPTER 2

Preliminaries

2.1 Relational Data

2.1.1 Data Model

In this section we recapitulate the basic characterization of the relational data model [63, 26]. The relational data model separates the structure of data and it content.

A *schema* $\mathbf{S}$ is a finite sequence of relation symbols $\langle R_1, \ldots, R_n \rangle$. Each *relation symbol* R_i has a finite set of attributes. We say relation symbol R_i has arity m, or for short R_i is m-ary, if R_i contains m attributes. Moreover, we associate each schema with a countably infinite *domain* $\mathbf{D}$.

An *instance* I over the schema $\mathbf{S}$ assigns to every m-ary relation symbol R_i in $\mathbf{S}$ an m-ary relation R_i^I. A *relation* R_i^I is a set of m-ary tupels. A *tuple* maps all m attributes of R_i to a value in the corresponding domain $\mathbf{D}$. We write $R_i^I(t)$ to denote that a tuple t occurs in relation R_i^I and call this occurrence a *fact*. As an alternative of saying that instance I is considered over schema $\mathbf{S}$, we sometimes say that instance I conforms to schema $\mathbf{S}$. We denote with $\mathsf{Inst}(\mathbf{S})$ the set of all instances conforming to schema $\mathbf{S}$. We write $\mathbf{D_a}$ for the *active domain of* I, i.e. the subset of values from $\mathbf{D}$ that are appearing in I.

If we have a schema $\mathbf{S}$ with relations $\langle S_1, \ldots, S_m \rangle$ and a schema $\mathbf{T}$ with relations $\langle T_1, \ldots, T_n \rangle$, we denote with $\langle \mathbf{S}, \mathbf{T} \rangle$ the combined schema $\langle S_1, \ldots, S_m, T_1, \ldots, T_n \rangle$. Analogously, if we have instance I over $\mathbf{S}$ and instance J over $\mathbf{T}$, we denote with $\langle I, J \rangle$ an instance K over $\langle \mathbf{S}, \mathbf{T} \rangle$ so that $S_i^I = S_i^K$ and $T_j^J = T_{i+j}^K$, for $1 \leq i \leq m$ and $1 \leq j \leq n$. Furthermore, if we consider two schemas, say $\mathbf{S}$ and $\mathbf{T}$, we always assume that both schemas have no relations symbols in common. We will sometimes abuse the notation and write R_i for the relation symbol and for the relation itself. This inaccurateness is common in the literature, since it is almost always clear from the context if the relation or the relation symbol is meant.

2.1.2 Incompleteness

An instance in the relational data model contains values from a domain consisting of a countably infinite set of *constants*, denoted with Const. An instance constructed only with those constants is called *complete*. In order to widen practicability of the relational data model, we must be able to represent incomplete information [28]. As it is common in the database community, we consider incomplete information in instances as present but unknown values. Thus, an instance with incomplete information can be interpreted by complete instances, where the unknown information is represented as elements from Const. The notion of *representation systems* [49, 6, 10] allows to define objects with different interpretations. A representation system is a tuple $(\mathbf{W}, \mathsf{rep})$, where $\mathbf{W}$ is a set of representatives, and rep is a function that assigns to every element in $\mathbf{W}$ a set of interpretations. In the literature are many different formalisms proposed which define how the representatives $\mathbf{W}$ and the function rep have to look like [49, 2]. We restrict us to the two for data exchange most relevant representation systems, namley naive and conditional tables. We stay close to the notation for incomplete information used in [10].

In the first approach we represent incomplete information with elements from a set of *variables*, denoted as Var. The literature refers to the elements of Var also often with the term labeled nulls. A *naive instance* I of a schema $\mathbf{S}$ assigns to every m-ary relation symbol R in $\mathbf{S}$ an m-ary relation $R^I \subseteq (\mathsf{Const} \cup \mathsf{Var})^m$, where Const and Var are assumed to be disjoint. The intuition is that if we do not know the exact value for an attribute in a fact we use an element from Var as replacement for the concrete value from Const. Thus, a naive instance represents a possibly infinite set of complete instances, where in each complete instance the elements from Var are replaced with elements from Const. We call such complete instances, which describe an interpretation of a naive instance, *possible worlds*. More formally, let $\mathsf{vars}(I)$ be the set of variables occurring in I, and let $\nu : \mathsf{vars}(I) \to \mathsf{Const}$ be a variable substitution. We denote with $\nu(R^I)$ the fact R^I, where each variable n is replaced by its image $\nu(n)$. The set of possible worlds for a naive instance I over a schema $\mathbf{S}$ is defined as follows:

$$\mathsf{rep}_{naive}(I) = \{J \in \mathsf{Inst}(\mathbf{S}) \mid \text{there is a } \nu : \mathsf{vars}(I) \to \mathsf{Const} \text{ such that } \nu(R^I) \subseteq R^J \text{ holds for every } R \in \mathbf{S}\}$$

It is important to notice that we are allowed to reuse variables in an instance to express that the same unknown element appears several times in an instance.

The second approach extends the concept of naive instances with *local conditions*. Local conditions are Boolean expressions with atoms of the form $x = y$ or $x \neq y$ with $x \in \mathsf{Var}$, $y \in (\mathsf{Const} \cup \mathsf{Var})$ and Boolean connectives $\wedge$ and $\vee$. A *conditional instance* I assigns to every m-ary relation symbol R in $\mathbf{S}$ a pair (R^I, ρ_R^I), where $R^I \subseteq (\mathsf{Const} \cup \mathsf{Var})^m$ and ρ_R^I is a function that assigns to every fact $R^I(t)$ a local condition. The semantics of the conditional instances is that every possible world contains only those facts for which

the local condition is satisfied. More formally, we denote with $\nu(R^I, \rho_R^I)$ the set R^I of facts, where every variable n is replaced by its image $\nu(n)$ and the local condition is satisfied. The set of possible worlds for a conditional instance I over a schema $\mathbf{S}$ is defined as follows:

$$\mathsf{rep}_{cond}(I) = \{J \in \mathsf{Inst}(\mathbf{S}) \mid \text{there is a } \nu : \mathsf{vars}(I) \to \mathsf{Const} \text{ such that } \nu(R^I, \rho_R^I) \subseteq R^J \text{ holds for every } R \in \mathbf{S}\}$$

We call a local condition *positive* if it does not contain an atom of the form $x \neq y$. An important fragment of conditional instances represent *positive conditional instances*, where the local condition is positive for every corresponding fact. Accordingly, the function rep_{pos} is defined similar to the function rep_{cond} with the difference that ρ_R^I assigns to every fact $R^I(t)$ a positive local condition. Let $\mathbf{W}_{naive}$, $\mathbf{W}_{cond}$ and $\mathbf{W}_{pos}$ be the set of all possible naive, conditional and positive conditional instances, respectively. Then $\mathcal{R}_{naive} = \{\mathbf{W}_{naive}, \mathsf{rep}_{naive}\}$, $\mathcal{R}_{cond} = \{\mathbf{W}_{cond}, \mathsf{rep}_{cond}\}$ and $\mathcal{R}_{pos} = \{\mathbf{W}_{pos}, \mathsf{rep}_{pos}\}$ are representation systems. As umbrella term for naive, conditional and positive conditional instances we use the term *incomplete instances*.

2.1.3 Query Answering

In this section we discuss query answering over incomplete data. In particular we highlight the semantics of querying over incomplete data and recall the query languages relational algebra and relational calculus. Furthermore, we return to mind two important special cases of both query languages. We stay close to the statements proposed in [64].

Let J be a complete instance over schema $\mathbf{S}$. A *query q over* $\mathbf{S}$ is a mapping that transforms instance J into a single relation. We say to this relation *answer of q over J*, and denote it with $q(J)$. If the schema and the querying instance is clear from the context, we use sometimes *query q* and *answer of q* as abbreviation. We are now interested in extending this semantics to instances containing incomplete information. In Section 2.1.2 we have argued that an incomplete instance I represents a set of complete instances, $\mathsf{rep}_{naive}(I)$, $\mathsf{rep}_{cond}(I)$ or $\mathsf{rep}_{pos}(I)$ respectively. Accordingly, the result of a query q over an incomplete instance I is a set of answers, consisting of the answers of the application of q to every possible world from $\mathsf{rep}(I)$. Usually, we are interested in the *certain answer of q with respect to I*, denoted as $\mathsf{certain}(q, I)$, i.e., those tuples that occur in the answer of every possible world.

Although, we have determined the semantics of queries over incomplete instances, we have to deal with the fact that a set of answers might consist of infinitely many elements, since incomplete instances have in general infinitely many interpretations. It seems natural to use the formalism of representation systems to express such sets of answers with single relations. As a further relaxation we require that an incomplete answer relation has to preserve only certain answers. More formally, the following equality must

hold:

$$\bigcap \mathsf{rep}(q(I)) = \bigcap \{q(J) \mid J \in \mathsf{rep}(I)\}$$

If this property is fulfilled by a representation system is depending on the features used in the query. If it is the case we say that we have a *weak representation system*. This concept was initially proposed by [49].

We recapitulate now two relevant query languages and present their basic properties. As first query language we consider *relational algebra*, that can be seen as procedural approach for querying data. Let I be an instance over a schema $\mathbf{S}$. Let R_1^I, R_2^I be two arbitrary relations in I and R_1, R_2 their corresponding relation symbols in $\mathbf{S}$. We denote with $t[X]$ the restriction of tuple t from R_1^I to X, where $X \subseteq R_1$. A query over I expressed in relational algebra is constructed with the following operators:

1. *Projection*: $\pi_X(R_1^I) = \{t[X] \mid t \in R_1^I\}$, where $X \subseteq R_1$.
2. *Selection*: $\sigma_\psi(R_1^I) = \{t \in R_1^I \mid \psi(t) = \mathbf{true}\}$, where ψ is a Boolean expression composed of logical connectives $\wedge$, $\vee$, $\neg$ and atoms of the form $A = a$ or $A = B$, such that $A, B \in R_1$ and $a \in \mathsf{Const}$.
3. *Union*: $R_1^I \cup R_2^I$, where $R_1 = R_2$ and $\cup$ is the set-theoretical union.
4. *Join*: $R_1^I \bowtie R_2^I = \{t \in R_3^{\bowtie} \mid R_3 = R_1 \cup R_2 \wedge t[R_1] \in R_1^I \wedge t[R_2] \in R_2^I\}$
5. *Renaming*: $\rho_{B \leftarrow A}(R_1^I) = \{\rho_{B \leftarrow A}(t) \mid t \in R_1^I\}$, where $\rho_{B \leftarrow A}(t)$ yields a tuple t' identical to t over the schema $(R_1 \backslash \{A\}) \cup \{B\}$.
6. *Difference*: $R_1^I - R_2^I = R_1^I \backslash R_2^I$, where $\backslash$ is the set-theoretical difference.

If we want to indicate that an expression uses only a subset of operators, we use their first letter as abbreviation. For instance if we have a query that uses only projection, selection and renaming, we say that the query is a PSR-expression. Furthermore, we denote a restricted version of the selection operator, where in ψ the negation is disallowed, with S^+. Imielinski and Lipski [49] showed among other things the following two results:

Theorem 2.1. [49] *PS^+UJR-expressions are the largest fragment of relational algebra such that naive instances form a weak representation system.*

This result states that if a query contains a difference operator or a selection with negation, then the resulting naive instance might not preserve the certain answers.

Theorem 2.2. [49] *Conditional instances form a weak representation system for relational algebra.*

It follows that conditional instances have more expressive power than naive instances. However, for many problems naive instances are sufficient and entail lower complexity bounds.

The second recalled query language is the *relational calculus* that presents a declarative approach and is by far the most significant query language

for data exchange. We say to queries in the relational calculus often *first-order queries*, or *FO-queries* for short. A first-order query over a schema $\mathbf{S}$ is composed of the following elements:

1. *Quantifiers*: $\forall$, $\exists$
2. *Boolean connectives*: $\wedge$, $\vee$, $\neg$
3. *Atomic formulae*: relations $R(t_1, \ldots, t_k)$ from $\mathbf{S}$ and equalities $t_1 = t_2$, where each t_i is either a variable or a constant from $\mathbf{D}$.

Assume a schema $\mathbf{S} = \{R_1, \ldots, R_n\}$ and an instance $I = \{R_1^I, \ldots, R_n^I\}$ over $\mathbf{S}$. Let the first-order structure $M_I = \langle \mathbf{D_a}, R_1^I, \ldots, R_n^I \rangle$ be the *model of* I, and let $q(x_1, \ldots, x_k)$ be a k-ary first-order query over $\mathbf{S}$, where $x_1, \ldots, x_k$ are free variables. The answer of $q(x_1, \ldots, x_k)$ over I is the set of tuples $\{\langle a_1, \ldots, a_k \rangle \mid M_I \models q(a_1, \ldots, a_k)\}$.

We say a first-order query q is *domain independent* if and only if we cannot find an instance I and two domains $\mathbf{D}_1$ and $\mathbf{D}_2$, both containing the active domain of I, such that evaluating q over I with domain $\mathbf{D}_1$ results in a different answer than evaluating q over I with domain $\mathbf{D}_2$. Codd [27] showed in 1972 the following fundamental result:

Theorem 2.3. [27] *Relational Algebra and the domain independent relational calculus have the same expressive power.*

In other words, the above result means that for every domain independent first-order query there exists an expression in relational algebra such that the result is identical over all databases and vice versa.

As next step the complexity of query answering is highlighted. Therefore, we have to transform the problem of query answering into a decision problem. A *Boolean query* is a query with no free variables such that we can either have an empty tuple or the empty set as result. We write $\mathsf{certain}(q, I) = \mathbf{true}$ if the answer of the Boolean query q over an instance I contains the empty tuple and $\mathsf{certain}(q, I) = \mathbf{false}$ if the Boolean query results in the empty set. The problem of evaluating a Boolean query over a given database is defined as follows:

PROBLEM:	QUERYEVALUATION(q)
INPUT:	a source instance I
QUESTION:	is $\mathsf{certain}(q, I) = \mathbf{true}$?

Notice that considering only Boolean queries does not lead to a loss of generality, since there is a PTIME reduction from the problem of computing the certain answers of a k-ary query to the QUERYEVALUATION(q) problem. In this reduction all of the $\mathbf{D_a}^k$ possible tuples are enumerated and tested if they are included in the certain answers.

Vardi [65] developed three complexity measures for categorizing query problems. *Combined complexity* considers the problem of query evaluation with arbitrary queries and source instances. It is the most general complexity

measure in the sense that it respects all sources of complexity. Another complexity measure is *data complexity* that considers an arbitrary query as fixed and examines only the complexity increase from the instance. Data complexity most adequately reflects the complexity of query evaluation if we assume that instances exceed queries in their size by far. Thus, in the range of this work we focus on data complexity. Clearly, QueryEvaluation(q) refers to data complexity. In addition, *query complexity* is a complexity measure where the complexity increase from the instance is considered as constant and only the complexity resulting from the query is investigated. Query complexity is from interest if the impact of queries growing in size or features is studied. It should not come as a surprise that the combined complexity is always at least as high as data complexity and query complexity.

It turns out that the complexity of QueryEvaluation(q) is for complete and incomplete data not the same. The result by Vardi [65] shows that QueryEvaluation(q) is tractable for complete instances.

Theorem 2.4. [65] *The* QueryEvaluation(q) *problem, where q is is a positive existential first order query, is for complete instances in LOGSPACE.*

Abiteboul et al. [2] showed that if the input instance contains incomplete information the complexity for QueryEvaluation(q) increases to intractability. Notice that the following result holds for each of the three discussed representation systems.

Theorem 2.5. [2] *The* QueryEvaluation(q) *problem, where q is is a positive existential first order query, is for incomplete instances coNP-complete.*

Finally, we consider two special cases of queries that have shown advantageous properties for data exchange. The first fragment are *conjunctive queries* (CQs) that have the form $\exists x_1 \ldots \exists x_n A_1 \wedge \ldots \wedge A_m$, where A_i are atomic formulae. In relational algebra conjunctive queries can be expressed with S^bPJR-expressions, where S^b is a restriction of the selection operator such that the Boolean expression ψ consists only of one atom of the form $A = a$ or $A = B$. An extended form of CQs are *union of conjunctive queries* (UCQs) that are in relational calculus a disjunction of conjunctive queries. They correspond to S^+PJRU-expressions in relational algebra.

2.2 XML Data

2.2.1 Data Model

In this section a definition of the notions of XML trees and DTDs is given. The notations introduced in this section are inspired by the notations used in [15] and the notations commonly used in XML exchange [9, 4, 3, 30, 23]. We start by assuming the following disjoint countably infinite sets:

- *Labels*, a set of names for element types,
- *Attributes*, a set of attribute names which are preceded for better recognizability by the symbol @, and

- V, a set of attribute values.

Consider two finite subsets $\Gamma \subset \mathit{Labels}$ and $\mathit{Att} \subset \mathit{Attrbiutes}$. An *XML tree* over Γ and Att is a structure $T = \langle U, \downarrow, \downarrow^*, \rightarrow, \rightarrow^*, lab, (\rho_a)_{a \in \mathit{Att}} \rangle$, where

- U is a finite set of node ids, such that $U \subset \mathbb{N}^*$ and for all $j < i$ it holds that if $n \cdot i \in U$ then also $n \cdot j \in U$;
- $\downarrow$ is the child relation, defined as $n \downarrow n \cdot i$, and $\downarrow^*$ is its reflexive-transitive closure;
- $\rightarrow$ is the next-sibling relation, given by $n \cdot i \rightarrow n \cdot (i+1)$, and its reflexive transitive closure is denoted as $\rightarrow^*$;
- lab is a labeling function from U to Γ;
- ρ_a are partial functions from U to V, such that $\rho_a(s) = v$ holds when node $s \in U$ contains value $v \in \mathcal{D}$ for the attribute $a \in \mathit{Att}$.

Note that we use the term *XML document* occasionally as synonym for the the term XML tree.

A *document type definition*, or short a *DTD*, over the finite sets $\Gamma \subset \mathit{Labels}$ and $\mathit{Att} \subset \mathit{Attributes}$ is a triple $D = \langle r, P_D, A_D \rangle$ defined as follows:

- $r \in \Gamma$ is the distinguished root symbol.
- P_D is a function from Γ to regular expressions over $\Gamma - \{r\}$ given by the grammar

  $$e := \epsilon \mid l, l \in \Gamma \mid e|e \mid ee \mid e^*,$$

  where ϵ is the empty string, $e|e$ is the union, ee is the concatenation, and e^* is the Kleene star. Moreover, we use the shorthands $e^+ = ee^*$ and $e? = e|\epsilon$. We write frequently $l \rightarrow e$ for the expression $P_D(l) = e$.
- A_D is a mapping from Γ to subsets of Att, that associates to each label a possibly empty set of attribute names. To simplify the notation, it is assumed that the attributes of a label have a specific order. Hence, a tree node with label l and n attributes is denoted as $l(a_1, \ldots, a_n)$.

An XML tree T *conforms* to a DTD $D = \langle r, P_D, A_D \rangle$, denoted as $T \models D$, if the label of the root node of T is r, for every node labeled with l the labels of its children read left-to-right form a string in the language defined by the regular expression $P_D(l)$, and the set of attributes of l is exactly $A_D(l)$.

Sometimes DTDs are of interest which are *nested relational* and represent a generalization of nested relations. In a nested relational DTD all produced regular expressions by P_D are of the form $l \rightarrow \hat{l}_1 \cdots \hat{l}_m$, where each $\hat{l}_i$ is either l_i, l_i^*, l_i^+ or $l_i?$ and each l_i is a distinct label. In addition, connecting for all $l \rightarrow \hat{l}_1 \cdots \hat{l}_m$ the label l with all labels l_i results in a graph that has no cycles.

2.2.2 Tree patterns

A natural way to query an XML tree is to check whether a specific pattern occurs in a XML document. Such a pattern is basically a recursive description of a root node and a listing of its children and descendants. We use the

extended notion of patterns as proposed in [4, 3], since they allow, compared to the patterns used in [9], the usage of the next- and following-sibling axes. A *tree pattern* formula over $\Gamma \subset Labels$ and $Att \subset Attributes$ is defined by the grammar

$$\begin{aligned} \pi &:= l(\mathbf{x})[\lambda] && \text{patterns} \\ \lambda &:= \epsilon \mid \mu \mid /\!\!/\pi \mid \lambda, \lambda && \text{sets} \\ \mu &:= \pi \mid \pi \rightarrow \mu \mid \pi \rightarrow^* \mu && \text{sequences} \end{aligned}$$

where l ranges over Γ and the *wildcard* symbol, denoted by _, which matches every label. We denote with $\pi(\mathbf{x})$ that the variables in $\mathbf{x}$ are used in the tree pattern π. In particular, every variable in $\mathbf{x}$ must occur exactly once in the tree pattern π. As it is common in the literature, we write sometimes $l(\mathbf{x})/l'(\mathbf{y})$ for $l(\mathbf{x})[l'(\mathbf{y})]$, and $l(\mathbf{x})/\!\!/l'(\mathbf{y})$ for $l(\mathbf{x})[/\!\!/l'(\mathbf{y})]$. Moreover, we abbreviate the tree pattern $l(\mathbf{x})[\epsilon]$ with $l(\mathbf{x})$.

We say a tree T *satisfies* a tree pattern $\pi(\mathbf{x})$ at node s with variables $\mathbf{x}$ interpreted as $\mathbf{a}$, written as $(T, s) \models \pi(\mathbf{a})$, if the following conditions hold:

- $(T, s) \models l(\mathbf{a})$ iff s is labeled with l and the attributes of s are $\mathbf{a}$.
- $(T, s) \models l(\mathbf{a})[\lambda_1, \lambda_2]$ iff $(T, s) \models l(\mathbf{a})[\lambda_1]$ and $(T, s) \models l(\mathbf{a})[\lambda_2]$ holds.
- $(T, s) \models l(\mathbf{a})[\mu]$ iff $(T, s) \models l(\mathbf{a})$ and s has a child s' so that $(T, s') \models \mu$.
- $(T, s) \models l(\mathbf{a})[/\!\!/\pi]$ iff $(T, s) \models l(\mathbf{a})$ and s has an descendant s' such that $(T, s') \models l(\mathbf{a})$.
- $(T, s) \models \pi \rightarrow \mu$ iff $(T, s) \models \pi$ and s has a sibling s' so that $(T, s') \models \mu$.
- $(T, s) \models \pi \rightarrow^* \mu$ iff $(T, s) \models \pi$ and s has a following sibling s' such that $(T, s') \models \mu$.

It is important to point out that if node s satisfies $l(\mathbf{a})[\lambda_1, \lambda_2]$, then λ_1 and λ_2 can be witnessed by the the same child of s. We write $T \models \pi(\mathbf{a})$ if a tree T satisfies a pattern π at its root node, i.e. if $(T, r) \models \pi(\mathbf{a})$. Moreover, with the use of the descendant operator $/\!\!/$, all patterns can be transformed into a pattern that can be witnessed at the root node of a tree. Thus, it suffices to consider only tree patterns that can be witnessed at the root node of trees. If we say a tree T satisfies a tree pattern $\pi(\mathbf{x})$ or write $T \models \pi(\mathbf{x})$, then the tree pattern is satisfied for some variable interpretation $\mathbf{a}$. For a tree pattern $\pi(\mathbf{x})$ and a tree T, we denote with $\pi(T)$ the set $\{\mathbf{a} | T \models \pi(\mathbf{x})\}$.

We discuss now basic properties of tree patterns. We start by considering the problem of checking whether a tree pattern occurs in an XML tree, or more formally:

PROBLEM:	TREEPATTERNEVALUATION(π)
INPUT:	a tree T and a tuple $\mathbf{a}$
QUESTION:	$T \models \pi(\mathbf{a})$?

Since the tree pattern π is in the above problem fixed it is referred as the data complexity version of the problem. Amano et al. [4] proved the following result:

Theorem 2.6. [4] *The* TreePatternEvaluation(π) *problem is DLOG-SPACE-complete.*

In the combined complexity version of the problem, denoted as TreePatternEvaluation, is the tree pattern considered as part of the input. Amano et al. [4] showed that also for the combined complexity variant of the problem tree patterns are well behaved and remain tractable.

Theorem 2.7. [4] *The* TreePatternEvaluation *problem is solvable in PTIME.*

We look next at a problem where its asked, whether a tree pattern can be satisfied in any tree conforming to a specific DTD. This problem has occurred several times in the literature in different manifestations.

Theorem 2.8. [5, 18, 20, 48, 4] *The problem of checking, for a DTD D and a tree pattern $\pi(\mathbf{x})$, whether there exists a tree T that conforms to D and satisfies $\pi(\mathbf{x})$ is NP-complete.*

2.2.3 Query Answering

The previous section specified tree patterns which allowed us to define basic queries. Essentially, the concept of tree patterns allowed us to determine whether a given structure is included in an XML tree. This section presents queries based on tree patterns, which allow projection and comparison of data values. A *conjunctive tree queries* is a expression of the form

$$\exists \mathbf{x}(\pi, \alpha) \,,$$

where π is a tree pattern, α is a conjunction of equalities and inequalities, and each free variable ξ_0 is safe, i.e. either ξ_0 appears in π or $\xi_0 = \xi_1, \xi_1 = \xi_2, \ldots, \xi_{k-1} = \xi_k$ are equalities in α and ξ_k appears in π. The semantics for the existence quantifier, conjunctions, equalities, inequalities, and free variables is as usual. Observe, that due to definition of tree pattern, conjunctions among patterns is not necessary. A query q is satisfied by the XML tree T, denoted as $T \models q$, if there exists a variable assignment for $\mathbf{x}$ and a valuation of the free variables, such that π and α are satisfied. The *output* of a query is the set of valuations of free variables in q, such that $T \models q$. The just defined queries form the class **CTQ**. We also consider *unions of conjunctive tree queries*. The class **UCTQ** consists of queries of the form $q_1(\mathbf{x}) \cup \ldots \cup q_n(\mathbf{x})$, where each q_i is a query from **CTQ**.

CHAPTER 3

Relational Data Exchange

The comprehensive study of the logical foundations of data exchange started in 2003 on the relational data model with the work of Fagin, Kolaitis, Miller and Popa [37]. In the following years a large number of subsequent research papers were published. As a result, relational data represents nowadays the best understood data model for data exchange. Although, surveys on relational data exchange have already appeared [50, 14], this chapter aims to give a more detailed overview of this topic. It is worth to mention that we focus on data complexity in the complexity analysis of the problems related to relational data exchange, i.e. we consider only the complexity arising from the size of the source data.

This chapter is organized as follows. In Section 3.1 the notion of schema mapping is defined, that essentially describes how source data can be transformed. The key element of schema mappings are logical formulae, called dependencies, that represent the relationship between source and target data. We consider dependencies in first-order and second-order logic and study their properties for data exchange with relational data. In Section 3.2 the problem of materializing solutions for source instances under schema mappings specified by first-order or second-order dependencies is studied. We show that there is for both kinds of formulae an algorithm that produces in polynomial time a solution if one exists. Section 3.3 considers the evaluation of queries posed over solutions. We show that query evaluation is for UCQs in general coNP-hard, but there are also special cases that allow efficient computation. In Section 3.4 we discuss the problem where we have two successive schema mappings given and we search for a third schema mapping that replaces the previous two. It turns out that for this problem second-order dependencies are more advantageous than first-order dependencies. Section 3.5 presents a generalization of the topics discussed so far in relational data exchange, where incomplete source data is allowed. The key idea is to use a more expressive representation system than the usual representation system formed by naive instances.

3.1 Data Exchange

In this section data exchange settings for relational data and its established notation are introduced. In the two subsequent sections 3.1.1 and 3.1.2, we focus on the logical formulae that relate two schemas. Moreover, a basic decision problem is discussed, where we ask if a given source instance can be transformed. In the range of this chapter data is assumed to be relational even though it is not explicitly mentioned. We start by giving a formal definition of the setting considered in data exchange.

Definition 3.1. *A* data exchange setting *is a* schema mapping*, defined as a triple* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ *that consists of a source schema* $\mathbf{S_1}$*, a target schema* $\mathbf{S_2}$*, and a set* Σ *of logical formulae called dependencies. An* instance of a schema mapping $\mathcal{M}$ *is an instance* $\langle I, J\rangle$ *over* $\langle \mathbf{S}_1, \mathbf{S}_2\rangle$ *that satisfies every dependency in* Σ*, denoted as* $\langle I, J\rangle \models \Sigma$*. This* J *is called a* solution for I under $\mathcal{M}$*, or for short a* solution*. The set of all solutions for* I *under* $\mathcal{M}$ *is denoted with* $\mathsf{Sol}_{\mathcal{M}}(I)$*.* $\mathsf{Inst}(\mathcal{M})$ *denotes the set of all instances* $\langle I, J\rangle$ *of a schema mapping* $\mathcal{M}$*.*

In the range of this work, we use the terms "data exchange setting" and "schema mapping" interchangeably. Moreover, we call instances conforming to the source schema $\mathbf{S}_1$ *source instances*. Analogously, we say to instances conforming to the target schema $\mathbf{S}_2$ *target instances*. As it is common in the literature, source instances are assumed to be complete, i.e. the domain of $\mathbf{S}_1$ contains only values from Const. In contrast, target instances can have values from $\mathsf{Const} \cup \mathsf{Var}$.

For the set of dependencies Σ we have to find a proper logical formalism. We would like to have the property that if instance $\langle I, J\rangle \models \Sigma$, and a second instance $\langle I', J'\rangle$ is isomorphic to $\langle I, J\rangle$, we also have that $\langle I', J'\rangle \models \Sigma$. More formally, the set $\mathsf{Inst}(\mathcal{M})$ should be closed under isomorphisms. Furthermore we would like to have a logical formalism with high expressivity and at the same time low complexity bounds for all relevant problems. Obvious candidates for such a formalism are first-order logic and second-order logic. Both candidates are preserved under isomorphisms. We will investigate first-order dependencies and second-order dependencies in Section 3.1.1 and 3.1.2, respectively.

We will see that the considered formalisms are only able to enforce which tuples have to be contained in a solution J. They are not able to restrict that no further tuples can be contained in a valid solution. Thus, schema mappings have for a source instance in general infinitely many solutions. This is due the fact that we can take a solution and receive another solution by adding further tuples. Nevertheless, there are also cases where no solutions exists. This observation leads to the problem where we ask if for a schema mapping and a given source instance a solution exists. More formally:

PROBLEM:	SOLUTIONEXISTENCE($\mathcal{M}$)
INPUT:	a source instance I
QUESTION:	is $\mathsf{Sol}_{\mathcal{M}}(I) \neq \emptyset$?

SolutionExistence($\mathcal{M}$) can be seen as the most basic data exchange problem considered in this context, since every other investigated problem needs to solve this problem directly or indirectly. Moreover, the logical formalism used for specifying the set of dependencies Σ must deliver reasonable complexity bounds for SolutionExistence($\mathcal{M}$).

3.1.1 First-Order Dependencies

In this section we investigate first-order logic to express dependencies of data exchange settings. We show that allowing arbitrary first-order formulas as dependencies makes the problem of deciding if a source instance has a solution under a schema mapping undecidable. Moreover, a restriction of first-order dependencies is introduced which is tractable and still allows to express interesting constraints. The following theorem follows directly from Proposition 3.9 in [39]:

Theorem 3.2. [39] *There exist a source instance I for a data exchange setting $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$, where Σ is a single first-order formula, such that* SolutionExistence($\mathcal{M}$) *is undecidable.*

Proof. *(Sketch)* We define $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ such that $\langle I, J\rangle \in \mathsf{Inst}(\mathcal{M})$, if I is an encoding of a Turing machine and J encodes a terminating computation. Σ consists of a first order expression such that $J \in \mathsf{Sol}_{\mathcal{M}}(I)$ if and only if J represents a terminating computation of I. If I is the encoding from a Turing machine that computes the halting problem, then it is undecidable to determine if $\mathsf{Sol}_{\mathcal{M}}(I) \neq \emptyset$. □

This result demands that we either search for a more suitable logical formalism or for a fragment of first-order which is decidable. In [37] Fagin et al. do the latter and restrict the class of allowed first-order dependencies in such a way that the set Σ can be split in two sets Σ_{12} and Σ_2:

Definition 3.3. *(1)* Σ_{12} *is a set of* source-to-target tuple-generating dependecies (st-tgds) *of the form* $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$*, where φ is a conjunction of atomic formulas over $\mathbf{S}_1$ and ψ is a conjunction of atomic formulas over $\mathbf{S}_2$.*

(2) Σ_2 *is a set of* target dependencies *consisting of* tuple-generating dependencies (tgds) *of the form* $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$*, where φ and ψ are conjunctions of atomic formulas over $\mathbf{S}_2$, and* equality-generating dependencies (egds) *of the form* $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow x_i = x_j)$*, where φ is a conjunction of atomic formulas over $\mathbf{S}_2$ and x_i, x_j are variables of $\mathbf{x}$.*

Observe, that in the above defined dependencies, it is required that every universally quantified variable appears at least once in the antecedent of a dependency. This requirement ensures that the solutions are domain independent, and was introduced in [35]. A data exchange setting using these dependencies is denoted with $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$. It is worth mentioning that in the data integration context [52] source-to-target tgds are known as Glav assertions. We will call a tgd or st-tgd *full* if it has no existentially

quantified variables, i.e. if it has the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$. Moreover, for better readability we omit in examples universal quantifiers. For instance, we write $E(x, y, z) \rightarrow \exists w M(x, y, w)$ instead of writing $\forall x \forall y \forall z(E(x, y, z) \rightarrow \exists w M(x, y, w))$. From now on if we speak about first-order dependencies in a relational data exchange setting we mean explicitly the dependencies defined in Definition 3.3.

Example 3.4. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a schema mapping in which $\mathbf{S}_1$ consists of two binary relations E, F and $\mathbf{S}_2$ consists of a ternary relation M and a binary relation N. The dependencies are defined as follows:

$$\begin{aligned}\Sigma_{12} = \{&E(x, y) \wedge E(y, z) \rightarrow \exists w M(x, z, w),\\ &F(x, y) \rightarrow N(x, y)\}\\ \Sigma_2 = \{&M(x, y, z) \wedge M(w, y, v) \rightarrow \exists u N(y, u),\\ &M(x, y, z) \wedge M(x, w, v) \rightarrow y = w\}\end{aligned}$$

Notice that the second dependency in Σ_{12} is a full source-to-target tgd. We now ask if for the source instance $I = \{E(a, c), E(b, c), E(c, d), F(a, b)\}$ a solution under $\mathcal{M}$ exists. A solution for I would be

$$J_1 = \{M(a, d, n_1), M(b, d, n_2), N(a, b), N(d, n_3)\},$$

where n_1, n_2 and n_3 are elements from Var. Clearly, an algorithm solving the SolutionExistence($\mathcal{M}$) problem must only witness the existence of a solution. As mentioned earlier, a schema mapping can have more than one solution. In fact, $\mathsf{Sol}_{\mathcal{M}}(I)$ has infinitely many solutions. Here are some of them:

$$\begin{aligned}J_2 =&\{M(a, d, b), M(b, d, c), N(a, b), N(d, a)\},\\ J_3 =&\{M(a, d, n_1), M(b, d, n_2), N(a, b), N(d, n_3), N(c, d)\},\\ J_4 =&\{M(a, d, n_1), M(b, d, n_2), M(a, d, d), N(a, b), N(d, n_3)\},\end{aligned}$$

where n_1, n_2 and n_3 are elements from Var.

Assume now the source instance $I' = \{E(a, b), E(b, c), E(b, d), F(a, b)\}$. A solution has to contain the facts $M(a, c, n_1)$ and $M(a, d, n_2)$, where n_1 and n_2 are values from $\mathsf{Const} \cup \mathsf{Var}$. It is easy to see, that these facts contradict with the egd in Σ_2. We conclude that I' has no solution. □

We introduced these new dependencies with the objective to find a logical formalism such that the SolutionExistence($\mathcal{M}$) problem becomes decidable and possibly also tractable. Nevertheless, [51] showed the following result:

Theorem 3.5. [51] *There exist a data exchange setting* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ *and a source instance* I *such that* SolutionExistence($\mathcal{M}$) *is undecidable.*

Proof. The undecidability of SolutionExistence($\mathcal{M}$) for $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ can be obtained by a reduction from Embedding. Let $\mathbf{B} = (B, g)$

be a partial algebra, where B is a finite non-empty set, called domain, and g is a partial binary function from $B \times B \to B$. Let $\mathbf{A} = (A, f)$ be an algebra, where A is the domain, and f is a total binary function $A \times A \to A$. We say that $\mathbf{B}$ is embeddable in $\mathbf{A}$ if $B \subseteq A$, and whenever we have that $g(a, b)$ is defined, $a, b \in B$, we have that $f(a, b) = g(a, b)$. In the decision problem EMBEDDING we are given a finite partial semigroup and we ask if its embeddable in a finite semigroup. The undecidability of EMBEDDING follows immediately from [46, 34].

We are given a partial algebra $\mathbf{B} = (B, g)$ defined as above. We define now a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ and a source instance I such that I has a solution under $\mathcal{M}$ if and only if $\mathbf{B}$ is embeddable in a finite semigroup. Schema $\mathbf{S}_1$ and schema $\mathbf{S}_2$ consist of one ternary relation R and R', respectively. Instance I is composed of the facts $\{(a, b, c) \in B^3 \mid g(a, b) = c\}$. The dependencies are defined as follows:

$$\begin{aligned}
\Sigma_{12} &= \{R(x, y, z) \to R'(x, y, z)\} \\
\Sigma_2 &= \{R'(x, y, z) \wedge R'(x, y, w) \to z = w, \\
&\quad R'(x, y, u) \wedge R'(y, z, v) \wedge R'(u, z, w) \to R'(x, v, w), \\
&\quad R'(x, y, z) \wedge R'(x', y', z') \to \exists w_1 \cdots \exists w_9 \\
&\qquad (R'(x, x', w_1) \wedge R'(x, y', w_2) \wedge R'(x, z', w_3) \wedge \\
&\qquad R'(y, x', w_4) \wedge R'(y, y', w_5) \wedge R'(y, z', w_6) \wedge \\
&\qquad R'(z, x', w_7) \wedge R'(z, y', w_8) \wedge R'(z, z', w_9))\}
\end{aligned}$$

The source-to-target tgd copies the tuples from relation R to relation R'. We have three target-dependencies, one egd and two tgds. The egd assure that the facts in R' actually encode a partial function. The second target dependency guarantees that the encoded partial function is associative. Clearly, an instance J satisfying the first two target dependencies is an encoding of a finite partial semigroup. The last tgd demands that for every two elements u and v there must be an element w, such that there exists a fact $R'(u, v, w)$ in a solution for I. In other words, the last target dependency asserts that the encoded partial semigroup J is actually a total semigroup. □

We are now interested in finding a further restriction for the dependencies from Definition 3.3 such that the SOLUTIONEXISTENCE($\mathcal{M}$) problem becomes decidable. Therefore, we have to understand what causes the undecidability.

Example 3.6. Consider the data exchange setting $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$. Schema $\mathbf{S}_1$ and schema $\mathbf{S}_2$ consist of one ternary relation R and R', respectively. Σ_{12} and Σ_2 are defined as follows:

$$\begin{aligned}
\Sigma_{12} &= \{R(x, y, z) \to R'(x, y, z)\} \\
\Sigma_2 &= \{R'(x, y, z) \to \exists w R'(y, w, x)\}
\end{aligned}$$

We will now examine what facts a solution J would have to contain for source instance $I = \{R(a, b, c)\}$. We start with $J' = \emptyset$ and add to it successively tuples to satisfy the dependencies. After satisfying Σ_{12} we have

$$J' = \{R'(a, b, c)\}.$$

The dependency in Σ_2 is now violated and we have to add the tuple $R'(b, n_1, a)$, where n_1 is an element of Var:

$$J' = \{R'(a, b, c), R'(b, n_1, a)\}.$$

The existence of $R'(b, n_1, a)$ forces the tgd in Σ_2 to fire again:

$$J' = \{R'(a, b, c), R'(b, n_1, a), R(n_1, n_2, b)\},$$

where n_1 and n_2 are elements of Var. The dependency $R'(x, y, z) \rightarrow R'(y, w, x)$ is still violated. It is easy to see that the dependency remains violated even if we keep adding tuples. The same behavior can be observed with the schema mapping from the proof of Theorem 3.5 and source instance $\{R(a, b, c), R(d, e, f)\}$. □

Intuitively, the undecidability is caused from tgds in Σ_2 that generate tuples which force the tgd to fire again. This behavior can be triggered from a single tgd or a series of tgds. Deutsch and Popa developed the following approach to detect such cases in 2001 and utilized it independently in [33] and [37].

Definition 3.7. *A* dependency graph *for a set Σ_2 of tgds over schema $\mathbf{S}_2$ is a directed graph defined as follows:*

1. *For every attribute A occurring in a relation symbol R in $\mathbf{S}_2$ there is a distinct node for the pair (R, A), called* position*.*
2. *For every tgd $\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x}, \mathbf{y})$ in Σ_2 and every occurrence of $x \in \mathbf{x}$ in φ in position (R, A) that also occurs in ψ:*
   a) *there exists an edge $(R, A) \rightarrow (S, B)$ for every position (S, B) where x occurs in ψ.*
   b) *there exists a* special edge *$(R, A) \overset{*}{\rightarrow} (T, C)$ for every existentially quantified variable $y \in \mathbf{y}$ that occurs in position (T, C).*

We say that Σ_2 is weakly acyclic *if the dependency graph has no cycle that contains a special edge.*

Example 3.8. Let $\mathbf{S}_2$ be a schema consisting of one ternary relation R. We identify an attribute of an m-ary relation symbol with the corresponding natural number between 1 and m. The set of tgds $\{R(x, y, z) \rightarrow \exists w R(y, w, x)\}$ is not weakly acyclic, because the special edge from position $(R, 1)$ to $(R, 2)$ and the edge from position $(R, 2)$ to $(R, 1)$ form a cycle. In contrast $\{R(x, y, z) \rightarrow \exists w R(y, x, w)\}$ is weakly acyclic. Figure 3.1 shows their dependency graphs. Figure 3.2 illustrates the dependency graph of the target dependencies from the schema mapping used in the proof for Theorem 3.5. □

We have now all sufficient restrictions for first-order dependencies in schema mappings such that the SolutionExistence($\mathcal{M}$) problem becomes decidable. Indeed, [37] showed that the SolutionExistence($\mathcal{M}$) problem, where the set of target tgds in $\mathcal{M}$ is weakly acyclic, can be solved in polynomial time. In Section 3.2.1 we will discuss an algorithm which is able to materialize a solution in polynomial time or fails if none exists. Since this algorithm

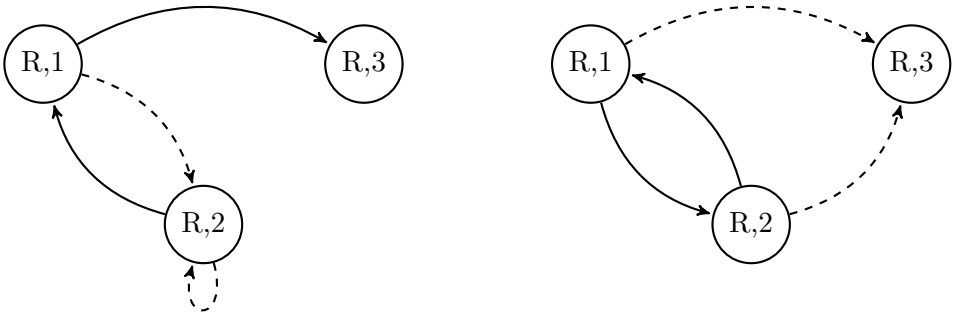


Figure 3.1: The dependency graphs of $\{R(x, y, z) \rightarrow \exists w R(y, w, x)\}$ and $\{R(x, y, z) \rightarrow \exists w R(y, x, w)\}$. Dashed arrows represent special edges.

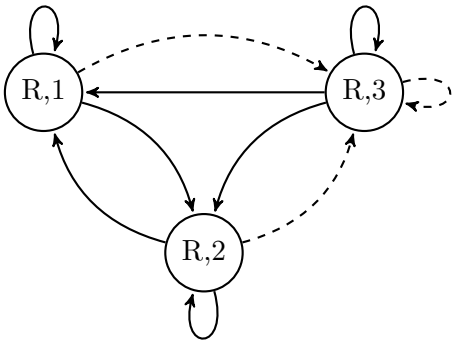


Figure 3.2: Dependency graph for Σ_2 from the schema mapping in the proof for Theorem 3.5. Dashed arrows represent special edges.

solves the SolutionExistence($\mathcal{M}$) problem implicitly, it can be seen as proof of PTIME-membership. Furthermore, [51] showed that there are instances and schema mappings for which SolutionExistence($\mathcal{M}$) is for the complexity class PTIME hard. We conclude:

Theorem 3.9. [51] SolutionExistence($\mathcal{M}$) *is for data exchange settings with weakly acyclic target tgds PTIME-complete.*

Proof. *(Sketch)* PTIME hardness of SolutionExistence($\mathcal{M}$) can be shown by a reduction from Horn3Sat. In Horn3Sat we are given a Boolean formula φ in conjunctive normal form with at most one positive literal and at most three literals per clause. The question is if φ is satisfiable. The PTIME-completeness of Horn3Sat is shown in [59].

The source schema $\mathbf{S}_1$ consists of two ternary and one unary relational symbol P, N and V, respectively. From an arbitrary given Boolean Horn formula φ with at most three literals per clause we construct an instance I of $\mathbf{S}_1$ as follows: for each clause of the form $(x \vee \neg y \vee \neg z)$ we add a fact $P(x, y, z)$; for each clause of the form $(\neg x \vee \neg y \vee \neg z)$ we add a fact $N(x, y, z)$; clauses consisting of exactly one positive literal are encoded with $P(x, x, x)$. Moreover, we add the tuples $V(a)$ and $V(b)$. Clearly, the construction of I is

feasible in logarithmic space. The target schema $\mathbf{S}_2$ consists of two ternary and three unary relation symbols P', N', M', V' and W', respectively. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a schema mapping with the following dependencies:

$$\begin{aligned}
\Sigma_{12} = \{&P(x,y,z) \rightarrow P'(x,y,z),\\
&N(x,y,z) \rightarrow N'(x,y,z),\\
&V(x) \rightarrow V'(x)\}\\
\Sigma_2 = \{&W'(u) \wedge W'(v) \rightarrow u = v,\\
&P'(x,x,x) \rightarrow M'(x),\\
&P'(x,y,z) \wedge M'(y) \wedge M'(z) \rightarrow M'(x),\\
&N'(x,y,z) \wedge M'(x) \wedge M'(y) \wedge M'(z) \wedge V'(u) \rightarrow W'(u)\}
\end{aligned}$$

A solution J for I under $\mathcal{M}$ contains in relation M at least those variables that are directly or indirectly forced to be evaluated to true in φ. Each variable for which no fact in M exists is in an evaluation of φ false. It is easy to verify that J indeed encodes a satisfying variable evaluation for φ. If φ is unsatisfiable then there is a clause $(\neg x \vee \neg y \vee \neg z)$ and x, y and z are forced to be true. In this case the last dependency in Σ_2 fires twice and generates the facts $W'(a)$ and $W'(b)$ that leads to a contradiction in the egd. □

Its worth to mention that SolutionExistence($\mathcal{M}$) problem is not only tractable for the class of schema mappings where the dependencies are assembled from st-tgds, egds and weakly acyclic target tgds. In [31] and [57] even broader classes of schema mappings are shown, such that the SolutionExistence($\mathcal{M}$) remains tractable. These less restrictive classes have the disadvantage that deciding if dependencies are contained in these classes is in coNP.

Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ where Σ_{12} is a set of source-to-target tgds and Σ_2 consists of a set of weakly acyclic tgds but has no egds. In this case the SolutionExistence($\mathcal{M}$) problem is trivial, since every source instance has a solution. It is common in the literature to consider only schema mappings with no target tgds. We denote such data exchange settings with $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$.

3.1.2 Second-Order Dependencies

In this section we consider second-order logic as the formalism for the set of dependencies in schema mappings. Since second-order logic is an extension of first-order logic the undecidability result from Theorem 3.2 applies also to unrestricted second-order formulas. We present in this section a restricted form of existential second-order formulas, as proposed in [39], which not only allow to materialize solutions in polynomial time, but also have favorable properties when we consider composition of schema mappings in Section 3.4. Furthermore, it is possible to transform every set of first-oder st-tgds into an equivalent second order formula conforming to the presented restrictions.

Before we can define the restrictions for second order formulas we have to introduce the notion of terms:

Definition 3.10. *Let* $\mathbf{x}$ *be a collection of variables, and* $\mathbf{f}$ *be a collection of function symbols. A* term based on $\mathbf{x}$ and $\mathbf{f}$*, or* term *for short, is:*

1. *every variable in* $\mathbf{x}$*, and*
2. *every* k*-ary function* $f(t_1, \ldots, t_k)$*, where* f *is function symbol in* $\mathbf{f}$ *and* $t_1, \ldots t_k$ *are terms.*

The restricted form of existential second-order formulas presented in [39] is defined as follows:

Definition 3.11. *Consider a source schema* $\mathbf{S}_1$ *and a target schema* $\mathbf{S}_2$*. A* second-order tuple-generating *dependency (SO-tgd) is of the form*

$$\exists \mathbf{f}((\forall \mathbf{x_1}(\varphi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x_n}(\varphi_n \rightarrow \psi_n))), \text{ s.t.}$$

1. $\mathbf{f}$ *is a collection of function symbols.*
2. *Each* φ_i *is a conjunction of atomic formulas* $S(x_1, \ldots, x_k)$*, where* S *is a* k*-ary relation symbol in* $\mathbf{S}_1$*,* $x_1, \ldots, x_k \in \mathbf{x_i}$*, and of equalities* $t = t'$ *of terms based on* $\mathbf{x_i}$ *and* $\mathbf{f}$*.*
3. *Each* ψ_i *is a conjunction of atomic formulas* $T(t_1, \ldots, t_k)$*, where* T *is a* k*-ary relation symbol in* $\mathbf{S}_2$*, and* $t_1, \ldots, t_k$ *are terms based on* $\mathbf{x_i}$ *and* $\mathbf{f}$*.*
4. *Each* $x \in \mathbf{x_i}$ *is a* safe *term with respect to* φ_i *and* $\mathbf{f}$*, i.e., each* x *satisfies one of the following recursive properties:*
   a) x *occurs in an atomic formula of* φ_i*.*
   b) x *occurs in an equality* $x = x'$ *or* $x' = x$*, where* x' *appears in an atomic formula of* φ_i*.*
   c) x *occurs in an equality* $x = f(x'_1, \ldots, x'_k)$ *or* $f(x'_1, \ldots, x'_k) = x$*, where* $f \in \mathbf{f}$ *and* $x'_1, \ldots, x'_k$ *appears in an atomic formula of* φ_i*.*

To prevent confusion with first-order dependencies, we will call the dependencies defined in Definition 3.11 always second-order tgds, second-order dependencies, or SO-tgds. If we speak instead about tgds we mean first-order tgds from Defintion 3.3. The fourth condition in Definition 3.11 ensures domain independence. Domain independence is the analog to the restriction of first-order tgds, which states that each universally quantified variable has to appear on the left-hand side of the implication. Besides, second-order tgds are closed under conjunction. Therefore, we restrict ourselves without loss of generality to data exchange settings with only one SO-tgd.

In the following example we show sample formulas and argue if all of their universally quantified variables are safe.

Example 3.12. Let $\mathbf{S}_1$ be a schema consisting of one binary relation symbol S, and let $\mathbf{S}_2$ be a schema consisting of one ternary relation symbol T.

The following formula is not a valid SO-tgd:

$$\exists f \forall x \forall y \forall z (S(x, y) \rightarrow T(x, y, f(z)))$$

According to rule *4.a)* the variables x and y are safe; z instead is unsafe, because it does not appear on the left side of the implication. The following formula is a slightly more complex example for an invalid SO-tgd:

$$\exists f \forall x \forall y \forall z \forall w (S(x, y) \wedge (z = w) \wedge (f(w) = x) \rightarrow T(x, y, z))$$

According to rule *4.a)* the variables x and y are safe; z and w are unsafe. In contrast, the following SO-tgd is valid:

$$\exists f \forall x \forall y \forall z \forall w (S(x, y) \wedge (x = z) \wedge (f(z) = w) \rightarrow T(x, y, f(z)))$$

Variables x and y are safe, because of rule *4.a)*; z is safe because of rule *4.b)*. Rule *4.c)* makes w safe. □

As next step, we have to define the semantics of existentially quantified function symbols. If we evaluate an SO-tgd we have to replace the existentially quantified function symbols with concrete functions. Furthermore, these functions must have an appropriate domain and range, such that the expected behavior can be expressed.

Definition 3.13. *Consider a combined instance $\langle I, J \rangle$ over $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$ and an SO-tgd σ of the form $\exists \mathbf{f} \sigma'$, where σ' is in first order. Let U be a* universe *consisting of* Var *and the active domain of $\langle I, J \rangle$. The structure $\langle U; I, J \rangle$ denotes the combined instance $\langle I, J \rangle$ that has universe U. The SO-tgd σ is* satisfied *by $\langle U; I, J \rangle$ if and only if there is a set of functions $\mathbf{f^0}$ with domain and range U such that σ' is satisfied by $\langle U; I, J \rangle$ when each function symbol in $\mathbf{f}$ is replaced by a function from $\mathbf{f^0}$. This behavior is denoted with $\langle U; I, J \rangle \models \sigma'[\mathbf{f} \rightarrow \mathbf{f^0}]$, or with $\langle I, J \rangle \models \sigma'[\mathbf{f} \rightarrow \mathbf{f^0}]$ if U is clear from the context.*

We give now an example that demonstrates a data exchange scenario defined by an SO-tgd and discusses possible solutions for a source instance.

Example 3.14. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a data exchange scenario, where $\mathbf{S}_1$ consists of two binary relation symbols P and E, and schema $\mathbf{S}_2$ consists of one ternary relation C. Σ_{12} consists of a single SO-tgd and is defined as follows:

$$\exists h \forall x \forall y \forall u \forall v (P(x, y) \wedge E(u, v) \wedge (h(x) = h(u)) \rightarrow C(h(x), y, v))$$

This schema mapping can be interpreted such that we have names and telephone numbers in P, and names and email addresses in E. If the names from P and E refer to the same person we can derive a new contact consisting of the name, telephone number and email address.

Consider the source instance below. For better readability telephone numbers and email addresses are represented by placeholders. Nonetheless, a, b,

c, d, e and f are values from Const.

$$I = \{P(\text{"Jim M."}, a), P(\text{"Robert P."}, b), P(\text{"Jimi Hendrix"}, c),$$
$$E(\text{"J. Morrison"}, d), E(\text{"Pete Townshend"}, e), E(\text{"Jimi Hendrix"}, f)\}$$

A solution for I under $\mathcal{M}$ is $J = \{C(n, c, f)\}$, where $n \in \mathsf{Var}$. We treat here the equality $h(x) = h(u)$ exactly then as satisfied if the values of x and u are syntactically identical.

Another interesting solution for I under $\mathcal{M}$ is $J' = \{C(\text{"Jim Morrison"}, a, d), C(\text{"Jimi Hendrix"}, c, f)\}$. Here $h(\cdot)$ gets interpreted and returns the full name of a person, i.e., $h(\text{"Jim M."})$ and $h(\text{"J. Morrison"})$ is interpreted as "Jim Morrison". □

We are now interested in comparing the second-order dependencies with the first-order dependencies presented in the previous section. We say two schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are *equivalent*, if both have the same source and target schema, and for every source instance I the sets $\mathsf{Sol}_{\mathcal{M}}(I)$ and $\mathsf{Sol}_{\mathcal{M}'}(I)$ contain exactly the same elements. Clearly, SO-tgds cannot simulate target-egds. It is easy to see that a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$, where Σ_{12} is a set of first-order source-to-target tgds and Σ_2 is a set of weakly acyclic first-order target tgds, can be rewritten in a schema mapping $\mathcal{M}' = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ defined by a set of first-order source-to-target tgds of the form

$$\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists y_1, \ldots, \exists y_n \psi(\mathbf{x}, y_1, \ldots, y_n)),$$

where φ is a conjunction of atomic formulas over $\mathbf{S}_1$ and ψ is a conjunction of atomic formulas over $\mathbf{S}_2$. We can receive a third schema mapping $\mathcal{M}''$ that is equivalent to $\mathcal{M}'$ and $\mathcal{M}$, where every first-order st-tgd is replaced with an SO-tgd of the form

$$\exists \mathbf{f} \forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}, f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))).$$

This translation from schema mappings with first order tgds to schema mappings with second order tgds is called *Skolemization*. In the example below we show how a schema mapping with two st-tgds can be transformed into a schema mapping with a single SO-tgd.

Example 3.15. Let $\mathbf{S}_1$ and $\mathbf{S}_2$ be source schema and target schema, respectively. Schema $\mathbf{S}_1$ consists of three binary relation symbols D, E and F. Schema $\mathbf{S}_2$ consists of one ternary relation M. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping, where Σ_{12} is the following set of first-order st-tgds:

$$D(x, y) \wedge E(y, z) \rightarrow \exists w M(x, z, w)$$
$$F(x, y) \rightarrow \exists v \exists w M(x, v, w)$$

We construct $\mathcal{M}' = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a single second-order tgd defined as follows:

$$\exists f \exists h((D(x, y) \wedge E(y, z) \rightarrow M(x, z, f(x, z))) \wedge$$
$$(F(x, y) \rightarrow M(x, h(x), f(x))))$$

The schema mappings $\mathcal{M}$ and $\mathcal{M}'$ are equivalent. □

We now consider the complexity of the SolutionExistence($\mathcal{M}$) problem, where $\mathcal{M}$ is a schema mapping with a single second-order tgd. If we recall the definition of SO-tgds, we see that on the target schema are only conjunctions of atomic formulas allowed. Not allowed are any kind of formulas that restrict the existence of facts in a solution. It follows that every data exchange setting $\mathcal{M}$ defined with SO-tgds has for each source instance arbitrarily many solutions. We conclude that the SolutionExistence($\mathcal{M}$) problem is trivial for data exchange settings composed of SO-tgds.

3.2 Solution Building

In the previous section the question if for an instance of a schema mapping a solution exists is examined. In this section we go a step further and aim to materialize a solution if one exists. More formally:

Problem:	DataExchange($\mathcal{M}$)
Input:	a source instance I
Goal:	if $\mathsf{Sol}_{\mathcal{M}}(I) \neq \emptyset$, find J s.t. $J \in \mathsf{Sol}_{\mathcal{M}}(I)$

We first introduce a subclass of solutions that have advantageous properties for materialization. Then an algorithm for materializing solutions is given for first-order and second-order dependencies.

In Section 3.1, we have already seen that in general a source instance can have infinitely many solutions under a schema mapping $\mathcal{M}$. A natural question is now if there are some solutions that are preferable over others. Intuitively, the materialized solution should represent the set of all solutions as well as possible. Solutions can contain facts that are not enforced by dependencies. A solution with such superfluous tuples represents neither all solutions adequately, nor is it economical to materialize it, since a solution can have arbitrarily many superfluous tuples. Moreover, for unknown values of attributes in a solution elements from $\mathsf{Const} \cup \mathsf{Var}$ can be chosen freely. A solution is clearly less representative for all solutions if it contains concrete values from Const instead of values from Var for unknown attributes. We aim to define a subclass of solutions that has neither unnecessary facts nor unjustified variable instantiations. As first step we have to fix the notion of homomorphism between two instances:

Definition 3.16. *Let I and I' be two instances over a schema $\mathbf{S}$ with values from $\mathsf{Const} \cup \mathsf{Var}$. A* homomorphism *$h : I \to I'$ is a mapping from the values used in I to the values used in I' where*

1. *$h(c) = c$ for every $c \in \mathsf{Const}$, and*
2. *for every tuple $t = (a_1, \ldots, a_n)$ that is a fact in I it holds that the tuple $t' = (h(a_1), \ldots, h(a_n))$ is a fact in I'.*

Based on homomorphisms we can formalize the above considerations:

Definition 3.17. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a schema mapping and I a source instance. A solution J is a* universal solution *for I iff for every solution J' for I there is a homomorphism $h : J \to J'$.*

It can be shown that if there is a solution for a schema mapping and a source instance, then there is also a universal solution. In the following example we review the solutions from Example 3.4 and Example 3.14.

Example 3.18. In Example 3.4 a schema mapping, defined by first order dependencies, with the corresponding source instance $I = \{E(a, c), E(b, c), E(c, d), F(a, b)\}$ is examined. The following solutions are listed:

$$\begin{aligned}
J_1 =& \{M(a, d, n_1), M(b, d, n_2), N(a, b), N(d, n_3)\} \\
J_2 =& \{M(a, d, b), M(b, d, c), N(a, b), N(d, a)\} \\
J_3 =& \{M(a, d, n_1), M(b, d, n_2), N(a, b), N(d, n_3), N(c, d)\} \\
J_4 =& \{M(a, d, n_1), M(b, d, n_2), M(a, d, d), N(a, b), N(d, n_3)\}
\end{aligned}$$

Variables n_1, n_2 and n_3 are elements from Var. We will now examine these solutions for the above described properties. All tuples in J_1 are demanded from the dependencies. For each attribute with unknown value a distinct element from Var is chosen. Solution J_2 also contains no superfluous tuples, but contains unjustified variable instantiations. Solution J_3 contains the same tuples as J_1 but has an additional fact that is not requested by the dependencies. The fourth solution J_4 has superfluous tuples and superfluous variable assignments. Solution J_1 contains exactly the tuples demanded from the dependencies and is therefore the most general of all four solutions. It is easy to see that there is a homomorphism from J_1 to J_2, J_3 and J_4. On the other hand there is no homomorphism from J_2, J_3 and J_4 to J_1. In addition, it can be shown that J_1 is a universal solution.

In Example 3.14 the following solutions for a schema mapping with second-order tgds and a source instance are shown:

$$\begin{aligned}
J &= \{C(n, c, f)\} \\
J' &= \{C(\text{"Jim Morrison"}, a, d), C(\text{"Jimi Hendrix"}, c, f)\}
\end{aligned}$$

Notice variable n is the only element from Var. We already argued that both solutions J and J' are in a sense natural. If we consider both solutions under the notion of homomorphisms we see that there is a homomorphism from J to J', but not the other way around. Furthermore, it can be shown that J is universal as well. □

It is important to notice that for a given source instance several universal solutions exist, which can also differ in their size. In this section we restrict ourselves to materialize an arbitrary universal solution. However, it would be more economical to compute the smallest universal solution. Fagin et al. [38] showed for first-order dependencies that the smallest universal solution always coincides with the core of an arbitrary universal solution. Moreover, there is an algorithm that computes the core of the universal solutions in polynomial time [44, 45].

In Section 3.2.1 and 3.2.2 algorithms for materializing solutions for data exchange settings specified by first-order and second-order dependencies are presented. Both algorithms are variations of the classical chase procedure [17]. The chase was initially developed to test whether a set of dependencies logically implies a given dependency.

3.2.1 First-Order Dependencies

In this section a modification of the classical chase procedure is shown that computes a universal solution for a source instance under a schema mapping in polynomial time or fails if no solution exists. This variant of the chase was introduced in [37]. Basically, the algorithm tries iteratively to satisfy all dependencies by adding new tuples or modifying existing ones. Before we can give an exact definition of the chase we have to introduce some auxiliary notions:

Definition 3.19. *Let $\varphi(\mathbf{x})$ be a conjunction of atomic formulas over a schema $\mathbf{S}$, and let I be an instance over $\mathbf{S}$. A* homomorphism *$h : \varphi(\mathbf{x}) \to I$ is a mapping from the variables in $\mathbf{x}$ to the the values used in I, such that for every $R(x_1, \ldots, x_n)$ in φ there is a fact $R(h(x_1), \ldots, h(x_n))$ in I.*

Definition 3.20. *Let $\langle I, J\rangle$ be an instance. A dependency d is* unsatisfied *if*

1. *d is a tgd of the form $\varphi(\mathbf{x}) \to \exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$ and there exists a homomorphism h from $\varphi(\mathbf{x})$ to $\langle I, J\rangle$, but there is no extension h' of h, such that h' is a homomorphism from $\varphi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y})$ to $\langle I, J\rangle$.*
2. *d is an egd of the form $\varphi(\mathbf{x}) \to (x_1 = x_2)$, and there is a homomorphism h from $\varphi(\mathbf{x})$ to $\langle I, J\rangle$, such that $h(x_1) \neq h(x_2)$ holds.*

Otherwise, dependency d is satisfied.

The specification of the chase is divided into a definition for chase steps and a definition of the chase itself, which is a sequence of chase steps.

Definition 3.21. (Chase step). *Consider an instance $\langle I, J\rangle$, and Σ a set of tgds and egds, which contain at least one unsatisfied dependency d. Let h be a homomorphism from the antecedent $\varphi(\mathbf{x})$ of d to $\langle I, J\rangle$. Dependent on d a* chase step *fails or produces an instance $\langle I, J'\rangle$ as follows:*

1. *If d is an unsatisfied tgd of the form $\varphi(\mathbf{x}) \to \exists\mathbf{y}\psi(\mathbf{x}, \mathbf{y})$, then $h' : \varphi(\mathbf{x}) \wedge \psi(\mathbf{x}, \mathbf{y}) \to \langle I, J\rangle$ is an extension to h, where every variable in $\mathbf{y}$ is mapped to an unused element from Var. J' is the union of J and the facts obtained by taking the image of the atoms of ψ under h'.*
2. *If d is an unsatisfied egd of the form $\varphi(\mathbf{x}) \to (x_1 = x_2)$ and at least one of $h(x_1)$ and $h(x_2)$ is assigned to an element of Var, say $h(x_1)$, then J' is a copy of J, where every occurrence of $h(x_1)$ in J is replaced by $h(x_2)$.*
3. *If d is an unsatisfied egd of the form $\varphi(\mathbf{x}) \to (x_1 = x_2)$, where both $h(x_1)$ and $h(x_2)$ are mapped on elements of Const, then the chase step fails.*

Definition 3.22. (Chase). *Let $\langle I, J\rangle$ be an instance and Σ a set of tgds and egds. A* chase sequence of $\langle I, J\rangle$ with Σ *is a possibly infinite sequence of instances $\langle I, J\rangle, \langle I, J'\rangle, \langle I, J''\rangle, \ldots$ where each successive instance is obtained by a chase step on the predecessor. A* chase of $\langle I, J\rangle$ with Σ *is called finite, if the number of chase steps is bounded by a constant. This is exactly then the case, when either the last chase step fails or after the last chase step no dependency in Σ is unsatisfied.*

In general, an instance can have an infinite chase sequence, which means that the chase does not terminate. This is due to the same fact that also Solution-Existence($\mathcal{M}$) is undecidable for cyclic dependencies, shown in the proof of Theorem 3.5.

For solving the DataExchange($\mathcal{M}$) problem we chase the combined instance $\langle I, \emptyset\rangle$. If there is a finite successful chase, we obtain the instance $\langle I, J\rangle$ as last element of the chase sequence, where J is the desired solution. Due the construction of the first-order dependencies, the source instance I remains unchanged. If the last chase step of a chase sequence fails, we also say the chase fails. In the following example we illustrate the chase on the schema mapping from Example 3.4.

Example 3.23. Consider a data exchange scenario $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$, where $\mathbf{S}_1$ consists of two binary relations E, F, and $\mathbf{S}_2$ consists of a ternary relation M and a binary relation N. The dependencies are defined as follows:

$$\Sigma_{12} = \{E(x, y) \wedge E(y, z) \rightarrow \exists w M(x, z, w), \quad (1)$$
$$F(x, y) \rightarrow N(x, y)\} \quad (2)$$
$$\Sigma_2 = \{M(x, y, z) \wedge M(w, y, v) \rightarrow \exists u N(y, u), \quad (3)$$
$$M(x, y, z) \wedge M(x, w, v) \rightarrow y = w\} \quad (4)$$

Let $I = \{E(a, c), E(b, c), E(c, d), F(a, b)\}$ be the source instance. The first element of the chase sequence is the combined instance $\langle I, \emptyset\rangle$. The tuples $E(a, c)$ and $E(c, d)$ satisfy the antecedent of tgd (1). We create tuple $M(a, d, n_1)$, such that n_1 is a fresh value from Var. Let J be a target instance containing exactly this tuple. Dependency (1) is still unsatisfied due to the existence of the tuples $E(b, c)$ and $E(c, d)$ in I. The combined instance $\langle I, J\rangle$ is transformed to $\langle I, J'\rangle$ by adding the fact $M(b, d, n_2)$ to J. Again, n_2 is an unused variable from Var. Tuple $F(a, b)$ satisfies the left hand side of the implication in tgd (2), while the right side is unsatisfied. We transform $\langle I, J'\rangle$ to $\langle I, J''\rangle$ by adding $N(a, b)$ to J'. The first two chase steps added tuples such that tgd (3) has become unsatisfied. Target instance J''' is obtained by extending J'' with the tuple $N(d, n_3)$, where $n_3 \in \mathsf{Var}$. The combined instance $\langle I, J'''\rangle$ satisfies all dependencies in $\Sigma_{12} \cup \Sigma_2$. The chase successfully terminates after the sequence

$$\langle I, \emptyset\rangle, \langle I, J\rangle, \langle I, J'\rangle, \langle I, J''\rangle, \langle I, J'''\rangle$$

of chase steps. The final target instance J''' contains the tuples $M(a, d, n_1)$, $M(b, d, n_2)$, $N(a, b)$ and $N(d, n_3)$.

Consider now a second source instance $I' = \{E(a,b), E(b,c), E(b,d), F(a,b)\}$. We start with $\langle I', \emptyset \rangle$. In the first chase step we satisfy tgd (1), by obtaining a target instance K containing exclusively the fact $M(a,c,n_1)$, where n_1 is an element of Var. Since $E(a,b)$ and $E(b,d)$ are facts in I dependency (1) is still unsatisfied. We transform $\langle I', K \rangle$ into $\langle I', K' \rangle$ by extending K with the fact $M(a,d,n_2)$, where $n_2 \in \mathsf{Var}$. In the combined instance $\langle I', K' \rangle$ tgd (2) and egd (4) are unsatisfied. In the next chase step we consider the latter dependency. Both values c and d are elements from Const. The chase step fails. □

It is worth to mention that it is not defined in which order unsatisfied dependencies have to be handled. Thereby, different chase sequences can lead to different solutions. We will now prove that if a source instance is chased and one chase sequence leads to a solution then all possible chase sequences lead to solutions. Analogously, we will show that if the chase fails, then there is a failing chase step in every possible chase sequence. Therefore, we will need a basic property of the chase step that was implicitly proved by [17, 55]. We will omit proving the following lemma, but a full proof can be found in [37].

Lemma 3.24. [37] *Let $\langle I, J' \rangle$ be the resulting instance of a non-failing chase step for an instance $\langle I, J \rangle$ and a dependency d. Moreover, let $\langle I, K \rangle$ be an instance that satisfies d, and there is a homomorphism $h_1 : \langle I, J \rangle \to \langle I, K \rangle$. Then there also exists a homomorphism $h_2 : \langle I, J' \rangle \to \langle I, K \rangle$.*

The following two theorems where presented first in [37].

Theorem 3.25. [37] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a schema mapping and I a source instance. If $\langle I, J \rangle$ is the result of a successful finite chase of $\langle I, \emptyset \rangle$ with $\Sigma_{12} \cup \Sigma_2$, then J is a universal solution.*

Proof. By definition $\langle I, J \rangle$ is the last element of a finite chase sequence where in each chase step a violation of the dependencies is repaired. Assume an arbitrary solution J' for I under $\mathcal{M}$, then there is clearly a homomorphism $h : \langle I, \emptyset \rangle \to \langle I, J' \rangle$. Furthermore, by definition $\langle I, J' \rangle$ satisfies all dependencies in $\Sigma_{12} \cup \Sigma_2$. By applying Lemma 3.24 at each chase step, we can conclude that there is also a homomorphism $h' : \langle I, J \rangle \to \langle I, J' \rangle$. Since schema $\mathbf{S}_1$ and schema $\mathbf{S}_2$ have by definition no relation symbol in common, it follows that there is also a homomorphism $h'' : J \to J'$. □

Theorem 3.26. [37] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a schema mapping and let I be a source instance. If the finite chase of $\langle I, \emptyset \rangle$ fails, then $\mathsf{Sol}_{\mathcal{M}}(I) = \emptyset$.*

Proof. Let $\langle I, J \rangle$ be the last element of a chase sequence of $\langle I, \emptyset \rangle$ before failing, i.e., the next chase step on egd d fails. Egd d is of the form $\varphi(\mathbf{x}) \to (x_1 = x_2)$. By construction, there is a homomorphism $g : \varphi(\mathbf{x}) \to J$, where $g(x_1)$ and $g(x_2)$ are two distinct elements from Const, say c_1 and c_2. We claim that there is no solution for I under $\mathcal{M}$. Assume the contrary, that there is a solution J'. Clearly, there is a homomorphism $h : \langle I, \emptyset \rangle \to \langle I, J' \rangle$, and by applying Lemma 3.24 at each chase step we can conclude that there

is also a homomorphism $h' : \langle I, J\rangle \to \langle I, J'\rangle$. Furthermore, there must be also a homomorphism $g' : \varphi(\mathbf{x}) \to J'$. Since J' is a solution, egd d has to be satisfied by J', that means $g'(c_1) = g'(c_2)$. This contradicts with the Definition 3.16, where it is stated that homomorphisms map constants on themselves. □

We have showed that if the chase terminates we have either that there is no solution and the chase fails or we get a universal solution, usually called *canonical universal solution*. In Section 3.1.1 we have encountered that if the target tgds are unrestricted deciding if a solution exists is undecidable. This result carries over to the chase procedure. Moreover, we described a restriction to the dependencies, named weakly acyclicity, that assures tractability. Fagin et al. [37] showed that the chase with weakly acyclic dependencies always terminates in polynomial time. This result follows immediately from the result that the length of the chase sequence is polynomially bounded by the input instance and the fact that in each chase step a constant number of facts are added.

Theorem 3.27. [37] *Let Σ_{12} be a set of st-tgds, and let Σ_2 be the union of weakly acyclic tgds and egds. The length of every chase sequence of an instance $\langle I, \emptyset\rangle$ with $\Sigma_{12} \cup \Sigma_2$ is bounded by a polynomial in the size of $\langle I, \emptyset\rangle$.*

Proof. *(Sketch)* An *incoming path* for a node (R, A) in the dependency graph of $\Sigma_{12} \cup \Sigma_2$ is any path ending in (R, A). The *rank* of position (R, A) is the maximum number of occurrences of special edges on any incoming path. The weak acyclicity property assures that the rank of every position is finite. Let r be the maximum rank of positions. Clearly, r is bounded by the number of nodes in the dependency graph. As next step, the set of positions is divided in the subsets $N_0, N_1, \ldots, N_r$, such that N_i contains exactly the positions with rank i. The last element of an arbitrary chase sequence of $\langle I, J\rangle$ is denoted with $\langle I, J'\rangle$. Let n be the number of distinct values of $\mathsf{Const} \cup \mathsf{Var}$ used in $\langle I, J\rangle$.

By induction it can be shown that the total number of distinct values in $\langle I, J'\rangle$ at positions from the subset N_i is bounded by $Q_i(n)$, where $Q_i(n)$ is a polynomial. Roughly, in the base case $Q_0(n) = n$, since there are no special edges and the maximum number of values that can occur at a position (R, A) in $\langle I, J'\rangle$ is the number of values existing in $\langle I, J\rangle$. In the inductive case the polynomial $Q_i(n)$ is $n + G(n) + P(n)$, where $G(n)$ is the number of how many fresh elements from Var can be generated by a chase step in a position of N_i, and $P(n)$ is the number of distinct values copied to positions of N_i from N_j with $j < i$.

Since, there are r subsets N_i, where r is a constant, there also exists a polynomial Q, such that the number of distinct values at a single position in $\langle I, J'\rangle$ is bounded by $Q(n)$. We conclude that the number of all possible tuples in $\langle I, J'\rangle$ is bounded by the polynomial $s \times (Q(n))^p$, where s is the number of relations and p is the number of positions in the schema of $\langle I, J'\rangle$. Notice that p and s are constants. Due the fact that each chase step for a

tgd adds a fact to J', it follows that the length of any chase sequence is at most $s \times (Q(n))^p$. □

Corollary 3.28. *Assume a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$, where Σ_{12} is a set of st-tgds and Σ_2 consists of weakly acyclic tgds and egds. The* DataExchange($\mathcal{M}$) *problem for a source instance I is solvable in polynomial time.*

3.2.2 Second-Order Dependencies

In this section the classical chase technique by Beeri and Vardi [17] is modified to handle second-order tgds. This variant was introduced in [39], and allows to compute universal solutions in polynomial time. Before we define the chase technique we have to introduce some terminology and notation:

Definition 3.29. *Let $\mathbf{V}$ be a set of values and $\mathbf{f}$ a set of function symbols. A* ground term u over $\mathbf{V}$ and $\mathbf{f}$ *is:*

1. *every value in $\mathbf{V}$, and*
2. *every function term $f(u_1, \ldots, u_k)$, where f is k-ary function symbol in $\mathbf{f}$ and $u_1, \ldots, u_k$ are ground terms over $\mathbf{V}$ and $\mathbf{f}$.*

A ground instance with respect to $\mathbf{V}$ and $\mathbf{f}$ *is an instance consisting only of ground terms over $\mathbf{V}$ and $\mathbf{f}$.*

This definition has huge similarities with the definition of terms in Section 3.1.2. While terms from Definition 3.10 are elements of SO-tgds, ground terms on the other hand are values of instances. In our context we use ground instances to describe the values of target instances. Moreover, an instance with values in $\mathbf{V}$, as used for source instances, is a ground instance over $\mathbf{V}$ and an arbitrary set of function symbols $\mathbf{f}$. Hence, ground instances represent a generalization to the instances defined and used before. Recall, that it suffices to consider data exchange settings $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$, where Σ is a single SO-tgd σ of the form

$$\exists \mathbf{f}((\forall \mathbf{x_1}(\varphi_1 \rightarrow \psi_1)) \wedge \ldots \wedge (\forall \mathbf{x_n}(\varphi_n \rightarrow \psi_n))).$$

We denote with $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ a schema mapping with a single SO-tgd. For better readability, we will denote a conjunct $\forall \mathbf{x_i}(\varphi_i \rightarrow \psi_i)$ from σ with C_i.

Definition 3.30. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ be a data exchange scenario and I a source instance. SO-tgd σ is of the form $\exists \mathbf{f}(C_1 \wedge \ldots \wedge C_n)$, where each C_i is defined as $\forall \mathbf{x_i}(\varphi_i \rightarrow \psi_i)$. Assume a mapping $h : \mathbf{x_i} \rightarrow I$ that maps the variables in $\mathbf{x_i}$ to values of I. Moreover, let t and t' be two terms over $\mathbf{x_i}$ and $\mathbf{f}$. The* equality $t = t'$ is satisfied in I under h *if one of the following two conditions is satisfied:*

1. *the equality has the form $x = x'$, where $x, x' \in \mathbf{x_i}$, and $h(x) = h(x')$ holds.*

2. *the equality has the form* $f(t_1, \ldots, t_l) = f(t'_1, \ldots, t'_l)$*, where* $f \in \mathbf{f}$ *and* $t_1, \ldots, t_l, t'_1, \ldots, t'_l$ *are terms over* $\mathbf{x_i}$ *and* $\mathbf{f}$*, and the equalities* $t_1 = t'_1, \ldots, t_l = t'_l$ *are satisfied in* I *under* h*.*

Definition 3.31. *Let* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ *be a schema mapping, where* σ *is composed of conjuncts of the form* $C_i = \forall \mathbf{x_i}(\varphi_i \rightarrow \psi_i)$*. In addition, let* I *be a source instance, and let* $h : \mathbf{x_i} \rightarrow I$ *be a mapping from the variables in* $\mathbf{x_i}$ *to the values of* I*. The function* h *is a* homomorphism from C_i to I *if we have that:*

1. *for every atomic formula* $S(y_1, \ldots, y_k)$ *in* φ_i*, where* $y_1, \ldots, y_k$ *are elements of* $\mathbf{x_i}$*, the tuple* $(h(y_1), \ldots, h(y_l))$ *is in relation* S^I*.*
2. *all equalities in* φ_i *are satisfied in* I *under* h*.*

We are now able to formalize the chase technique for second-order tgds. We divide the definition as in the first-order case into chase step and chase.

Definition 3.32. (Chase Step). *Let* C_i *be a conjunct of the form* $\forall \mathbf{x_i}(\varphi_i \rightarrow \psi_i)$ *and* $\mathbf{V}$*. Let* I *be an an instance over schema* $\mathbf{S}_1$ *with values from set* $\mathbf{V}$*, and let* J *be a ground instance with respect to* $\mathbf{V}$ *and* $\mathbf{f}$ *over schema* $\mathbf{S}_2$*. Moreover, assume that there is a homomorphism* h *from* C_i *to* I*. Conjunct* C_i can be applied to $\langle I, J \rangle$ with h *if* ψ_i *consists of an atomic formula* $T(t_1, \ldots, t_k)$*, such that* $T(h(t_1), \ldots, h(t_k))$ *is not a fact in* J*.*

If C_i *can be applied, then a* chase step *transforms* $\langle I, J \rangle$ *into* $\langle I, J' \rangle$*, where* J' *is the union of* J *and all tuples* $T(h(t_1), \ldots, h(t_k))$ *for which the corresponding relational atom* $T(t_1, \ldots, t_k)$ *is in* ψ_i*.*

Definition 3.33. (Chase). *Let* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ *be a schema mapping, and let* I *be a source instance. A (chase) is a finite sequence of of instances* $\langle I, \emptyset \rangle, \langle I, J_1 \rangle, \ldots \langle I, J_m \rangle$*, where each successive instance is obtained by a chase step on the predecessor. For the last element, called the* result *of the chase, holds that there is no conjunct* C_i *of* σ *and homomorphism* h*, such that* C_i *can be applied to* $\langle I, J_m \rangle$ with h*.*

Several remarks are in order now. The DataExchange($\mathcal{M}$) problem, for a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ and a given source instance I, can be solved by chasing the combined instance $\langle I, \emptyset \rangle$. Since instance I is never changed by a chase step, the resulting chase sequence is by definition finite. This is due to the fact that it cannot happen that we have to apply a conjunct with the same homomorphism more than once. We illustrate the chase for SO-tgds on the schema mapping that is initially presented in Example 3.14.

Example 3.34. Assume a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$, where $\mathbf{S}_1$ consists of two binary relation symbols P and E, and schema $\mathbf{S}_2$ consists of one ternary relation C. The SO-tgd σ is defined as follows:

$$\exists h \forall x \forall y \forall u \forall v (P(x, y) \wedge E(u, v) \wedge (h(x) = h(u)) \rightarrow C(h(x), y, v))$$

Assume the following source instance. Note that all values in I are from Const.

$$I = \{P(\text{"Jim M."}, a), P(\text{"Robert P."}, b), P(\text{"Jimi Hendrix"}, c),$$
$$E(\text{"J. Morrison"}, d), E(\text{"Pete Townshend"}, e), E(\text{"Jimi Hendrix"}, f)\}$$

The chase starts with $\langle I, \emptyset \rangle$. We have to find a homomorphism g from the conjunct of σ to the instance I. In such a homomorphism every atomic formula has to be mapped on a tuple from I. Furthermore, every equality has to be satisfied in that homomorphism. In this case $g(x) = g(u)$ is exactly then satisfied if we have $x = u$. It is not hard to see, that there is only one valid homomorphism g, where $P(g(x), g(y))$ is $P(\text{"Jimi Hendrix"}, c)$ and $E(g(u), g(v))$ is $E(\text{"Jimi Hendrix"}, f)$. By applying the conjunct from σ with g we receive the tuple $C(h(\text{"Jimi Hendrix"}), c, f)$. As already indicated in Example 3.14, we cannot find any other applicable homomorphism from σ to I. Thus, the target instance $J = \{C(h(\text{"Jimi Hendrix"}), c, f)\}$ is a solution for I under $\mathcal{M}$. Furthermore, J is a ground instance with respect to the set of source values of I and the set of function symbols $\{h\}$.

Each ground function term can be considered as a distinct value. Hence, we can replace all ground function terms by distinct elements from Var. We obtain $J = \{C(n, c, f)\}$ as final solution, where n is an element from Var. □

We are now interested to show that the chase technique for SO-tgds always delivers universal solutions. Therefore, we need the following property, shown by [39]:

Lemma 3.35. [39] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$ be a schema mapping, where σ is an SO-tgd, and let $\langle I, J' \rangle$ be the resulting instance of a chase step on instance $\langle I, J \rangle$ over the schema $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$. Moreover, assume a combined instance $\langle I, K \rangle$ over $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$ that satisfies σ, and there is a homomorphism h from $\langle I, J \rangle$ to $\langle I, K \rangle$. Then, h is also a homomorphism from $\langle I, J' \rangle$ to $\langle I, K \rangle$.*

Due to Lemma 3.35, we are able to show that the result of a chase is always a universal solution. The following result originates from [39].

Theorem 3.36. [39] *Consider a data exchange scenario $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$, where σ is a single SO-tgd. Then for every source instance I, chasing the combined instance $\langle I, \emptyset \rangle$ with σ results in an instance $\langle I, J \rangle$, such that J is a universal solution for I under $\mathcal{M}$.*

Proof. The second-order tgd σ is of the form $\exists \mathbf{f} \sigma'$, where σ' is in first order. Assume an arbitrary solution K for I under $\mathcal{M}$. By definition, we have that $\langle U; I, K \rangle \models \sigma$, where U is a universe. Consider a collection of functions $\mathbf{f^0}$ over U such that $\langle U; I, K \rangle \models \sigma'[\mathbf{f} \to \mathbf{f^0}]$.

Let $\mathbf{V}$ be the values that occur in I. We define a mapping h as the identity function over $\mathbf{V}$, and $h(f(u_1, \ldots, u_k)) = f^0(h(u_1), \ldots, h(u_k))$ for every ground function term $f(u_1, \ldots, u_k)$ over $\mathbf{V}$ and $\mathbf{f}$. It is easy to see that h is a

homomorphism from $\langle I, \emptyset \rangle$ to $\langle I, K \rangle$. By applying Lemma 3.35 at each chase step, we can conclude that $h : \langle I, J \rangle \rightarrow \langle I, K \rangle$ is also a homomorphism. Since K is an arbitrary solution, we have that J has to be a universal solution. □

We have already argued that the chase of a source instance combined with an empty set is decidable. Fagin et al. [39] showed that the chase is for second-order tgds actually tractable.

Theorem 3.37. [39] *Assume a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma)$, where σ is a single SO-tgd, and an instance I over $\mathbf{S}_1$. Chasing $\langle I, \emptyset \rangle$ with σ terminates in polynomial time in the size of I.*

Proof. We want to show that the chase terminates in polynomial time in the size of I. Therefore, it has to be shown first that the corresponding chase sequence has length bounded by a polynomial in the size of I. Secondly, we have to prove that each chase step is computable in polynomial time.

For each conjunct C of σ we can have at most n^k homomorphisms, where n is the number of distinct values in I and k is the maximum number of universally quantified variables in a single conjunct of σ. From σ there can be $c \times n^k$ homomorphisms into I, where c is the number of conjuncts in σ. Since an SO-tgd can only fire once for each homomorphism and the number of homomorphisms does not change during the chase, we conclude that we can have at most $c \times n^k$ chase steps.

Let t be the maximum number of atoms in a formula ψ in σ. We know that in a chase step exactly one conjunct is satisfied. In other words in a chase step there can be added at most t tuples. Therefore, the number of tuples in a solution is at most $t \times c \times n^k$. Let $q(n)$ be the time to check if a possible homomorphism candidate is indeed a homomorphism from a considered conjunct to I as specified in Definition 3.30. It is immediate that time $q(n)$ is polynomial in n. We can find at each chase step an applicable homomorphism in at most $c \times n^k \times q(n)$ time. For each applicable homomorphism we have to check that the implied tuples do not already exist in the intermediary combined instance. This can be done in the number of at most added tuples in a chase step times the number of tuples in a solution. More formally, the existence of t tuples in the target can be checked in at most $t^2 \times c \times n^k$ steps. It follows, that the chase can be computed in at most $t^2 \times c \times n^k + c \times n^k \times q(n)$ steps.

We conclude that the overall needed time to chase is at most the maximum number of chase steps times the time we need at most for each chase step. More formally, the needed time is $t^2 \times c^2 \times n^{2k} + c^2 \times n^{2k} \times q(n)$. Clearly, this number is a polynomial in the size of I. □

3.3 Query Answering

In this section query answering with respect to schema mappings is considered. Intuitively, if we have a source instance and a query formulated over the target schema of a data exchange scenario, we would like to have an an-

swer that represents the answers of evaluating the query over each solution. In the following definition we adapt the concept of certain answers from Section 2.1.3 to queries over solutions from schema mappings:

Definition 3.38. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a data exchange scenario, and let I be a source instance. Assume a k-ary query q over $\mathbf{S}_2$. The* set of certain answers of q with respect to I under $\mathcal{M}$, *denoted with* $\mathsf{certain}_{\mathcal{M}}(q, I)$, *is defined as* $\bigcap\{q(J) \mid J \in \mathsf{Sol}_{\mathcal{M}}(I)\}$.

Sometimes, we say only *certain answers* of a query q if the schema mapping is understood from the context and the source instance can be arbitrary. Furthermore, if we speak of query answering in the context of data exchange we assume that queries are formulated over the target schema. The associated decision problem asks if the certain answers for a Boolean query include the empty tuple or not. More formally:

PROBLEM:	QUERYEVALUATION$(\mathcal{M}, q)$
INPUT:	a source instance I
QUESTION:	$\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$?

An important variation of QUERYEVALUATION$(\mathcal{M}, q)$ asks if a given tuple t is included in the certain answers of a k-ary query. This variation can be transformed in the QUERYEVALUATION$(\mathcal{M}, q)$ problem by substituting the tuple t into the query. Moreover, we are able to compute the certain answers of a k-ary query q, by iteratively checking if one of the polynomially many k-ary tuples is included in $\mathsf{certain}_{\mathcal{M}}(q, I)$. Thus, we are allowed to consider without loss of generality only Boolean conjunctive queries.

Essentially, there are two issues to take into account when we are computing the certain answers of a query. The first issue is that a source instance can have infinitely many solutions under a given data exchange setting. Secondly, a solution is in general incomplete, and as described in Section 2.1.3 query answering over incomplete data is coNP-complete.

We show in Section 3.3.1 that the certain answers of a UCQ using first-order or second-order dependencies, is computable in polynomial time in the size of the source instance. In Section 3.3.2 we consider an extension of UCQs that allows inequalities. It turns out, that the technique from Section 3.3.1 cannot be applied on UCQs with inequalities. Moreover, the complexity rises in general from polynomial time to coNP. Though, there are restricted cases of UCQs with inequalities such that the QUERYEVALUATION$(\mathcal{M}, q)$ problem remains tractable. We discuss two of them.

3.3.1 Unions of Conjunctive Queries

In this section it is shown that the certain answers of a union of conjunctive queries can be computed in polynomial time using a single universal solution. First, we have to introduce a subset from an answer that contains only constants:

Definition 3.39. *Let J be an incomplete instance over a schema $\mathbf{S}$ and q a query over $\mathbf{S}$. We write $q(J)_{\downarrow}$ for the set of those tuples in $q(J)$ that contain only values from* Const*. In the case that q is a Boolean query we have that $q(J)_{\downarrow} = q(J)$.*

Fagin et al. [37] showed the following result for first-order dependencies, but it can also applied to second-order dependencies [39]:

Theorem 3.40. [37] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a schema mapping, where Σ is either a set of SO-tgds or a union of st-tgds, target tgds and egds. Moreover, consider a universal solution J for a source instance I, and a UCQ q over $\mathbf{S}_2$. Then,* $\mathsf{certain}_{\mathcal{M}}(q, I) = q(J)_{\downarrow}$.

Proof. Let t be an arbitrary tuple, such that $t \in \mathsf{certain}_{\mathcal{M}}(q, I)$. From Definition 3.38 we know that t must be in every solution, and thus $\mathsf{certain}_{\mathcal{M}}(q, I) \subseteq q(J)$. Furthermore, t consists only of values from Const. Assume to the contrary, that $n \in t$, where $n \in \mathsf{Var}$. By definition n must be contained in every $\{q(K) \mid K \in \mathsf{Sol}_{\mathcal{M}}(I)\}$ and therefore also in every $K \in \mathsf{Sol}_{\mathcal{M}}(I)$. Take an arbitrary $K \in \mathsf{Sol}_{\mathcal{M}}(I)$ and transform K to K' by replacing every occurrence of n with an unused element from Const. Clearly, K' is a valid solution and must be contained in $\mathsf{Sol}_{\mathcal{M}}(I)$, which is a contradiction. We conclude that also $\mathsf{certain}_{\mathcal{M}}(q, I) \subseteq q(J)_{\downarrow}$ holds.

Let t be an arbitrary tuple from $q(J)_{\downarrow}$. There must be a disjunct of the form $\exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$ from q, such that there is homomorphism $g : \varphi(\mathbf{x}, \mathbf{y}) \to J$ and $g(\mathbf{x}) = t$. We know from the definition of universal solutions that there must be also a homomorphism $h : J \to J'$, where J' is an arbitrary solution from $\mathsf{Sol}_{\mathcal{M}}(I)$. Since Definition 3.16 states that constants must be mapped on themselves, we have that $h(g(\mathbf{x})) = t$ and therefore also $t \in q(J')$. We conclude that $q(J)_{\downarrow} \subseteq \mathsf{certain}_{\mathcal{M}}(q, I)$ holds. □

Example 3.41. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a schema mapping, where $\mathbf{S}_1$ consists of a ternary relation symbol E and a binary relation symbol F, and $\mathbf{S}_2$ consists of a single ternary relation symbol M. Σ consists of the following first-order dependencies:

$$\begin{aligned} \Sigma_{12} = \{&E(x, y, z) \to M(x, y, z), \\ &F(x, y) \to \exists z M(x, y, z)\} \\ \Sigma_2 = \emptyset \end{aligned}$$

Assume $I = \{E(a, b, c), F(d, e)\}$ as source instance, and the conjunctive query $q = \exists x M(x, y, z)$. It is easy to see that $\mathsf{certain}_{\mathcal{M}}(q, I) = \{(b, c)\}$.

We compute the universal solution $J = \{M(a, b, c), M(d, e, n)\}$, where $n \in \mathsf{Var}$, by chasing $\langle I, \emptyset \rangle$ with Σ. Evaluating q with instance J results in the set $\{(b, c), (e, n)\}$. We have that $q(J)_{\downarrow} = \{(b, c)\}$. □

In addition it can be shown that only for universal solutions the equality $\mathsf{certain}_{\mathcal{M}}(q, I) = q(J)_{\downarrow}$ holds. This fact further approves that universal solutions are a good choice for materialization. Since we can compute universal

solutions in polynomial time for first-order and second-order dependencies, the following corollary results immediately from Theorem 3.40:

Corollary 3.42. *Consider a data exchange scenario $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$, such that Σ is either a single SO-tgd or the union of st-tgds, egds and a weakly acyclic set of target tgds. Moreover, consider an arbitrary source instance I and a UCQ over $\mathbf{S}_2$. The set $\mathsf{certain}_{\mathcal{M}}(q, I)$ can be computed in polynomial time in the size of I.*

3.3.2 Queries with Inequalities

In this section we study query answering with queries that allow inequalities. We focus on an extension of CQs and UCQs that allows inequalities, defined as follows:

Definition 3.43. *A* conjunctive query with inequalities *over a schema $\mathbf{S}$, or CQ$^{\neq}$ for short, is a formula of the form $\exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$, such that $\varphi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over $\mathbf{S}$ with inequalities of the form $z_i \neq z_j$, where z_i and z_j are variables in $\mathbf{x} \cup \mathbf{y}$. A* union of conjunctive queries with inequalities *is a disjunction of conjunctive queries with inequalities and is denoted with UCQ$^{\neq}$.*

Unfortunately, [37] showed that the technique presented in Section 3.3.1 does not always lead to the certain answers of a CQ$^{\neq}$ respectively a UCQ$^{\neq}$.

Theorem 3.44. [37] *There is a data exchange setting $\mathcal{M}$, a universal solution J for source instance I, and conjunctive query with inequalities q, such that $\mathsf{certain}_{\mathcal{M}}(q, I) = q(J)_{\downarrow}$ does not hold.*

Proof. Let source schema $\mathbf{S}_1$ and target schema $\mathbf{S}_2$ each consist of one binary relation, and let the dependencies of $\mathcal{M}$ be defined as follows:

$$\Sigma_{12} = \{S(x, y) \rightarrow \exists z(T(x, z) \wedge T(z, y))\}$$
$$\Sigma_2 = \emptyset$$

Assume that q is a Boolean CQ$^{\neq}$ defined as $\exists x \exists y(T(x, y) \wedge (x \neq y))$, and the source instance I is the set $\{S(a, a)\}$. A valid solution for I under $\mathcal{M}$ is $J' = \{T(a, a)\}$, and therefore $q(J') = \mathbf{false}$. We conclude $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{false}$. A chase of $\langle I, \emptyset \rangle$ with Σ_{12} results in the universal solution $J = \{T(a, n), T(n, a)\}$, where $a \in \mathsf{Const}$ and $n \in \mathsf{Var}$. Since, by definition $\mathbf{Const} \cap \mathbf{Var} = \emptyset$ we have that $a \neq n$, and therefore also $q(J) = q(J)_{\downarrow} = \mathbf{true}$. □

Apparently, we need a new approach for solving unions of conjunctive queries with negations. Several remarks are in order now. First of all, we consider without loss of generality only Boolean queries. A Boolean conjunctive query q with inequalities is of the form

$$\exists \mathbf{x} \left(\varphi(\mathbf{x}) \wedge \left(\bigwedge_i (x_i^1 \neq x_i^2) \right) \right),$$

where $\varphi(\mathbf{x})$ is a conjunctive query, i.e. a conjunction of atomic formulas. The negation of this CQ$^{\neq}$ q can be transformed into an equivalent formula of the form

$$\forall \mathbf{x} \left(\varphi(\mathbf{x}) \rightarrow \left(\bigvee_i (x_i^1 = x_i^2) \right) \right),$$

called *disjunctive egd*. Notice, that disjunctive egds are a generalization from the so far used egds. In addition, if e is a disjunctive egd and has the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow ((x_1^1 = x_1^2) \vee \ldots \vee (x_l^1 = x_l^2)))$, we write $e_1, \ldots, e_l$ for the egds $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow (x_i^1 = x_i^2))$ *associated* with e, where $1 \leq i \leq l$. A further observation is that we can consider a UCQ$^{\neq}$ q as a disjunction over the union of the set of CQs C with the set of CQs$^{\neq}$ E. The approach for evaluating UCQs$^{\neq}$ proposed by [37] utilizes the following property:

Lemma 3.45. [37] *There exists a solution J for source instance I that satisfies every CQ$^{\neq}$ in E, and satisfies no CQ in C if and only if* $\mathsf{certain}_{\mathcal{M}}(q, I) =$ **false**.

We define now the disjunctive chase, which is a generalization of the chase from Section 3.2.1 and represents a special case of the chase proposed in [32]. Like before, we distinguish between chase step and chase.

Definition 3.46. (Disjunctive chase step). *Let $\langle I, J \rangle$ be an instance, and let $e : \forall \mathbf{x} \varphi(\mathbf{x}) \rightarrow ((x_1^1 = x_1^2) \vee \ldots \vee (x_l^1 = x_l^2))$ be a disjunctive egd. Assume that e is unsatisfied, i.e. there is a homomorphism h from $\varphi(\mathbf{x})$ to $\langle I, J \rangle$ such that $h(x_1^1) \neq h(x_1^2) \wedge \ldots \wedge h(x_l^1) \neq h(x_l^2)$ holds. Hence, every associated egd $e_1, \ldots, e_l$ is unsatisfied. There are two different outcomes:*

1. *The disjunctive chase step fails if no associated egd $e_1, \ldots, e_l$ can be satisfied, i.e. if all $h(x_i^1)$ and $h(x_i^2)$ are mapped to elements of* Const.
2. *Let $e_{i_1} \ldots e_{i_r}$ be those egds where at least one of $h(x_{i_j}^1)$ and $h(x_{i_j}^2)$ is assigned to an element of* Var, *say $h(x_{i_j}^1)$. The* disjunctive chase step *results in a set of instances $\{\langle I, J_1 \rangle, \ldots, \langle I, J_r \rangle\}$, such that J_j satisfies e_{i_j} and is obtained by replacing every occurrence of $h(x_{i_j}^1)$ in J by $h(x_{i_j}^2)$.*

Definition 3.47. (Disjunctive chase). *Assume a set of st-tgds Σ_{12}, a union of a set of tgds with a set of egds Σ_2, a set of disjunctive egds E, and an instance $\langle I, J \rangle$. A* chase tree of $\langle I, J \rangle$ with $\Sigma_{12} \cup \Sigma_2 \cup E$ *is a possibly infinite tree, where $\langle I, J \rangle$ is the root node. The set of children $\{\langle I, J_{i_1} \rangle, \ldots, \langle I, J_{i_r} \rangle\}$ of a node $\langle I, J_i \rangle$ is obtained depending on the considered dependency either by a disjunctive chase step or by the chase step described in Definition 3.21.*

A disjunctive chase is called finite *if the chase tree has finitely many nodes, and on each leaf node there is either no dependency in $\Sigma_{12} \cup \Sigma_2 \cup E$ unsatisfied, or there is a dependency in $\Sigma_2 \cup E$ such that the chase step respectively disjunctive chase step fails.*

Notice that also the disjunctive chase has the property that there is no order in which unsatisfied dependencies have to be processed. Thus, instead of chasing $\langle I, \emptyset \rangle$ with $\Sigma_{12} \cup \Sigma_2 \cup E$, we are also allowed to chase $\langle I, J \rangle$ with $\Sigma_2 \cup E$, where J is a universal solution for source instance I. The QueryEvaluation$(\mathcal{M}, q)$ problem for UCQs$^{\neq}$ comes down to generating a chase tree of the source instance and checking if a leaf in the chase tree exists that satisfies all disjunctive egds in E and no CQ in C. Fagin et al. [37] showed that this is possible in coNP.

Theorem 3.48. [37] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a schema mapping, where Σ_{12} is a set of source-to-target tgds and Σ_2 is a union of a set of egds with a weakly acyclic set of tgds. Then, computing the certain answers of a union of conjunctive queries with inequalities is in coNP.*

Proof. *(Sketch)* Consider a source instance I and a UCQ$^{\neq}$ q, where the sets C and E denote the disjuncts in q that are CQs and CQs$^{\neq}$, respectively. Assume that $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{false}$. We argued already in Section 3.2.1 that we can compute a universal solution in polynomial time. We guess a sequence of homomorphisms and dependencies for the chase steps. Moreover, we guess a sequence of decisions that determines which child we consider in each chase step. Thereby, we are able to compute a solution J for I under $\mathcal{M}$ that satisfies all CQs$^{\neq}$ in E in polynomial time. Subsequently, we check in polynomial time that J satisfies no CQ in C, using the technique from Section 3.3.1. Consider now the opposite case that $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$. We can either have that none of the leaves in the chase tree satisfies all CQs$^{\neq}$ in E, or all leaves that satisfy all CQs$^{\neq}$ in E satisfy also at least one CQ in C. We conclude that deciding if $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{false}$ is in NP, and therefore solving the QueryEvaluation$(\mathcal{M}, q)$ problem is in coNP. □

Theorem 3.48 represents an upper complexity bound for computing the certain answers of a UCQ with inequalities. It can be shown that the QueryEvaluation$(\mathcal{M}, q)$ problem is also coNP-hard for queries with inequalities. We do not explicitly discuss this result since it is implied by Theorem 3.51. A self-evident question is if we can restrict the QueryEvaluation$(\mathcal{M}, q)$ problem with UCQs$^{\neq}$ in such a way that it becomes solvable in polynomial time. We consider two restrictions of UCQs$^{\neq}$ that allow computing the certain answers in polynomial time. In the first restriction the number of inequalities per disjunct is limited. The following result, proposed by [37], shows that there is an algorithm that solves the QueryEvaluation$(\mathcal{M}, q)$ problem for UCQs$^{\neq}$, where the number of inequalities per disjunct is limited to one, in polynomial time.

Theorem 3.49. [37] *Consider a data exchange scenario $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$, where Σ_{12} is a set of st-tgds and Σ_2 is a set of egds and weakly acyclic tgds. Then, the certain answers of a union of conjunctive queries with at most one inequality per disjunct can be computed in polynomial time.*

Proof. Assume an arbitrary source instance I. We compute a universal solution J for I in polynomial time using the chase from Section 3.2.1.

Without loss of generality let q be a UCQ$^{\neq}$ with at most one inequality per disjunct, such that q has the form $q_1 \vee q_2$, where q_1 is the disjunction of a set C of CQs and q_2 is the disjunction of a set E of CQs$^{\neq}$. Recall, that disjunctive egds with exactly one inequality degenerate to egds from Definition 3.3. Thus, we can transform q_2 into a conjunction of a set E of egds. We chase $\langle I, J\rangle$ with $\Sigma_2 \cup E$ using the chase defined in Section 3.2.1. We have already shown that this chase runs in polynomial time and either fails or results in a target instance J'. If the chase fails we set according to Lemma 3.45 $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$. If the chase does not fail we check for each CQ in C if it is satisfied by J'. Clearly, there are only polynomially many CQs in C and each of them can be evaluated according to Theorem 3.40 in polynomial time. If J' satisfies at least one of the CQs in C we have $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$, otherwise $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{false}$. It follows that this algorithm has polynomial runtime.

We consider now the correctness of the described algorithm. If the chase $\langle I, J\rangle$ with $\Sigma_2 \cup E$ fails, we have that q_2 is satisfied in J. Since J is universal, every solution satisfies q_2 and therefore $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$. If the chase succeeds with a solution J', we have that some of the solutions for I under $\mathcal{M}$ satisfy E and J' is universal for them. Those solutions which do not satisfy E, satisfy by definition q_2 and with it q. If J' satisfies at least one CQ in C, we have that every solution that satisfy E satisfies also this CQ. It follows that either q_1 or q_2 is satisfied, and hence $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{true}$. If J' satisfies no conjunctive query in C, then J' is a witness that $\mathsf{certain}_{\mathcal{M}}(q, I)$ must be **false**. □

The question arises if we are able to find efficient algorithms for UCQs$^{\neq}$ with more than one inequality per disjunct and no further restrictions. It turns out that, under the assumption that P≠NP, this is not possible [37]. We discuss here an even stronger result proposed by Madry [54]. Therefore we have to introduce a restricted form of schema mappings that is already intensively studied in the data integration context [52].

Definition 3.50. *A data exchange setting $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ is called* local-as-view setting*, or* LAV setting *for short, if all source-to-target dependencies in Σ_{12} are of the form $\forall \mathbf{x}(R(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$, where $R(\mathbf{x})$ is a single atomic formula over $\mathbf{S}_1$,ψ is a conjunction of atomic formulae over $\mathbf{S}_2$, and all variables in $\mathbf{x}$ occur in ψ.*

Theorem 3.51. [54] QueryEvaluation$(\mathcal{M}, q)$ *for LAV settings and conjunctive queries with two or more inequalities is coNP-complete.*

Proof. *(Sketch)* Clearly, Theorem 3.51 implies membership in coNP, but for the restricted case of LAV settings membership was already shown before by [1]. Hardness is shown by a reduction from the complement of 3Sat. Basically, the reduction creates from a formula ϕ in 3CNF a source instance I with four binary relations P, L, R and N. Relation P can be interpreted as a digraph, such that each tuple in P represents an arc. The target schema contains two binary relations P' and N'. A solution J contains in P' all tuples from P. Furthermore, for each tuple in relation L and R there must

be a directed path of length two respectively four in the digraph represented by P'. The query q asks if there is a node with three or more outgoing arcs in the digraph represented by P'. It can be shown that ϕ is satisfiable if and only if $\mathsf{certain}_{\mathcal{M}}(q, I) = \mathbf{false}$. □

The above result also states that we have to consider further restrictions if we want to find a tractable algorithm that computes the certain answers for UCQs$^{\neq}$ with more than one inequality per disjunct. We discuss now the restrictions that allow finding an efficient algorithm for UCQs$^{\neq}$ with at most two inequalities per disjunct. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping with a universal solution J for source instance I, and let q be a UCQ$^{\neq}$ with at most two inequalities per disjunct. Since $\mathcal{M}$ has no target dependencies, we do not have to consider the case where for a source instance there exists no solution. A first observation is that, if we allow in Σ_{12} only full st-tgds, it becomes easy to show that QueryEvaluation$(\mathcal{M}, q)$ is tractable. This is due the fact that if a universal solution satisfies q then all others solutions do. We can relax this restriction by allowing elements from Var occur in I, such that variables from q that appear in inequalities cannot be mapped on them. Arenas et al. [12] proposed even less restrictive conditions that allow QueryEvaluation$(\mathcal{M}, q)$ to be tractable:

Definition 3.52. *Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$. We say that the i-th attribute of a relation symbol $T \in \mathbf{S}_1$* cannot be nullified *if there is no st-tgd in Σ_{12} that has the i-th attribute of T existentially quantified.*

Let q be a Boolean conjunctive query with two inequalities of the form $\exists\mathbf{x}(\varphi(\mathbf{x}) \wedge (x_1^1 \neq x_1^2) \wedge (x_2^1 \neq x_2^2))$. The conjunctive query q has almost constant inequalities *under $\mathcal{M}$, if at least x_i^1 or x_i^2 cannot be nullified, where $1 \leq i \leq 2$. Query q has* constant joins *under $\mathcal{M}$ if only variables that occur in φ once can be nullified.*

Example 3.53. Consider a data exchange scenario $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where $\mathbf{S}_1$ consists of one binary relation symbol, and $\mathbf{S}_2$ consists of a ternary relation symbol. Σ_{12} is defined as follows:

$$\Sigma_{12} = \{E(x, y) \rightarrow \exists z M(x, y, z)\}$$

The first two attributes of the relation symbol M cannot be nullified, while the third can be nullified. Consider the following three queries:

$$\begin{aligned} q_1 =& \exists x \exists y \exists z \exists u \exists v (M(x, y, z) \wedge M(x, u, v) \wedge y \neq u \wedge z \neq v) \\ q_2 =& \exists x \exists y \exists z \exists u (M(x, y, z) \wedge M(x, u, z) \wedge y \neq u) \\ q_3 =& \exists x \exists y \exists z \exists u \exists v (M(x, y, z) \wedge M(x, u, v) \wedge y \neq v \wedge z \neq u) \end{aligned}$$

We have that q_1 has constant joins, but $z \neq v$ destroys the almost constant inequality property. Query q_2 has almost constant inequalities but not constant joins, because of the occurrences of variable z. The last query q_3 satisfies both conditions. □

The following result was shown by Arenas et al. [12]. Moreover, they showed that tractability for QueryEvaluation$(\mathcal{M}, q)$ is immediately lost, if any of the restrictions is removed.

Theorem 3.54. [12] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping. Assume a query q consisting of a disjunction of conjunctive queries with at most one inequality and constant joins under $\mathcal{M}$, and conjunctive queries with two inequalities, constant joins and almost constant inequalities under $\mathcal{M}$. Then,* QueryEvaluation$(\mathcal{M}, q)$ *can be solved in polynomial time.*

3.4 Composing Schema Mappings

So far, we have exclusively studied properties and problems where the data exchange setting is considered as fixed. The fundamental idea in model management [19] is to consider operations on data exchange settings. The two most fundamental operators are inversion and composition. The intuition of the inversion operator is to produce from a given schema mapping $\mathcal{M}_{12}$ a schema mapping $\mathcal{M}_{21}$, such that if J is a solution for I under $\mathcal{M}_{12}$ then I is a solution for J under $\mathcal{M}_{21}$. The investigation [36, 40, 12, 41] in this area has showed that even defining the exact semantics of the inversion operator is by no means trivial. The composition operator produces a schema mapping $\mathcal{M}_{13}$ that yields to the same solutions as the successive application of two given schema mappings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. In contrast to the inversion operator, there is an agreement in the community about the semantics of the composition. In this section the composition operator which is studied in [53, 39, 58, 19] is discussed, but we primarily focus on the results of Fagin et al. [39]. We start by giving the definition of the composition operator.

Definition 3.55. *Consider the data exchange settings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ and $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$. Schema mapping $\mathcal{M}_{13}$ is a* composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$*, denoted as $\mathcal{M}_{13} = \mathcal{M}_{12} \circ \mathcal{M}_{23}$, if $\mathsf{Inst}(\mathcal{M}_{13})$ consists of the following instances:*

$$\{\langle I_1, I_3\rangle \mid \textit{there is an } I_2 \textit{ so that } \langle I_1, I_2\rangle \in \mathsf{Inst}(\mathcal{M}_{12}) \wedge \langle I_2, I_3\rangle \in \mathsf{Inst}(\mathcal{M}_{23})\}$$

We discuss in this section only data exchange settings without target dependencies. Moreover, the composition of two schema mappings always exists and is unique up to logical equivalence. For the complexity analysis of compositions the *composition query* is of interest, which is a decision problem where we ask if a set of pairs of instances is contained in the composition of two schema mappings. More formally:

Problem:	CompositionQuery$(\mathcal{M}_{12}, \mathcal{M}_{23})$
Input:	instances I_1 and I_3
Question:	is $\langle I_1, I_3\rangle \in \mathsf{Inst}(\mathcal{M}_{12}) \circ \mathsf{Inst}(\mathcal{M}_{23})$?

The following example illustrates a composition query and shows a composition of two schema mappings with first order dependencies:

Example 3.56. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two schema mappings, such that $\mathbf{S}_1$ and $\mathbf{S}_3$ consist of one ternary relation

symbol. Schema $\mathbf{S}_2$ consists of one binary relation symbol. The source-to-target tgds of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ are defined as follows:

$$\Sigma_{12} = \{D(x,y,z) \rightarrow E(x,y) \wedge E(y,z)\}$$
$$\Sigma_{23} = \{E(x,y) \wedge E(x,z) \rightarrow \exists w F(x,y,w)$$

Notice that the st-tgd in $\mathcal{M}_{12}$ is full. Assume a source instance $I_1 = \{ D(a,b,c), D(c,a,b)\}$ over $\mathbf{S}_1$. A chase of $\langle I_1, \emptyset\rangle$ with Σ_{12} results in an instance $I_2 = \{E(a,b), E(b,c), E(c,a)\}$. Furthermore, a chase of $\langle I_2, \emptyset\rangle$ with Σ_{23} results in an instance $I_3 = \{F(a,b,n_1), F(b,c,n_2), F(c,a,n_3)\}$, where $n_1, n_2, n_3 \in \mathsf{Var}$. Thus, the CompositionQuery$(\mathcal{M}_{12}, \mathcal{M}_{23})$ problem for I_1 and I_3 results in the answer **true**. Let $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ be a data exchange scenario with the following dependencies:

$$\begin{aligned}\Sigma_{13} = \{&D(x,y,z) \wedge D(x,u,v) \rightarrow \exists w F(x,y,w),\\ &D(x,y,z) \wedge D(v,x,u) \rightarrow \exists w F(x,y,w),\\ &D(z,x,y) \wedge D(x,u,v) \rightarrow \exists w F(x,y,w),\\ &D(z,x,y) \wedge D(v,x,u) \rightarrow \exists w F(x,y,w)\}\end{aligned}$$

It can be seen that $\mathcal{M}_{13}$ is the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. The technique for producing those dependencies is described in the proof of Theorem 3.57. Furthermore, I_3 is a solution for I_1 under $\mathcal{M}_{13}$. □

There are two obvious ways to solve the CompositionQuery$(\mathcal{M}_{12}, \mathcal{M}_{23})$ problem for two schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ and two given instances I_1 and I_3. The naive approach is to guess an instance I_2 over $\mathbf{S}_2$ and check if $\langle I_1, I_2\rangle \models \Sigma_{12}$ and $\langle I_2, I_3\rangle \models \Sigma_{23}$ holds. If we assume that $\langle I_1, I_2\rangle \models \Sigma_{12}$ and $\langle I_2, I_3\rangle \models \Sigma_{23}$ can be checked in polynomial time, as it is for first and second-order dependencies, then the CompositionQuery$(\mathcal{M}_{12}, \mathcal{M}_{23})$ problem can be solved in nondeterministic polynomial time. The second, more sophisticated approach is to compute a composition $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ and check if $\langle I_1, I_3\rangle \models \Sigma_{13}$ holds.

In Section 3.4.1 the composition of two schema mappings with first-order dependencies and the complexity of the corresponding composition query is studied. It turns out that the composition of two schema mappings cannot always be defined with st-tgds. For the special case where at least the first schema mappings consist only of full st-tgds a finite composition can be found, such that the composition query can be solved in polynomial time. Otherwise, the naive approach has to be used and the complexity raises to NP-completeness. In Section 3.4.2 compositions with schema mappings that use SO-tgds are discussed. The key result is that in contrast to schema mappings with first-order dependencies, schema mappings with SO-tgds allow always to compute a composition. Nevertheless, the CompositionQuery$(\mathcal{M}_{12}, \mathcal{M}_{23})$ problem remains NP-complete.

3.4.1 First-Order Dependencies

The definition of the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem considers both data exchange settings $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ as constant, and therefore computing a composition $\mathcal{M}_{13}$ does not result in a complexity increase. Moreover, for a schema mapping $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ it can be decided in polynomial time if an instance $\langle I_1, I_2 \rangle$ over $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$ satisfies all dependencies in Σ_{12}. It follows that if we can compute a composition $\mathcal{M}_{13}$ consisting of a finite set of st-tgds, then the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem is tractable. The following result, proposed in [39], shows a fragment of the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem that allows computing a finite composition with first-order dependencies.

Theorem 3.57. [39] *The composition of two data exchange settings* $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ *and* $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$*, where* Σ_{12} *is a set of full st-tgds and* Σ_{23} *is a set of st-tgds, can be defined with a finite set of st-tgds. Hence, the* CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) *problem can be solved in polynomial time.*

Proof. *(Sketch)* An introductory observation is that every full st-tgds can be represented by a set of st-tgds with only a single atomic formula in their conclusion. Therefore, without loss of generality it can be assumed that the dependencies in Σ_{12} are of the form $\forall \mathbf{z_i}(\varphi_i(\mathbf{z_i}) \rightarrow R_i(\mathbf{u_i}))$, where $\mathbf{u_i} \in \mathbf{z_i}$. Moreover, the dependencies in Σ_{23} are of the form $\forall \mathbf{x}((R_1(\mathbf{x_1}) \wedge \ldots \wedge R_k(\mathbf{x_k})) \rightarrow \exists \mathbf{y} \psi(\mathbf{x_0}, \mathbf{y}))$. If an st-tgd τ in Σ_{23} has an atomic formula R_i with $1 \leq i \leq k$ in their antecedent, such that there is no st-tgd in Σ_{12} with R_i in the conclusion, then we can exclude τ from further considerations. For all other st-tgds in Σ_{23}, we construct a set of all possible st-tgds of the form

$$\forall \mathbf{z}' \forall \mathbf{x}((\varphi_1[\mathbf{u_1} \mapsto \mathbf{x_1}] \wedge \ldots \wedge \varphi_k[\mathbf{u_k} \mapsto \mathbf{x_k}]) \rightarrow \exists \mathbf{y} \psi(\mathbf{x_0}, \mathbf{y})),$$

where φ_i is an antecedent of an st-tgd in Σ_{12} with $R_i(\mathbf{u_i})$ as conclusion. The expression $\varphi_i[\mathbf{u_i} \mapsto \mathbf{x_i}]$ replaces all occurrences of $\mathbf{u_i}$ with $\mathbf{x_i}$ in φ_i. The variables in $\mathbf{z}'$ are all variables in $\varphi_1 \ldots \varphi_k$ that are not affected from the replacements. Since, there are only finitely many combinations of st-tgds in Σ_{12}, also the constructed set of st-tgds is finite. The union of all constructed sets of st-tgds forms Σ_{13}. The resulting composition is defined as $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$.

It follows immediately by the construction of $\mathcal{M}_{13}$ that if $\langle I_1, I_3 \rangle \in \mathsf{Inst}(\mathcal{M}_{12}) \circ \mathsf{Inst}(\mathcal{M}_{23})$, then also $\langle I_1, I_3 \rangle \models \Sigma_{13}$ holds. Moreover, we have to show for an instance $\langle I_1, I_3 \rangle$ that satisfies Σ_{13}, that also $\langle I_1, I_3 \rangle \in \mathsf{Inst}(\mathcal{M}_{12}) \circ \mathsf{Inst}(\mathcal{M}_{23})$ holds. It suffices to show that $\langle I_2, I_3 \rangle \in \mathsf{Inst}(\mathcal{M}_{23})$, where I_2 is the universal solution resulting from chasing $\langle I_1, \emptyset \rangle$ with Σ_{12}. □

Note that the composition in Example 3.56 illustrates the technique used in the above proof. Fagin et al. [39] showed that if we allow arbitrary st-tgds in the first of the two composed schema mappings, then we can have that the composition is only expressible with infinitely many st-tgds or is not even definable with first-order dependencies. We restrict us to the second case, but both cases imply that the we cannot apply the approach where we solve

the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem with the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$.

Theorem 3.58. [39] *There exist two schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$, where Σ_{12} consist of a single st-tgd and Σ_{23} is a set of full st-tgds, such that the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ cannot be defined by first-order dependencies.*

Proof. *(Sketch)* Consider that $\mathbf{S}_1$ consists of one unary relation symbol, $\mathbf{S}_2$ consists of single binary relation symbol, and $\mathbf{S}_3$ consists of one binary and one unary relation symbol. The dependencies in $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ are defined as follows:

$$\begin{aligned}\Sigma_{12} &= \{D(x) \rightarrow \exists y E(x, y)\} \\ \Sigma_{23} &= \{E(x, y) \rightarrow F(x, y), \\ &\qquad E(x, x) \rightarrow G(x)\}\end{aligned}$$

Assume now the instance $I_1 = \{D(a)\}$ and the instance $I_3 = \{F(a, b)\}$. Notice that in I_3 relation G^{I_3} is empty. Clearly, the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem is for instance $\langle I_1, I_3 \rangle$ satisfied. Let $I_3' = \{F(a, a)\}$, with $G^{I_3} = \emptyset$, be another instance. The CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem is for instance $\langle I_1, I_3' \rangle$ unsatisfied. It can be showed that every st-tgd that satisfies $\langle I_1, I_3 \rangle$ also satisfies $\langle I_1, I_3' \rangle$. □

We have already mentioned that if a composition of two schema mappings with finitely many st-tgds cannot be computed, then the composition query can be solved with the naive approach which has a runtime in NP. Furthermore, [39] showed that for such a case the composition query is also hard for NP.

Theorem 3.59. [39] *There exist two schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$, where Σ_{12} is a set of st-tgds and Σ_{23} is a set of full st-tgds, such that the* CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) *problem is NP-complete.*

Proof. *(Sketch)* It suffices to show NP membership for the case, where Σ_{12} and Σ_{23} are arbitrary st-tgds, and NP hardness for the case, where Σ_{23} contains only full st-tgds. Let $\langle I_1, I_3 \rangle$ be an arbitrary instance over $\langle \mathbf{S}_1, \mathbf{S}_2 \rangle$. NP membership can be shown by guessing an instance I_2 over $\mathbf{S}_2$ and checking in polynomial time if $\langle I_1, I_2 \rangle \models \Sigma_{12}$ and $\langle I_2, I_3 \rangle \models \Sigma_{23}$ holds.

NP hardness is shown by a reduction from the Homomorphism problem to the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem. The Homomorphism problem is a generalization of the 3Sat and the 3Colorability problem, where we ask for two given instances I and J over the same schema, if there is a homomorphism from I to J. □

3.4.2 Second-Order Dependencies

In contrast to first-order dependencies, second-order dependencies are closed under composition, i.e. for every two schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$

and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ defined with st-tgds or SO-tgds, we are able to construct a composition $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \sigma_{13})$ such that σ_{13} is a single SO-tgd. In the proof of Theorem 3.58 two schema mappings are shown, where a composition cannot be defined with first-order dependencies. We show now that for these schema mappings a composition via SO-tgds can be found. This example is adopted from [39].

Example 3.60. Assume three schemata $\mathbf{S}_1$, $\mathbf{S}_2$ and $\mathbf{S}_3$, where $\mathbf{S}_1$ consists of a unary, $\mathbf{S}_2$ of a binary, and $\mathbf{S}_3$ of a unary and a binary relation symbol. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be data exchange settings with the following dependencies:

$$\begin{aligned}\Sigma_{12} &= \{D(x) \rightarrow \exists y E(x,y)\}\\ \Sigma_{23} &= \{E(x,y) \rightarrow F(x,y),\\ &\qquad E(x,x) \rightarrow G(x)\}\end{aligned}$$

The composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ can be defined as schema mapping $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \sigma_{13})$, where σ_{13} is the following SO-tgd:

$$\exists f(\forall x(D(x) \rightarrow F(x, f(x))) \wedge \forall x(D(x) \wedge (x = f(x)) \rightarrow G(x))).$$

□

Fagin et al. [39] proposed the algorithm $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ that computes a composition $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \sigma_{13})$ of two given data exchange settings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \sigma_{23})$. Recall, that SO-tgds are closed under conjunction and therefore every set of SO-tgds can be transformed into a single SO-tgd. Moreover, by Skolemizing every st-tgd can be transformed into an SO-tgd. We do not give a formal specification of the $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ algorithm, but give instead an informal description in the following example.

Example 3.61. Consider the schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \sigma_{23})$. Schema $\mathbf{S}_1$ has two binary relation symbols D and E, schema $\mathbf{S}_2$ has one ternary relation symbol F, and $\mathbf{S}_3$ consists of a unary relation symbol G and a binary relation symbol I. The SO-tgds in $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ are defined as follows.

$$\begin{aligned}\sigma_{12} =& \exists f(\forall x \forall y \forall z(D(x,y) \wedge E(y,z) \rightarrow F(x, f(y), z)))\\ \sigma_{23} =& \forall u \forall v(F(u,v,u) \rightarrow G(v)) \wedge\\ & \exists h(\forall u \forall v \forall w \forall s \forall t(F(u,v,w) \wedge F(s,v,t) \rightarrow I(h(u,s), h(w,t))))\end{aligned}$$

The $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ algorithm is divided in four steps. In the first step we initialize the sets $\mathcal{S}_{12}$ and $\mathcal{S}_{23}$. An SO-tgd consists of a conjunction of implications of the form $\varphi(\mathbf{x}) \rightarrow \bigwedge_{j=1}^{k} R_j(\mathbf{t_j})$, where k is the number of atomic formulas in the conclusion of the implication and $\mathbf{x}$ contains every $\mathbf{t_j}$. The set $\mathcal{S}_{12}$ is defined as the set of formulas $\varphi(\mathbf{x}) \rightarrow R_1(\mathbf{t_1}), \ldots, \varphi(\mathbf{x}) \rightarrow R_k(\mathbf{t_k})$. Set $\mathcal{S}_{23}$ is defined analogously. After the first step $\mathcal{S}_{12}$ and $\mathcal{S}_{23}$ contain

the following elements:

$$\begin{aligned}\mathcal{S}_{12} &= \{D(x,y) \wedge E(y,z) \rightarrow F(x,f(y),z)\}\\ \mathcal{S}_{23} &= \{F(u,v,u) \rightarrow G(v),\\ &\qquad F(u,v,w) \wedge F(s,v,t) \rightarrow I(h(u,s),h(w,t))\}\end{aligned}$$

In the second step we compose $\mathcal{S}_{12}$ with $\mathcal{S}_{23}$. If we have an implication χ in $\mathcal{S}_{23}$, such that the antecedent contains an atomic formula that is not included in any conclusion of an implication of $\mathcal{S}_{12}$, then we remove χ from $\mathcal{S}_{23}$. For all other implications χ of the form $\phi \rightarrow \psi$ in $\mathcal{S}_{23}$, we replace every atomic formula $R(\mathbf{x})$ in ϕ with the antecedent of implication $\varphi \rightarrow R(\mathbf{t})$ from $\mathcal{S}_{12}$. In addition, we add in the antecedent conjunctions of equalities, which link the variables in $\mathbf{x}$ with the terms in $\mathbf{t}$. Should in $\mathcal{S}_{12}$ exist more than one formula with $R(\mathbf{t})$ in the conclusion, then we replace $R(\mathbf{x})$ with the antecedent of each $R(\mathbf{t})$ in a fresh copy of χ. For instance implication $F(u,v,u) \rightarrow G(v)$ is replaced with

$$\chi_1 : D(x,y) \wedge E(y,z) \wedge (v = f(y)) \rightarrow G(v).$$

Moreover, variables with overlapping names occurring in a replacement get renamed. The formula $F(u,v,w) \wedge F(s,v,t) \rightarrow I(h(u,s),h(w,t))$ is transformed by the second step into

$$\begin{aligned}\chi_2 :& D(x_0,y_0) \wedge E(y_0,z_0) \wedge (u = x_0) \wedge (v = f(y_0)) \wedge (w = z_0)\\ &\wedge D(x_1,y_1) \wedge E(y_1,z_1) \wedge (s = x_1) \wedge (v = f(y_1)) \wedge (t = z_1)\\ &\rightarrow I(h(u,s),h(w,t)).\end{aligned}$$

In the third step we simplify $\mathcal{S}_{23}$ by removing superfluous equalities and variables in the implications:

$$\begin{aligned}\chi_1' :& D(x,y) \wedge E(y,z) \rightarrow G(f(y))\\ \chi_2' :& D(x_0,y_0) \wedge E(y_0,z_0) \wedge D(x_1,y_1) \wedge E(y_1,z_1) \wedge (f(y_0) = f(y_1))\\ &\rightarrow I(h(x_0,x_1),h(z_0,z_1))\end{aligned}$$

In step four the resulting SO-tgd $\exists f(\exists x \exists y \exists z \chi_1' \wedge \exists x_0 \exists y_0 \exists z_0 \exists x_1 \exists y_1 \exists z_1 \chi_2')$ is returned. □

The following result by [39], states that $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ indeed computes the correct composition for every pair of data exchange settings defined by SO-tgds.

Theorem 3.62. [39] *Consider two data exchange settings* $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{12})$ *and* $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \sigma_{23})$*, where* σ_{12} *and* σ_{23} *are two SO-tgds. Then, the algorithm* $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ *computes an SO-tgd* σ_{13}*, such that* $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \sigma_{13})$ *is the composition of* $\mathcal{M}_{12}$ *and* $\mathcal{M}_{23}$*.*

The size of the SO-tgd returned by the $\mathsf{Compose}(\mathcal{M}_{12}, \mathcal{M}_{23})$ algorithm can be exponential in the maximum number of atomic formulas appearing in the antecedent of any implication of σ_{23}. Although, we can always compute

a composition, the CompositionQuery($\mathcal{M}_{12}, \mathcal{M}_{23}$) problem still remains NP-complete. This is due the fact that deciding if an instance $\langle I_1, I_2 \rangle$ satisfies an SO-tgd is in contrast to first-order dependencies an NP-complete problem.

3.5 Exchanging Incomplete Data

So far we have always assumed that source instances of a data exchange setting are complete. On the other hand, solutions are in general incomplete. The question then arises, how natural this assumption is and how the existing techniques can be extended such that they can handle incomplete source instances. In this section we discuss an approach based on positive conditional instances that allows data exchange with incomplete source instances, proposed by Arenas et al. [10]. Therefore, we reconsider the topics of the former sections, namely solution building, query answering and composing schema mappings with schema mappings that allow incomplete source instances. We start with the observation that naive instances do not have enough expressive power for representing solutions for incomplete source instances. Therefore we have to formalize the property that a representation system has to fulfill to be able to represent the space of solutions.

Definition 3.63. *Consider a representation system* $(\mathbf{W}, \mathsf{rep})$ *and a class of schema mappings* C*. If for every schema mapping* $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ *in* C *and for every* $\mathcal{U} \in \mathbf{W}$ *over* $\mathbf{S}_1$*, there exists a* $\mathcal{V} \in \mathbf{W}$ *over* $\mathbf{S}_2$ *such that* $\mathsf{rep}(\mathcal{V}) = \bigcup_{I \in \mathsf{rep}(\mathcal{U})} \mathsf{Sol}_{\mathcal{M}}(I)$ *holds, then* $(\mathbf{W}, \mathsf{rep})$ *is called a* strong representation system for C*.*

The notion of strong representation systems was initially proposed for queries [49]. Fagin et al. showed that the representation system formed by naive instances is powerful enough to represent the space of solutions for the class of schema mappings defined by st-tgds [37] and SO-tgds [39] for any complete source instance. The following example shows that we lose this property if we allow the source instance to be incomplete.

Example 3.64. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping, where S_1 consists of a single binary relation symbol D and S_2 consists of a single unary relation symbol E. Σ_{12} is a set of first-order dependencies, defined as:

$$D(x, x) \rightarrow E(x)$$

Assume a naive source instance $I = \{D(a, n)\}$, where n is an element of Var. A solution J for I under $\mathcal{M}$ must be able to express that if in an interpretation of I $n = a$ holds, then $E(a)$ must be a fact of every solution of that interpretation. It is easy to see that for naive instances this is not the case.

Consider a positive conditional instance $I' = \{D(a, n)\}$ with $\rho_D^{I'} : (a, n) \rightarrow$ **true**, where $n \in \mathsf{Var}$. Moreover, let $J' = \{E(n)\}$ with $\rho_E^{J'} : (n) \rightarrow n = a$ be a conditional instance over $\mathbf{S}_2$. It can be shown that J' is indeed a solution for I' under $\mathcal{M}$, i.e. $\mathsf{rep}(J') = \bigcup_{I \in \mathsf{rep}(I')} \mathsf{Sol}_{\mathcal{M}}(I)$ holds. □

From the above example we can conclude that $\{\mathbf{W}_{naive}, \mathsf{rep}_{naive}\}$ is not a strong representation system for the class of schema mappings defined by st-tgds. We have also seen that positive conditional instances can represent the space of solutions for at least a specific data exchange setting. Arenas et al. [10] showed that this result holds also in general. Moreover, positive conditional instances can correctly represent the solutions of schema mappings defined with SO-tgds.

Theorem 3.65. [10] *Positive conditional instances form a strong representation system for the class of data exchange settings where the dependencies are defined either as st-tgds or SO-tgds.*

Clearly, this result holds also for the representation system $\{\mathbf{W}_{cond}, \mathsf{rep}_{cond}\}$. In the literature it was frequently observed that more expressive representation systems lead to higher complexity bounds on problems. Hence, we are interested in a representation system which is expressive enough to be strong, but is still sufficiently simple for allowing efficient algorithms. Arenas et al. [10] showed that the property of being a strong representation system for the class of schema mappings defined by st-tgds or SO-tgds is immediately lost if we apply any further restriction on the local conditions of positive conditional instances. This result gives strong evidence that $\{\mathbf{W}_{pos}, \mathsf{rep}_{pos}\}$ is indeed the favorable representation system for data exchange with incomplete instances. As next step, we have to reconsider the semantics of solutions if incomplete source instances are allowed.

Definition 3.66. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a schema mapping and let $\mathcal{R} = \{\mathbf{W}, \mathsf{rep}\}$ be a representation system. Consider $\mathcal{U}$, $\mathcal{V}$ elements of $\mathbf{W}$ over schema $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. Then, $\mathcal{V}$ is an* $\mathcal{R}$-solution for $\mathcal{U}$ under $\mathcal{M}$ *if $\mathsf{rep}(\mathcal{V}) \subseteq \bigcup_{I \in \mathsf{rep}(\mathcal{U})} \mathsf{Sol}_{\mathcal{M}}(I)$ holds. Moreover, this $\mathcal{R}$-solution is called* universal *if also $\mathsf{rep}(\mathcal{V}) = \bigcup_{I \in \mathsf{rep}(\mathcal{U})} \mathsf{Sol}_{\mathcal{M}}(I)$ holds.*

Several remarks are in order now. If $\mathcal{V}$ is an $\mathcal{R}$-solution for $\mathcal{U}$ under a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ and $J \in \mathsf{rep}(\mathcal{V})$, then there is an $I \in \mathsf{rep}(\mathcal{U})$ such that $\langle I, J \rangle \models \Sigma$ holds. Universal $\mathcal{R}$-solutions allow representing the space of possible solutions with a single element of the representation system. Consequently, the notion of strong representation system demands that for every source instance for a schema mapping a universal $\mathcal{R}$-solution has to exist. The instance J' in Example 3.64 is a universal $\mathcal{R}_{pos}$-solution.

With $\{\mathbf{W}_{pos}, \mathsf{rep}_{pos}\}$ we have found a representation system which allows us to correctly represent solutions of schema mappings specified by st-tgds or SO-tgds. Hence, we have the foundation to discuss the problems of the former sections for data exchange settings with incomplete source instances. We start with the DataExchange($\mathcal{M}$) problem. In Section 3.2 universal solutions have been shown to be suitable for materialization. In the presence of a representation system $\mathcal{R}$ materializing a universal $\mathcal{R}$-solution seems the obvious choice. The following result, proposed in [10], states that the DataExchange($\mathcal{M}$) problem can be solved efficiently for an incomplete source instance.

Theorem 3.67. [10] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$ be a data exchange setting, where Σ is either a set of st-tgds or a single SO-tgd, and let I be a positive conditional instance over $\mathbf{S}_1$. Then, there exists an algorithm that computes a universal $\mathcal{R}_{pos}$-solution for I under $\mathcal{M}$ in polynomial time.*

Proof. *(Sketch)* The algorithm is based on the classical chase technique [17] and can be directly applied to schema mappings specified by st-tgds. The chase procedure provided by [10] differs from the chase considered in Section 3.2.1 in order that it can handle local conditions and relationships between elements from Var in the source instance. Moreover, this algorithm can also be used to solve schema mappings specified by a single SO-tgds. This is due to the fact that an arbitrary schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{12})$, where σ_{12} is an SO-tgd, can be expressed as a finite sequence of schema mappings specified by st-tgds $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12}), \ldots, \mathcal{M}_{k-1k} = (\mathbf{S}_{k-1}, \mathbf{S}_k, \Sigma_{k-1k})$ such that $\mathcal{M} = \mathcal{M}_{12} \circ \ldots \circ \mathcal{M}_{k-1k}$ holds [39]. Then, a universal $\mathcal{R}_{pos}$-solution for a source instance I under $\mathcal{M}$ can be computed by computing iteratively a universal $\mathcal{R}_{pos}$-solution for I_i under $\mathcal{M}_{ii+1}$, where $1 \leq i < k$, $I_1 = I$ and I_i is the universal $\mathcal{R}_{pos}$-solution for I_{i-1} under $\mathcal{M}_{i-1i}$. □

The next discussed problem is QueryEvaluation$(\mathcal{M}, q)$. In order that positive conditional instances can be handled adequately, we have to adapt the notion of certain answers of a target query q with respect to the source instance I under the considered schema mapping $\mathcal{M}$ as follows:

$$\mathsf{certain}_{\mathcal{M}}(q, I) = \bigcap \{q(J) \mid J \text{ is an } \mathcal{R}_{pos}\text{-solution for } I \text{ under } \mathcal{M}\}.$$

In Section 3.3, we achieved polynomial runtime for computing the certain answers with restricting the query language in such a way that queries can be directly evaluated on the universal solutions. This method can also applied on schema mappings with positive conditional instances. Whereas in the case of naive universal solutions the largest applicable class of queries are UCQs, [10] showed that arbitrary first-order queries can be directly computed on universal $\mathcal{R}_{pos}$-solutions.

Theorem 3.68. [10] *Consider a data exchange setting $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is set of st-tgds, a positive conditional source instance I, and an arbitrary first-order query q over $\mathbf{S}_2$. Then, $\mathsf{certain}_{\mathcal{M}}(q, I) = q(J)$ holds for every universal $\mathcal{R}_{pos}$-solutions J for I under $\mathcal{M}$.*

Furthermore, [10] showed that the QueryEvaluation$(\mathcal{M}, q)$ problem can be solved comparably efficiently for positive conditional instances as for the usual case:

Theorem 3.69. [10] *Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a set of st-tgds. Moreover, consider a positive conditional source instance I, and a query q consisting of a union of conjunctive queries with at most one inequality per disjunct over $\mathbf{S}_2$. Then* QueryEvaluation$(\mathcal{M}, q)$ *can be decided in polynomial time.*

The last topic that we consider is composing schema mappings in presence of conditional instances. For this purpose we have to introduce an analog for the notion $\mathsf{Inst}(\mathcal{M})$ that supports positive conditional instances:

Definition 3.70. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping. A* positive conditional mapping generated from $\mathcal{M}$*, denoted as $\mathsf{pc}(\mathcal{M})$, is a set of pairs of a positive conditional source and target instance and is defined as*

$$\{(I_1, I_2) \mid I_2 \text{ is an } \mathcal{R}_{pos}\text{-solution for } I_1 \text{ under } \mathcal{M}\}.$$

Notice that for a given schema mapping $\mathcal{M}$ and an arbitrary element of $\mathsf{Inst}(\mathcal{M})$, say $\langle I, J\rangle$, a pair of positive conditional instances (I', J') exists such that $(I', J') \in \mathsf{pc}(\mathcal{M})$. We know from Section 3.4.2 that SO-tgds are closed under composition, i.e. for two schema mappings $\mathcal{M}_{12}$, $\mathcal{M}_{23}$ specified by SO-tgds, there is a third schema mapping specified by SO-tgds such that $\mathcal{M}_{13} = \mathcal{M}_{12} \circ \mathcal{M}_{23}$ holds. Arenas et al. [10] showed that SO-tgds remain still closed under composition if we allow positive conditional instances.

Theorem 3.71. [10] *Consider two schema mappings $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \sigma_{23})$, where σ_{12} and σ_{23} are two SO-tgds. Then there exists an SO-tgd σ_{13}, such that for the schema mapping $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_2, \sigma_{13})$ the equality $\mathsf{pc}(\mathcal{M}_{13}) = \mathsf{pc}(\mathcal{M}_{12}) \circ \mathsf{pc}(\mathcal{M}_{23})$ holds.*

CHAPTER 4

Knowledge Exchange

Knowledge exchange is a form of data exchange where data is exchanged between *knowledge bases*. Basically, a knowledge base consists of a set of facts and a set of rules over these facts that allow to deduce new facts. Facts in knowledge bases are usually termed as *explicit knowledge* and are given in the context of this thesis by relational instances. Accordingly, the rules assigned to a knowledge base are called *implicit knowledge* and are given by a set of logical formulae. An example for knowledge bases are Datalog programs, where the extensional database represents the explicit knowledge and the intentional database expresses the implicit knowledge. Another example for knowledge bases are description logic specifications, where ABox and TBox correspond to explicit and implicit knowledge.

Clearly relational instances, introduced in Section 2.1, can be seen as knowledge bases with no implicit knowledge. Thus, knowledge bases are a generalization of relational instances. A natural question is whether some of the techniques from relational data exchange can be adapted to data exchange on knowledge bases. It turns out that the framework for representation systems from [10], discussed in Section 3.5, can be applied to knowledge bases. In particular, Arenas, Pérez and Reutter [10] laid the foundations of exchanging explicit knowledge as well as implicit knowledge via schema mappings. Knowledge exchange was further investigated in the context of description logics in [7, 8, 24].

This chapter is entirely based on the results of Arenas, Pérez and Reutter [10] and is structured as follows. In Section 4.1 the notions of knowledge bases and knowledge exchange are specified. In addition an example is given that is referred throughout the chapter. The most fundamental problem in knowledge exchange is to materialize a target knowledge base for a source knowledge base under a schema mapping. Clearly, this requires being able to check, whether a knowledge base truly represents another knowledge base under a given schema mapping. It turns out that this problem is in general undecidable and is further investigated in Section 4.2. In Section 4.3 and Section 4.4 two techniques for materializing a knowledge base with respect

to a schema mapping and a source knowledge base are illustrated. In Section 4.3 universal knowledge base solutions are introduced, which are the analog of the universal solutions from Section 3.5. Minimal knowledge base solutions, discussed in Section 4.4, are a relaxation of universal knowledge base solutions and allow much more expressive implicit knowledge.

4.1 Knowledge Bases and Knowledge Exchange

In this section the formalism of knowledge bases is introduced. In addition, the notion of solution in the context of exchanging knowledge between two knowledge bases is defined. As already mentioned, knowledge bases distinguish between *explicit data* and *implicit data*. To be able to reuse many of the concepts discussed so far, we use in this chapter relational data for the explicit data. Logical formulae are used to express implicit data.

Definition 4.1. *A* knowledge base *over a schema* $\mathbf{S}$ *is a pair* (I, Σ)*, where* I *is an instance over* $\mathbf{S}$ *and* Σ *is a set of logical formulae over* $\mathbf{S}$*. The set of possible* models *of* (I, Σ)*, denoted as* $\mathsf{Mod}(I, \Sigma)$*, is defined as*

$$\mathsf{Mod}(I, \Sigma) = \{K \in \mathsf{Inst}(\mathbf{S}) \mid K \supseteq I \text{ and } K \models \Sigma\}.$$

A knowledge base (I, Σ) can be seen as an representative for the set $\mathsf{Mod}(I, \Sigma)$. On the other hand, each element of $\mathsf{Mod}(I, \Sigma)$ can be seen as an interpretation of (I, Σ). Let $\mathbf{K}$ be the set of all possible knowledge bases over all possible relational schemas, then $\mathcal{R} = (\mathbf{K}, \mathsf{Mod})$ forms a representation system for knowledge bases. Hence, we are allowed to apply the concept of solution specified in Definition 3.66 for knowledge bases.

Definition 4.2. *Let* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ *be a schema mapping, and let* (I, Σ_1) *and* (J, Σ_2) *be two knowledge bases over* $\mathbf{S}_1$ *and* $\mathbf{S}_2$*, respectively. We say* (J, Σ_2) *is a* knowledge base solution for (I, Σ_1) under $\mathcal{M}$*, if* $\mathsf{Mod}(J, \Sigma_2) \subseteq \bigcup_{K \in \mathsf{Mod}(I, \Sigma_1)} \mathsf{Sol}_{\mathcal{M}}(K)$ *holds, i.e. for every instance* $L \in \mathsf{Mod}(J, \Sigma_2)$ *there is a* $K \in \mathsf{Mod}(I, \Sigma_1)$ *such that* $\langle K, L \rangle \in \mathsf{Inst}(\mathcal{M})$*. The set of all knowledge base solutions for* (I, Σ_1) *under* $\mathcal{M}$ *is denoted by* $\mathsf{Sol}_{\mathcal{M}}(I, \Sigma_1)$*.*

Notice, that this definition corresponds to Definition 3.66, where the notion of solution for incomplete source instances under a schema mapping is introduced. We say in the above case that (I, Σ_1) is a *source knowledge base* and (J, Σ_2) is a *target knowledge base*. In addition, we say that I is the *source explicit knowledge* and that J is the *target explicit knowledge*. Analogously, Σ_1 refers to the *source implicit knowledge* and Σ_2 refers to the *target implicit knowledge*.

The most crucial problem concerning knowledge exchange is the problem of constructing a knowledge base solution for a source knowledge base under a given schema mapping. More formally:

PROBLEM:	KNOWLEDGEEXCHANGE($\mathcal{M}$)
INPUT:	source knowledge base (I, Σ_1)
QUESTION:	find (J, Σ_2) s.t. $(J, \Sigma_2) \in \mathsf{Sol}_{\mathcal{M}}(I, \Sigma_1)$

We illustrate the concepts introduced so far with the following example. Notice, that we are using tgds from Definition 3.3 as implicit knowledge for the considered knowledge bases.

Example 4.3. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping, where schema $\mathbf{S}_1$ consists of two binary relation symbols D, E and one ternary relation symbol F. Schema $\mathbf{S}_2$ consists of one binary relation symbol D' and one ternary relation symbol F'. The dependencies of Σ_{12} are defined as follows:

$$\begin{aligned}\Sigma_{12} = \{&D(x,y) \rightarrow D'(x,y),\\ &F(x,y,z) \rightarrow F'(x,y,z)\}\end{aligned}$$

Consider a knowledge base (I, Σ_1) over $\mathbf{S}_1$, where I consists of the tuples from $\{D(a,b), D(b,c), D(b,d), E(a,b), E(b,c), E(c,d)\}$ and Σ_1 is specified as follows:

$$\begin{aligned}\Sigma_1 = \{&D(x,y) \wedge D(y,z) \rightarrow F(x,y,z),\\ &E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)\}\end{aligned}$$

Notice that the logical formulae in Σ_1 are full tgds as specified in Definition 3.3. One can infer from I and Σ_1 the tuples $\{F(a,b,c), F(a,b,d), F(b,c,d)\}$. We are now interested in defining a second knowledge base (J, Σ_2) this time over $\mathbf{S}_2$ such that (J, Σ_2) is a knowledge base solution for (I, Σ_1) under $\mathcal{M}$. An obvious approach would be to make all the data in (I, Σ_1) explicit and construct a solution with the techniques from Section 3.2. In this case J contains the tuples in $\{D'(a,b), D'(b,c), D'(b,d), F'(a,b,c), F'(a,b,d), F'(b,c,d)\}$ and Σ_2 is the empty set. A more economical approach is to keep as much data implicit as possible. We know from Σ_1 and Σ_{12} that some of the facts in relation F'^J can be derived from the tuples in D'^J if we set Σ_2' to

$$\{D'(x,y) \wedge D'(y,z) \rightarrow F'(x,y,z)\}.$$

The explicit knowledge in J' has to consist of the facts in $\{D'(a,b), D'(b,c), D'(b,d), F'(b,c,d)\}$. Note that the tuple $F'(b,c,d)$ must still be contained in J', since it is inferred from $E(b,c)$ and $E(c,d)$. Moreover, it can be shown that (J', Σ_2') is a knowledge base solution for (I, Σ_1) under $\mathcal{M}$. □

4.2 Recognizing Knowledge Base Solutions

As first step towards an algorithm that solves KnowledgeExchange($\mathcal{M}$) is to address the problem of deciding whether a target knowledge base is a knowledge base solution for the a source knowledge base under a schema mapping. It turns out that this problem is by no means trivial and therefore needs closer consideration.

Problem:	SolutionTesting($\mathcal{M}$)
Input:	knowledge bases (I, Σ_1) and (J, Σ_2)
Question:	is $(J, \Sigma_2) \in \mathsf{Sol}_{\mathcal{M}}(I, \Sigma_1)$?

Arenas et al. [10] showed that even if we restrict the language expressing the implicit knowledge of the knowledge bases to tgds and the dependencies of the schema mappings to st-tgds, the SolutionTesting($\mathcal{M}$) problem remains undecidable.

Theorem 4.4. [10] *There exists a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} consists of a set of st-tgds, and two knowledge bases with implicit knowledge defined by tgds for which the* SolutionTesting($\mathcal{M}$) *problem is undecidable.*

Proof. *(Sketch)* Theorem 4.4 can be shown, like Theorem 3.5, by a reduction from the undecidable problem Embedding [46, 34]. In Embedding we ask for a finite set B and a partial associative function $g : B \times B \to B$, whether there exists a set $B \subseteq A$ and a total associative function $f : A \times A \to A$ that extends g. Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping with schemata $\mathbf{S}_1 = \langle C, E, N, G, F \rangle$ and $\mathbf{S}_2 = \langle F' \rangle$, and the single dependency $\Sigma_{12} = \{F(x) \to F'(x)\}$. Essentially, an instance of SolutionTesting($\mathcal{M}$), consisting of source knowledge base (I, Σ_1) and target knowledge base (J, Σ_2), is constructed from an arbitrary partial algebra $\mathbf{B} = (B, g)$ as follows. For each $g(b_i, b_j) = b_k$ exists in relation G^I a tuple $G(b_i, b_j, b_k)$. The relations C^I, E^I and N^I are binary relations that allow to determine equivalence of two elements of B. The unary relation F^I remains empty. The dependencies in Σ_1 are defined in such a way that there exists an instance $K \in \mathsf{Mod}(I, \Sigma_1)$ with $F^K = \emptyset$ if and only if $\mathbf{B} = (B, g)$ is embeddable in a finite semigroup $\mathbf{A} = (A, f)$. The target knowledge base (J, Σ_2) consists of $F'^J = \emptyset$ and $\Sigma_2 = \emptyset$. Therefore, the target knowledge base (J, Σ_2) is a knowledge base solution for (I, Σ_1) under $\mathcal{M}$ if and only if $\mathbf{B} = (B, g)$ can be embedded in $\mathbf{A} = (A, f)$. □

Note that the previous result also holds for the more restrictive case where the dependencies in the schema mapping are restricted to full st-tgds, i.e. the st-tgds do not contain existential quantified variables. Hence, Arenas et al. [10] focused on restricting the implicit knowledge of the knowledge bases to obtain decidability.

Theorem 4.5. [10] *Consider a given schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a set of st-tgds. Moreover, let (I, Σ_1) and (J, Σ_2) be a source and a target knowledge base, where Σ_1 and Σ_2 are two sets of full tgds. Then* SolutionTesting($\mathcal{M}$) *is $\Delta_2^P[O(\log n)]$-complete.*

Proof. *(Sketch)* Recall that $\Delta_2^P[O(\log n)]$ is the class of decision problems that can be decided in polynomial time by a deterministic Turing machine that is allowed to call an NP oracle logarithmically many times. Membership is shown by an algorithm that runs in $\mathrm{P}^{||\mathrm{NP}}$, i.e. the algorithm solves the SolutionTesting($\mathcal{M}$) problem in polynomial time and is allowed to call once polynomially many NP oracles in parallel. Buss and Hay [25] and Wagner [67] showed that the class $\mathrm{P}^{||\mathrm{NP}}$ is equivalent to the class $\Delta_2^P[O(\log n)]$.

Let K be the result of chasing I with Σ_1. Analogously, let L be the result of chasing J with Σ_2. It can be shown that (J, Σ_2) is a knowledge base solution

for (I, Σ_1) under $\mathcal{M}$ if and only if $(K, L) \models \Sigma_{12}$. Moreover, checking whether a tuple $R(\mathbf{a})$ is included in K can be done in NP by first guessing a chase sequence of I with Σ_1 and validating each chase step in polynomial time. The algorithm starts by creating for each relation symbol R_1 in S_1 all possible tuples $R_1(\mathbf{a})$. Equally, for each relation symbol R_2 in S_2 all possible tuples $R_2(\mathbf{a})$ are generated. Since S_1 and S_2 are both considered as fixed, the number of created tuples is bounded by the number of elements in the domain and the sum of the arities of the relation symbols. The instances K and L are computed by asking polynomially many NP oracles in parallel, whether the created tuples are included in the corresponding result of chasing I with Σ_1 and of chasing J with Σ_2. Lastly, we check whether $(K, L) \models \Sigma_{12}$ holds.

$\Delta_2^P[O(\log n)]$-hardness is shown by a reduction from MaxOddClique, which was shown in [66] to be a $\Delta_2^P[O(\log n)]$ hard problem. MaxOddClique asks whether the size of the maximum clique of an undirected graph is odd. The fixed schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ consists of schemata $S_1 = \langle Succ, First, Clique, E\rangle$ and $S_2 = \langle Succ', First', Clique', E'\rangle$, and the dependency

$$\Sigma_{12} = \{Clique(x) \wedge Succ(x, y) \rightarrow Clique'(y)\}.$$

The intuition of the reduction is as follows. The relations E and E' each store a copy of the undirected input graph. The $Succ^I$ relation contains the tuples $Succ(i, i{+}1)$ for all $i \in \{1 \ldots n\}$, where n is the number of nodes in the input graph. Every interpretation $\mathsf{Mod}(I, \Sigma_1)$ contains the tuple $Clique(l)$ if and only if the input graph contains a clique of size l, where l is even. Accordingly, every interpretation $\mathsf{Mod}(J, \Sigma_2)$ contains the tuple $Clique'(m)$ iff the input graph contains a clique of size m, where m is odd. Therefore, (J, Σ_2) is a knowledge base solution for (I, Σ_1) if and only if the input graph has a maximum clique of odd size. □

Due to the above result in the rest of this chapter we will restrict ourselves to knowledge bases defined by full tgds. The complexity of the SolutionTesting($\mathcal{M}$) problem can be further lowered if the source implicit knowledge or the target implicit knowledge is assumed to be fixed. Therefore we introduce three variants of the SolutionTesting($\mathcal{M}$) problem where either the source implicit knowledge, the target implicit knowledge, or the source and target implicit knowledge are fixed. In the SolutionTesting($\mathcal{M}, \Sigma_2$) problem for instance, a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and a set of logical sentences Σ_2 over $\mathbf{S}_2$ is given. The problem asks for a source knowledge base (I, Σ_1) and a target instance J, whether (J, Σ_2) is a knowledge base solution for (I, Σ_1) under $\mathcal{M}$. The two other problems SolutionTesting($\mathcal{M}, \Sigma_1$) and SolutionTesting($\mathcal{M}, \Sigma_1, \Sigma_2$) are defined analogously. Arenas et al. [10] showed the following complexity bounds:

Theorem 4.6. [10] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping where Σ_{12} is a set of st-tgds. Moreover, let (I, Σ_1) and (J, Σ_2) be knowledge base defined by full tgds over $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively.*

(1) If Σ_1 and Σ_2 are fixed, then the SolutionTesting($\mathcal{M}, \Sigma_1, \Sigma_2$) *problem is in PTIME.*

(2) If Σ_1 is fixed, then SolutionTesting$(\mathcal{M}, \Sigma_1)$ *is NP-complete.*

(3) If Σ_2 is fixed, then SolutionTesting$(\mathcal{M}, \Sigma_2)$ *is coNP-complete.*

Proof. (Sketch) Assume that K is the result of chasing I with Σ_1, and that L is the result of chasing J with Σ_2. Arenas et al. [10] showed that (J, Σ_2) is a knowledge base solution for (I, Σ_1) under $\mathcal{M}$ if and only if $\langle K, L\rangle \models \Sigma_{12}$ holds.

If Σ_1 and Σ_2 are fixed, then the instances K and L can be computed in polynomial time. Moreover, also $\langle K, L\rangle \models \Sigma_{12}$ can be checked in polynomial time. It follows that SolutionTesting$(\mathcal{M}, \Sigma_1, \Sigma_2)$ can be decided in polynomial time.

We discuss now NP membership of SolutionTesting$(\mathcal{M}, \Sigma_1)$. We start by computing K. Since Σ_1 is fixed, K can be computed with the chase procedure in polynomial time. Afterwards, for every st-tgds in Σ_{12} of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ and every tuple $\mathbf{a}$ such that $\varphi(\mathbf{a})$ is satisfied by K, a chase sequence of J with Σ_2 that satisfies $\psi(\mathbf{a}, \mathbf{b})$ is guessed. Lastly, we check in polynomial time if the guessed chase sequences are valid. NP hardness of SolutionTesting$(\mathcal{M}, \Sigma_1)$ can be shown by a reduction from 3-Colorability.

In the coNP membership proof of SolutionTesting$(\mathcal{M}, \Sigma_2)$ we show that (J, Σ_2) is not a knowledge base solution for (I, Σ_1), i.e. $\langle K, L\rangle \not\models \Sigma_{12}$ holds. Since Σ_2 is fixed, we can compute L in polynomial time. We start by guessing a tuple $\mathbf{a}$, a chase sequence of I with Σ_1, and an st-tgds $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi(\mathbf{x}, \mathbf{y}))$ in Σ_{12}. Afterwards we validate the chase sequence and check whether $\varphi(\mathbf{a})$ is satisfied by the last element of the chase sequence and that there exists no tuple $\mathbf{b}$ such that L satisfies $\psi(\mathbf{a}, \mathbf{b})$. A reduction from the complement of 3-Colorability can be used to show coNP hardness of SolutionTesting$(\mathcal{M}, \Sigma_2)$. □

4.3 Universal Knowledge Base Solutions

We are now interested in an algorithm that solves the KnowledgeExchange$(\mathcal{M})$ problem, i.e. an algorithm that materializes a knowledge base solution for a source knowledge base (I, Σ_1) under a schema mapping $\mathcal{M}$. Due to the Definition 4.2, infinitely many knowledge base solutions for (I, Σ_1) under $\mathcal{M}$ exist. A reasonable question is which of those solutions should be materialized. It seems natural to materialize the knowledge base solution which models represents the models of all other knowledge base solutions. Such solutions are the analog to the universal solutions in the relational scenario.

Definition 4.7. *Consider a given schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, and two knowledge bases (I, Σ_1) and (J, Σ_2) over $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. Knowledge base (J, Σ_2) is a* universal knowledge base solution for (I, Σ_1) under $\mathcal{M}$ *if $\mathsf{Mod}(J, \Sigma_2) = \bigcup_{K \in \mathsf{Mod}(I, \Sigma_1)} \mathsf{Sol}_{\mathcal{M}}(K)$ holds.*

Note that the solution $(J, \emptyset)$ from Example 4.3 is a universal knowledge base solution. Moreover, it follows immediately from the above definition that universal knowledge base solutions can be used to compute the certain answers of arbitrary first order queries. Besides, the problem of finding such a universal knowledge base solution is a more restrictive case of the KnowledgeExchange($\mathcal{M}$) problem and is defined as follows:

Problem:	UniversalKnowledgeBaseSolution($\mathcal{M}$)
Input:	source knowledge base (I, Σ_1)
Question:	find (J, Σ_2) s.t. $\mathsf{Mod}(J, \Sigma_2) = \mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))$

We show in the following theorem, which was proposed in [10], that for source knowledge bases defined by full tgds and schema mappings defined by full st-tgds, a universal knowledge base solution always exists and can be computed by the chase procedure introduced in Section 3.2.1.

Theorem 4.8. [10] *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping, where Σ_{12} is a set of full st-tgds. Moreover, let (I, Σ_1) be a knowledge base over $\mathbf{S}_1$, where Σ_1 is a set of full tgds. If J is the resulting instance of chasing I with Σ_1 and subsequently with Σ_{12}, then the knowledge base $(J, \emptyset)$ is a universal knowledge base solution for (I, Σ_1) under $\mathcal{M}$.*

Proof. We have to show that $\mathsf{Mod}(J, \emptyset) = \bigcup_{K \in \mathsf{Mod}(I, \Sigma_1)} \mathsf{Sol}_{\mathcal{M}}(K)$ holds.

By the definition of the chase we have that $J \in \mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))$. Moreover, $J \in \mathsf{Mod}(J, \emptyset)$ and for every $J' \in \mathsf{Mod}(J, \emptyset)$ we have that $J \subseteq J'$. It follows that $\mathsf{Mod}(J, \emptyset) \subseteq \bigcup_{K \in \mathsf{Mod}(I, \Sigma_1)} \mathsf{Sol}_{\mathcal{M}}(K)$.

Assume that there is instance L such that $L \in \mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))$ and $L \notin \mathsf{Mod}(J, \emptyset)$. We know that for all $J' \in \mathsf{Mod}(J, \emptyset)$ it holds that $J \subseteq J'$. This implies that there must exist a tuple $t \in J$ such that $t \notin L$. According to Definition 3.22 tuples are added during a chase step if and only if they are needed to satisfy a dependency. We conclude that without t, a dependency in Σ_1 or Σ_{12} is violated and thus $L \notin \mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))$, which is a contradiction. Hence, we have also that $\mathsf{Mod}(J, \emptyset) \supseteq \bigcup_{K \in \mathsf{Mod}(I, \Sigma_1)} \mathsf{Sol}_{\mathcal{M}}(K)$. □

In addition to the former result Arenas et al. [10] showed that the UniversalKnowledgeBaseSolution($\mathcal{M}$) problem can be solved in exponential time.

Theorem 4.9. [10] *Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and source knowledge base (I, Σ_1), where Σ_{12} is a set of full st-tgds and Σ_1 is a set of full tgds. There exists an algorithm that computes a polynomial-size universal knowledge base solution for (I, Σ_1) under $\mathcal{M}$ in exponential time.*

Proof. Let $k_{\mathbf{S}_1}$ and $k_{\mathbf{S}_2}$ be the sum of the arities of all relation symbols in $\mathbf{S}_1$ and $\mathbf{S}_2$, respectively. The active domain of I is denoted with $\mathbf{D_a}$. Chasing I with Σ_1 results in at most ${\mathbf{D_a}}^{k_{\mathbf{S}_1}}$ tuples of the maximum size of $k_{\mathbf{S}_1}$. Accordingly, the outcome of chasing the result of the previous chase with Σ_{12} are at most $({\mathbf{D_a}}^{k_{\mathbf{S}_1}})^{k_{\mathbf{S}_2}}$ tuples with the maximum size $k_{\mathbf{S}_2}$. □

We have shown how to produce universal knowledge base solutions with the empty set as implicit knowledge. However, with full tgds in the target implicit knowledge the amount of tuples in the target explicit knowledge could be substantially reduced. The question is natural if there are universal knowledge base solutions that reduce the size of their explicit knowledge by using full tgds as implicit knowledge. The following results states that the explicit knowledge of all universal knowledge base solutions are equivalent to another. Thus, the universal knowledge base solutions produced by applying the chase procedure first on the source implicit knowledge and then on the dependencies are optimal under the viewpoint of the size of the explicit and the implicit knowledge.

Theorem 4.10. *Let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping and let (I, Σ_1) be a source knowledge base, where Σ_{12} is a set of full st-tgds and Σ_1 is a set of full tgds. Consider two arbitrary universal knowledge base solutions (J, Σ_2) and $(J', \emptyset)$ with Σ_2, a set of full tgds, then we have that $J = J'$.*

Proof. We know from Definition 4.7 that $\mathsf{Mod}(J', \emptyset) = \mathsf{Mod}(J, \Sigma_2)$. Moreover, it is easy to see that $J' \in \mathsf{Mod}(J', \emptyset)$ and for every $K \in \mathsf{Mod}(J', \emptyset)$ we have that $K \supseteq J'$. We have to show that $J = J'$. Assume the contrary, $J \neq J'$.

Since the full tgds in Σ_2 can neither eliminate nor modify tuples, it cannot be that (J, Σ_2) is a universal knowledge base solution and $J \supset J'$.

Assume that $J \subset J'$. It follows that there is at least one full tgd $\tau : \forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \psi(\mathbf{x}))$ in Σ_2 which is unsatisfied, i.e. there is a homomorphism h from $\varphi(\mathbf{x})$ to J, but there is no extension to h from $\varphi(\mathbf{x}) \wedge \psi(\mathbf{x})$ to J. Let $\varphi(\mathbf{x})$ be of the form $R_1(\mathbf{x}_1) \wedge \ldots \wedge R_n(\mathbf{x}_n)$, where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbf{x}$. Moreover, let X be a set $\{R_1(\mathbf{a}_1), \ldots, R_n(\mathbf{a}_n)\}$ of tuples, such that values in $\mathbf{a}_1, \ldots, \mathbf{a}_n$ do not occur in J'. By definition $J' \cup X \in \mathsf{Mod}(J', \emptyset)$. However, dependency τ is unsatisfied in $J' \cup X$ and thus $J' \cup X \notin \mathsf{Mod}(J, \Sigma_2)$, which contradicts with the fact that (J, Σ_2) is a universal knowledge base solution. □

4.4 Minimal Knowledge Base Solutions

In the previous section universal knowledge base solutions were discussed. It has turned out that the universality property does not allow the efficient use of implicit knowledge. In this section minimal knowledge base solutions are introduced which relax the notion of universal knowledge base solutions in a way that allows implicit knowledge to be more expressive. We start with two preliminary definitions.

Definition 4.11. *Let X be a set of instances over schema $\mathbf{S}$, and let K be an arbitrary instance in X. The set X is called* closed-up *if for all instances $K' \supseteq K$ also $K' \in X$ holds.*

Definition 4.12. *Consider a set of instances X over schema $\mathbf{S}$. The set of* minimal instances *is defined as*

$$\mathsf{Min}(X) = \{K \in X \mid \textit{there is no } K' \in X \textit{ s.t. } K' \subset K\}.$$

Notice, that $\mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))$ is a closed up set for every source knowledge base (I, Σ_1) and every schema mapping defined by st-tgds and SO-tgds. Furthermore, a closed-up set of instances can be characterized by its minimal instances, i.e. if X and Y are two sets of instances over a schema $\mathbf{S}$, then $X = Y$ if and only if $\mathsf{Min}(X) = \mathsf{Min}(Y)$. On the basis of Definition 4.11 and Definition 4.12, we are now able to give a relaxation of the notion of universal knowledge base solutions from Definition 4.7.

Definition 4.13. *Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and a source knowledge base (I, Σ_1). A target knowledge base (J, Σ_2) is a* minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$ *if and only if*

$$\mathsf{Min}(\mathsf{Mod}(J, \Sigma_2)) = \mathsf{Min}(\mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1))).$$

Several remarks are in order now. Based on the definition of the chase we have that chasing I with Σ results in an element of $\mathsf{Min}(\mathsf{Mod}(I, \Sigma))$ for an arbitrary knowledge base (I, Σ) defined by full tgds. In addition, Arenas et al. [10] showed that an element of $\mathsf{Min}(\mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1)))$ can be computed by chasing I with Σ_1 and then with Σ_{12}. It can be easily seen, that every universal knowledge base solution is also a minimal knowledge base solution, but not every minimal knowledge base solution is a universal knowledge base solution. The knowledge bases (J, Σ_2) and (J', Σ_2') from Example 4.3 are both an example for minimal knowledge base solutions, while (J, Σ_2) is also a universal knowledge base. The problem of finding a minimal knowledge based solution for a given schema mapping and a source knowledge base is defined as follows:

PROBLEM:	MINIMALKNOWLEDGEBASESOLUTION($\mathcal{M}$)
INPUT:	source knowledge base (I, Σ_1)
QUESTION:	find (J, Σ_2) s.t. $\mathsf{Min}(\mathsf{Mod}(J, \Sigma_2)) = \mathsf{Min}(\mathsf{Sol}_{\mathcal{M}}(\mathsf{Mod}(I, \Sigma_1)))$

Clearly, with the technique described in Section 4.3 also the MINIMALKNOWLEDGEBASESOLUTION($\mathcal{M}$) problem can be solved. Though, we are interested in a method that produces more sophisticated minimal knowledge base solutions, i.e. an algorithm that computes minimal knowledge base solutions with as much implicit knowledge as possible. Moreover, the implicit knowledge of a computed minimal knowledge base solution should depend only on the corresponding source implicit knowledge and the schema mapping and not on the source explicit knowledge. Safe sets of dependencies are defined as follows.

Definition 4.14. *Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ defined by full st-tgds and a set Σ_1 of full tgds over $\mathbf{S}_1$. We say a set Σ_2 of full*

tgds over $\mathbf{S}_2$ *is* safe *for* Σ_1 *and* $\mathcal{M}$*, if for every source instance* I *exists a target instance* J*, such that* (J, Σ_2) *is a minimal knowledge base solution for* (I, Σ_1) *under* $\mathcal{M}$*.*

In the subsequent two sections a technique is discussed which produces minimal knowledge base solutions with safe and meaningful implicit knowledge. This technique was initially proposed by Arenas, Pérez and Reutter [10] and basically consists of the application of two algorithms followed by the chase. In the first algorithm, discussed in Section 4.4.1, the implicit knowledge of a minimal knowledge based solution is materialized. The second algorithm, illustrated in Section 4.4.2, allows in combination with the chase technique to compute an explicit knowledge, such that the materialized explicit and implicit knowledge form together a minimal knowledge base solution.

4.4.1 Implicit Knowledge Computation

As already mentioned, for a schema mapping $\mathcal{M}$ and a source knowledge base (I, Σ_1) defined by full st-tgds and full tgds respectively, the result of solving the MinimalKnowledgeBaseSolution($\mathcal{M}$) problem is a minimal knowledge base solution (J, Σ_2). Arenas et al. [10] showed that there is an algorithm which produces a target implicit knowledge Σ_2, such that every other safe set for Σ_1 and $\mathcal{M}$ is implied by Σ_2. The generated target knowledge is a safe set of second-order logic sentences that cannot be expressed in first-order logic. In this section we show the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ algorithm, defined in [10], that produces a safe set of full tgds. Although this safe set of tgds is not optimal, it is nontrivial and should suffice in practice. We illustrate this algorithm by first discussing the techniques to solve the subproblems and show then how they are combined to produce a target implicit knowledge.

An *unfolding* of a set Σ of full tgds over a schema $\mathbf{S}$ is a set Σ^+ of full tgds with a single atomic formula in the conclusion. For every $I \in \mathsf{Inst}(\mathbf{S}_1)$ and every tuple $R(\mathbf{a})$ created by the chase of I with Σ, there exists a formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow R(\mathbf{x}))$ in Σ^+, such that $\varphi(\mathbf{a})$ is satisfied in I. We denote with $\mathsf{Unfold}(\Sigma)$ a procedure that computes the unfolding of a set Σ of full tgds. Such an unfolding Σ^+ is finite if the set Σ is *acyclic*, i.e. there exists a numbering of the relation symbols used in Σ, such that whenever a relation symbol R is in the antecedent and relation symbol S is in the conclusion of a full tgd in Σ, we have that $f(R) < f(S)$. Example 4.15 illustrates the notion of acyclicty of full tgds and the unfolding of a set of full tgds.

Example 4.15. Consider a schema $\mathbf{S}$ consisting of one binary relation symbol R and two unary relation symbols S and T. Let Σ be a set of full tgds defined as:

$$\{S(x) \rightarrow T(x)$$
$$T(x) \rightarrow S(x)\}$$

According to the definition of acyclicity, $f(R) < f(S)$ as well as $f(S) < f(R)$ must hold. It is easy to see that there is no valid numbering such that both

inequalities hold and thus Σ is cyclic. Consider now a second set of full tgds:

$$\begin{aligned}\Sigma' = \{&R(x,y) \rightarrow S(x) \wedge S(y)\\ &S(x) \rightarrow T(x)\}\end{aligned}$$

Let $f(R) = 1$, $f(S) = 2$ and $f(T) = 3$ be a numbering of the relation symbols in Σ'. Since $f(R) < f(S) < f(T)$ holds, we have that Σ' is acyclic. The unfolding of Σ' is:

$$\begin{aligned}\Sigma^+ = \{&R(x,y) \rightarrow S(x)\\ &R(x,y) \rightarrow S(y)\\ &S(x) \rightarrow T(x)\\ &R(x,y) \rightarrow T(x)\\ &R(x,y) \rightarrow T(y)\}\end{aligned}$$

□

Consider a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$. A query q over $\mathbf{S}_1$ is *target rewritable under* $\mathcal{M}$, if for every $I \in \mathsf{Inst}(\mathbf{S}_1)$ there exists a query q' over $\mathbf{S}_2$ such that $q(I) = \mathsf{certain}_{\mathcal{M}}(q', I)$. In such a case is q' the *target rewriting for* q *under* $\mathcal{M}$. In other words we ask if enough information is preserved by a schema mapping such that the query can be answered using the target knowledge. It is shown in [11] that the problem of deciding whether a conjunctive query is target rewritable under a schema mapping defined by full tgds is in coNEXPTIME. Furthermore, [11] states that there is a procedure $\mathsf{TargetRewriting}(\mathcal{M}, \mathsf{q})$ that computes a target rewriting for q and $\mathcal{M}$ in UCQ$^{\neq}$, or fails if q is not target rewritable under $\mathcal{M}$.

A further subtask in the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ algorithm is the $\mathsf{ComposeFull}(\mathcal{M}_{12}, \mathcal{M}_{23})$ procedure, which computes for the schema mappings defined by full st-tgds $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$ the set Σ_{13} of full st-tgds, such that $\mathcal{M}_{13} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma_{13})$ is, according to Definition 3.55, the composition of $\mathcal{M}_{12}$ and $\mathcal{M}_{23}$. The procedure $\mathsf{ComposeFull}(\mathcal{M}_1, \mathcal{M}_2)$ was introduced by Fagin et al. [39] and we already discussed a slightly more general version in the proof of Theorem 3.57. The algorithm to compute the target implicit knowledge for a source knowledge base and a schema mapping is defined as follows:

Algorithm: $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$

Input : An acylic set Σ_1 of full tgds over schema $\mathbf{S}_1$ and a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a set of full st-tgds.

Output: A safe set Σ_2 for Σ_1 and $\mathcal{M}$ of full tgds with inequalities.

1 $\Sigma_1^+ \longleftarrow \mathsf{Unfold}(\Sigma_1)$;
2 $\Sigma_{21} \longleftarrow \emptyset$;
3 **forall the** $(\varphi(\mathbf{x}) \to R(\mathbf{x})) \in \Sigma_1^+$ **do**
4 **if** $\varphi(\mathbf{x})$ *is target rewritable* **then**
5 $(\gamma_1(\mathbf{x}) \vee \ldots \vee \gamma_n(\mathbf{x})) \longleftarrow \mathsf{TargetRewriting}(\mathcal{M}, \varphi(\mathbf{x}))$;
6 $\Sigma_{21} \longleftarrow \Sigma_{21} \cup (\gamma_1(\mathbf{x}) \to R(\mathbf{x})) \cup \ldots \cup (\gamma_n(\mathbf{x}) \to R(\mathbf{x}))$;
7 $\hat{\mathbf{S}}_2 \longleftarrow \{\hat{R} \mid R \in \mathbf{S}_2\}$;
8 $\Sigma'_{12} \longleftarrow$ replace in Σ_{12} every $R \in \mathbf{S}_2$ by the corresponding $\hat{R} \in \hat{\mathbf{S}}_2$;
9 $\Sigma'_2 \longleftarrow \mathsf{ComposeFull}((\mathbf{S}_2, \mathbf{S}_1, \Sigma_{21}), (\mathbf{S}_1, \hat{\mathbf{S}}_2, \Sigma'_{12}))$;
10 $\Sigma_2 \longleftarrow$ replace in Σ'_2 every $\hat{R} \in \hat{\mathbf{S}}_2$ by the corresponding $R \in \mathbf{S}_2$;

In the following example we illustrate the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ algorithm with the schema mapping and source implicit knowledge from Example 4.3.

Example 4.16. Let Σ_1 be a source implicit knowledge and let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping with schemata $\mathbf{S}_1 = \langle D, E, F\rangle$ and $\mathbf{S}_2 = \langle D', F'\rangle$. The dependencies of Σ_1 and Σ_{12} are defined as follows:

$$\begin{aligned}\Sigma_1 = \{&D(x,y) \wedge D(y,z) \to F(x,y,z),\\ &E(x,y) \wedge E(y,z) \to F(x,y,z)\}\\ \Sigma_{12} = \{&D(x,y) \to D'(x,y),\\ &F(x,y,z) \to F'(x,y,z)\}\end{aligned}$$

It is easy to see that Σ_1 is acyclic and is already unfolded, i.e. $\mathsf{Unfold}(\Sigma_1) = \Sigma_1$ and thus $\Sigma_1 = \Sigma_1^+$. The target rewriting of query $\exists x \exists y(D(x,y) \wedge D(y,z))$ is given by $\exists x \exists y(D'(x,y) \wedge D'(y,z))$. Query $\exists x \exists y(E(x,y) \wedge E(y,z))$ on the other hand is not target rewritable under $\mathcal{M}$. After line 6 we have

$$\Sigma_{21} = \{D'(x,y) \wedge D'(y,z) \to F(x,y,z)\}.$$

Afterwards, a copy $\hat{\mathbf{S}}_2 = \langle \hat{D}', \hat{F}'\rangle$ of schema $\mathbf{S}_2$ is created. In line 8 we assign Σ'_{12} to the result of replacing in Σ_{12} every relation symbol from $\mathbf{S}_2$ by the analogous relation symbol from $\hat{\mathbf{S}}_2$. We have

$$\begin{aligned}\Sigma'_{12} = \{&D(x,y) \to \hat{D}'(x,y),\\ &F(x,y,z) \to \hat{F}'(x,y,z)\}.\end{aligned}$$

The dependencies of the composition of $(\mathbf{S}_2, \mathbf{S}_1, \Sigma_{21})$ and $(\mathbf{S}_1, \hat{\mathbf{S}}_2, \Sigma'_{12})$ are given by $\{D'(x,y) \wedge D'(y,z) \to \hat{F}'(x,y,z)\}$. In the last step all relation symbols from $\hat{\mathbf{S}}_2$ get replaced by the relation symbols in $\mathbf{S}_2$ and we return

$$\Sigma_2 = \{D'(x,y) \wedge D'(y,z) \to F'(x,y,z)\}.$$

□

The following result, shown by Arenas, Pérez and Reutter [10], states that the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ is correct.

Theorem 4.17. [10] *The* $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ *algorithm computes a safe set* Σ_2 *of tgds with inequalities for the acyclic set* Σ_1 *of full tgds and the schema mapping* $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$*, where* Σ_{12} *is a set of full tgds.*

Proof. Let I be an arbitrary instance over $\mathbf{S}_1$, and let J^* be the result of chasing I with Σ_1 and then with Σ_{12}. We have to show that it is always the case that $J^* \models \Sigma_2$, i.e. Σ_2 is safe for Σ_1 and $\mathcal{M}$.

Let Σ_2' be the set constructed in line 9 of the algorithm $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$. Accordingly, Σ_{21} and Σ_{12}' are the sets obtained in line 6 and line 8, respectively. Moreover, let $\hat{J}^*$ be the result of replacing in J^* every relation symbol R from $\mathbf{S}_2$ with the corresponding relation symbol $\hat{R}$ from $\hat{\mathbf{S}}_\mathbf{2}$. For every $I \in \mathsf{Inst}(\mathbf{S_1})$, showing that $J^* \models \Sigma_2$ holds is equivalent to showing that $\langle J^*, \hat{J}^* \rangle \models \Sigma_2'$ holds. Consequently, it suffices to show that for every $I \in \mathsf{Inst}(\mathbf{S_1})$, there exists instance K over $\mathbf{S}_1$, such that $\langle J^*, K \rangle \models \Sigma_{21}$ and $\langle K, \hat{J}^* \rangle \models \Sigma_{12}'$. Let K' be a instance over $\mathbf{S}_1$ obtained by chasing I with Σ_1.

Clearly, $\langle K', \hat{J}^* \rangle \models \Sigma_{12}'$ holds, since $\hat{J}^*$ is the result of chasing K' with Σ_{12}'. We now show that also $\langle J^*, K' \rangle \models \Sigma_{21}$ holds. Consider a dependency $\gamma_i(\mathbf{x}) \rightarrow R(\mathbf{x})$ in Σ_{21} and a tuple $\mathbf{a}$ such that $J^* \models \gamma_i(\mathbf{a})$. We know from line 5 that there is a UCQ$^{\neq}$ of the form $\gamma_1(\mathbf{x}) \vee \ldots \vee \gamma_i(\mathbf{x}) \vee \ldots \vee \gamma_n(\mathbf{x})$, such that $J^* \models \gamma_1(\mathbf{a}) \vee \ldots \vee \gamma_i(\mathbf{a}) \vee \ldots \vee \gamma_n(\mathbf{a})$. Since $\gamma_1(\mathbf{x}) \vee \ldots \vee \gamma_i(\mathbf{x}) \vee \ldots \vee \gamma_n(\mathbf{x})$ is a target rewriting of $\varphi(\mathbf{x})$ and the fact that J^* is obtained by chasing K' with Σ_{12}, we have also that $K' \models \varphi(\mathbf{a})$. It remains to show that also $R(\mathbf{a}) \in K'$ holds, such that $(\varphi(\mathbf{x}) \rightarrow R(\mathbf{x})) \in \Sigma_1^+$. Since K' is the result of chasing I with Σ_1, by definition $K' \models \Sigma_1$ and thus also $K' \models \Sigma_1^+$. It follows that $R(\mathbf{a}) \in K'$. □

4.4.2 Explicit Knowledge Computation

Assume that a source knowledge base (I, Σ_1) and a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_1 is an acyclic set of full tgds and Σ_{12} is a set of full st-tgds. In the previous section the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ algorithm is introduced that computes a safe set Σ_2 for Σ_1 and $\mathcal{M}$ of full tgds over $\mathbf{S}_2$. In this section an approach is discussed that produces an instance J over $\mathbf{S}_2$, such that (J, Σ_2) is a minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$. A candidate for such an instance J is the result of chasing I with Σ_1 and then with Σ_{12}. Clearly, (J, Σ_2) is a minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$, but the explicit and the implicit knowledge contains superfluous knowledge. Intuitively, a much more efficient approach is to materialize a instance J' containing as few tuples implied by Σ_2 as possible.

Arenas et al. introduced in [10] the algorithm $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ that produces a set Σ_1' of full tgds over $\mathbf{S}_1$, such that J' obtained by chasing I with Σ_1' and then with Σ_{12} is still a minimal knowledge base solution and

$J' \subseteq J$ holds. For the computed set Σ_1' and every instance I over S_1 the following conditions hold:

(C1) the result of chasing I with Σ_1' is contained in the output of chasing I with Σ_1.

(C2) (J', Σ_2) is minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$.

The $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ algorithm is defined as follows:

Algorithm: $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$

Input : An acylic set Σ_1 of full tgds over schema $\mathbf{S}_1$, schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a set of full st-tgds, and a safe set Σ_2 for Σ_1 and $\mathcal{M}$ of full tgds with inequalities.

Output: A minimal set Σ_1' of full tgds that satisfies the conditions (C1) and (C2).

```
1 Σ1' ⟵ Unfold(Σ1);
2 while σ ∈ Σ1' s.t. Σ1'\σ satisfies (C1) and (C2) do
3 |  Σ1' = Σ1'\σ;
```

In the following example the $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ algorithm is demonstrated on the source knowledge base and schema mapping which were already used in Example 4.3 and Example 4.16.

Example 4.18. Consider source knowledge base (I, Σ_1), where I consists of the tuples $\{D(a,b), D(b,c), D(b,d), E(a,b), E(b,c), E(c,d)\}$ and the implicit knowledge is given by

$$\begin{aligned}\Sigma_1 = \{&D(x,y) \wedge D(y,z) \rightarrow F(x,y,z),\\ &E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)\}.\end{aligned}$$

Moreover, let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping where the dependencies are defined as

$$\begin{aligned}\Sigma_{12} = \{&D(x,y) \rightarrow D'(x,y),\\ &F(x,y,z) \rightarrow F'(x,y,z)\}.\end{aligned}$$

In Example 4.16 we have illustrated how the application of the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ procedure on $\mathcal{M}$ and Σ_1 results in

$$\Sigma_2 = \{D'(x,y) \wedge D'(y,z) \rightarrow F'(x,y,z)\}.$$

Recall from Example 4.3 the outputs of chasing I with Σ and of chasing I with Σ_1 and then with Σ_{12} which we denote in this example with K_1 and K_{12}, respectively.

$$\begin{aligned}K_1 = \{&D(a,b), D(b,c), D(b,d), E(a,b), E(b,c), E(c,d),\\ &F(a,b,c), F(a,b,d), F(b,c,d)\}\\ K_{12} = \{&D'(a,b), D'(b,c), D'(b,d), F'(a,b,c), F'(a,b,d), F'(b,c,d)\}\end{aligned}$$

It is easy to see that Σ_1 is already unfolded, i.e. $\mathsf{Unfold}(\Sigma_1) = \Sigma_1$. We start by checking whether Σ_1 without dependency $E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)$ still satisfies conditions (C1) and (C2). We start by chasing I with $D(x,y) \wedge D(y,z) \rightarrow F(x,y,z)$ an receive

$$L_1 = \{D(a,b), D(b,c), D(b,d), E(a,b), E(b,c), E(c,d), F(a,b,c), F(a,b,d)\}.$$

Chasing L successively with Σ_{12} and Σ_2 results in instance

$$L_2 = \{D'(a,b), D'(b,c), D'(b,d), F'(a,b,c), F'(a,b,d)\}.$$

Instance L_1 satisfies condition (C1), but instance L_2 violates condition (C2). Thus, condition $E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)$ cannot removed from Σ_1. Conversely, it can be shown that dependency $D(x,y) \wedge D(y,z) \rightarrow F(x,y,z)$ can be removed, such that for every instance in $\mathsf{Inst}(\mathbf{S}_1)$ the conditions (C1) and (C2) are satisfied. Let J_1, J_{12} and J_2 be the output of chasing I with $E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)$, J_1 with Σ_{12} and J_{12} with Σ_2, respectively.

$$\begin{aligned} J_1 &= \{D(a,b), D(b,c), D(b,d), E(a,b), E(b,c), E(c,d), F(a,b,c), F(b,c,d)\} \\ J_{12} &= \{D'(a,b), D'(b,c), D'(b,d), F'(a,b,c), F'(b,c,d)\} \\ J_2 &= \{D'(a,b), D'(b,c), D'(b,d), F'(a,b,c), F'(b,c,d), F'(a,b,d)\} \end{aligned}$$

The instances J_1 and J_2 satisfies both conditions (C1) and (C2). Since no further tgd can be removed, the output of $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ is

$$\Sigma_1' = \{E(x,y) \wedge E(y,z) \rightarrow F(x,y,z)\}.$$

We conclude that (J_{12}, Σ_2) is a minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$. □

The correctness of the $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ follows immediately from the following lemma. The proof of Lemma 4.19 can be found in the full version of [10].

Lemma 4.19. [10] *Let Σ_1 be an acyclic set of full tgds over $\mathbf{S}_1$, and let $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ be a schema mapping with a set of full st-tgds Σ_{12}. Moreover, consider a set of full tgds Σ_1' for which $\Sigma_1 \models \Sigma_1'$ holds, and a set Σ_2 of full tgds with inequalities that is safe for Σ_1 and $\mathcal{M}$. It can be decided in exponential time whether for every $I \in \mathsf{Inst}(\mathbf{S}_1)$ the chase of I subsequently with Σ_1', Σ_{12} and Σ_2 results in the same target instance as the chase of I with Σ_1 and then with Σ_{12}.*

Note, that the output of $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ is not necessarily unique. The result of $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ depends on the order in which the dependencies are processed. As already shown in Example 4.18, the MINIMAL-KNOWLEDGEBASESOLUTION($\mathcal{M}$) problem can be solved with a combination of the $\mathsf{FullSafe}(\mathcal{M}, \Sigma_1)$ procedure, the $\mathsf{Minimize}(\mathcal{M}, \Sigma_1, \Sigma_2)$ procedure and the chase procedure. The following theorem, proposed in [10], is a direct consequence of Theorem 4.17 and Lemma 4.19.

Theorem 4.20. [10] *Consider an acyclic set Σ_1 of full tgds over $\mathbf{S}_1$ and a schema mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, where Σ_{12} is a set of full st-tgds. Let Σ_2 be the result of the* FullSafe$(\mathcal{M}, \Sigma_1)$ *procedure and let Σ_1' be the result of* Minimize$(\mathcal{M}, \Sigma_1, \Sigma_2)$*. Furthermore, let J be the result of subsequently chasing an arbitrary instance I over $\mathbf{S}_1$ with Σ_1' and Σ_{12}. Then (J, Σ_2) is a minimal knowledge base solution for (I, Σ_1) under $\mathcal{M}$.*

CHAPTER 5

XML Data Exchange

The XML data model can be seen as a generalization of the relational data model. In fact, relational data can be represented by flat trees. It suggests itself to consider also data exchange for XML documents. While the key problems are the same as in relational data exchange, the results differ in XML data exchange. This is mainly due to the fact that we have to consider not only the values of data, but also their structure. Moreover, there has to be dealt with the fundamental issue that the concept of universal solutions does not exist in XML data exchange, i.e. there is in general no single solution that represents all possible solutions. Nevertheless, suitable algorithms can be found for many of the problems that we have already encountered in relational data exchange and in knowledge exchange. We focus in this chapter on three fundamental problems, namely the problem of recognizing solutions, the problem of determining whether a solution exists, and the problem of query evaluation.

XML data exchange was first investigated by Arenas and Libkin [9]. Their proposed schema mappings were rather simplistic and did not allow to reason over the entire structure of XML trees. For example, their formalism did not take horizontal navigation into account. Amano et al. [4, 3] addressed these issues by extending the notion of XML schema mappings from Arenas and Libkin [9]. David et al. [30] criticized the fact that XML schema mappings reason over trees, but queries over them result in tuples. Therefore, they proposed a pattern-based XML query language with trees as outputs and showed how they could be applied in XML data exchange. Moreover, Bojańczyk et al. [23] closed different open questions concerning the static analysis of XML data exchange settings.

In this chapter we focus on the results from [9, 4, 3]. In Section 5.1, XML data exchange is formalized. In addition, a classification is presented that allows us to denote restricted versions of XML schema mappings. We conclude this section by investigating the problem of deciding whether a pair of XML trees satisfies all dependencies imposed by XML schema mappings. In Section 5.2 two problems regarding the static analysis of XML trees are discussed. In

particular, we highlight the problem where we ask whether a solution under a given schema mapping exists for either all or at least for one of the source trees. Query answering over the target with respect to a source XML tree is studied in Section 5.3. Query answering is in general in coNP, but there exists a tractable algorithm if the features of the XML schema mappings are restricted.

5.1 XML Data Exchange Settings

In this section the notion of data exchange settings from Chapter 3 is adapted for an application on the XML data model. Motivated by the fact that the XML data model can be seen as a generalization of the relational data model, also XML data exchange should represent a generalization of relational data exchange. In the relational case a data exchange setting is a triple consisting of two schemata and a set of dependencies. As counterpart to relational schemata the formalism of document type definitions (DTDs) and xml schema definitions (XSDs) have to be considered [56]. It has turned out that the data exchange community favors the more simplistic notion of DTDs for XML data exchange [9, 4, 3, 30, 23]. On account of the tree structure of XML documents, also an alternative for the relational dependencies from Section 3.2.1 and Section 3.2.2 has to be found. Arenas and Libkin [9] provided dependencies for XML schema mappings based on tree-pattern formulae, which were extended in [4]. We start with the definition of XML schema mappings and define the corresponding dependencies in the following subsections.

Definition 5.1. *An* XML data exchange setting *is an* XML schema mapping *given by a triple* $\mathcal{M} = (D_1, D_2, \Sigma)$ *consisting of a* source DTD D_1*, a* target DTD D_2*, and a set of dependencies* Σ*. Consider a tree* T_1 *conforming to* D_1 *and a tree* T_2 *conforming to* D_2*, then* T_2 *is a* solution for T_1 under $\mathcal{M}$ *if* $\langle T_1, T_2 \rangle$ *satisfies all dependencies in* Σ*. The set of all solutions for* T_1 *under* $\mathcal{M}$ *is denoted with* $\mathsf{Sol}_{\mathcal{M}}(T_1)$*. Moreover,* $\mathsf{Sol}(\mathcal{M})$ *denotes the set of all tuples* $\langle T_1, T_2 \rangle$*, where* T_1 *and* T_2 *conform to* D_1 *and* D_2*, respectively, and* T_2 *is a solution for* T_1 *under* $\mathcal{M}$*.*

We use the terms XML data exchange setting and XML schema mapping interchangeably in this thesis. Moreover, in the range of this chapter we omit frequently the term XML. Conversely, if we consider data other than XML it is always stated explicitly. Furthermore, we say *source tree* as a shorthand for an XML tree conforming to a source DTD. Analogously, we write *target tree* for a XML tree conforming to a target DTD.

As already pointed out, we have to specify dependencies that can handle the tree structure. In Section 5.1.1 source-to-target dependencies are introduced. Some of the problems discussed in this chapter have different complexity bounds if we restrict some of the features of the used XML documents and source-to-target dependencies. We give in Section 5.1.2 a classification of XML data exchange settings that allow different sets of features in their

DTDs and dependencies. In Section 5.1.3 we discuss complexity results of problems concerning the recognition of solutions under XML schema mappings.

5.1.1 Source-to-Target Dependencies

In Definition 5.1 schema mappings for the XML data model are introduced. However, this definition did not specify how dependencies have to look like. Such dependencies should naturally extend st-tgds from Chapter 3. This section presents dependencies based on tree patterns.

Definition 5.2. *A* source-to-target dependency*, or* std *for short, is a formula of the form*

$$\pi(\mathbf{x}, \mathbf{y}), \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \pi'(\mathbf{x}, \mathbf{z}), \alpha'(\mathbf{x}, \mathbf{z}),$$

where π and π' are tree patterns over $\mathbf{x} \cup \mathbf{y}$ and $\mathbf{x} \cup \mathbf{z}$, respectively, and α is a conjunction of equalities and inequalities among the variables $\mathbf{x} \cup \mathbf{y}$, and α' is a conjunction of equalities and inequalities among the variables $\mathbf{x} \cup \mathbf{z}$. Moreover, each variable ξ_0 in $\mathbf{x} \cup \mathbf{y}$ is safe*, i.e. either ξ_0 appears in π or $\xi_0 = \xi_1, \xi_1 = \xi_2, \ldots, \xi_{k-1} = \xi_k$ are equalities in α and ξ_k appears in π.*

Two trees T_1 and T_2 satisfy *an std σ, if for every pair of tuples $\mathbf{a}$ and $\mathbf{b}$ which satisfy $\pi(\mathbf{a}, \mathbf{b})$ and $\alpha(\mathbf{a}, \mathbf{b})$, there exists a tuple $\mathbf{c}$ such that $\pi'(\mathbf{a}, \mathbf{c})$ and $\alpha'(\mathbf{a}, \mathbf{c})$ are satisfied.*

According to our definition of tree patterns, variables are allowed to occur in a tree pattern only once. We relax sometimes this requirement and express equalities by repeating the corresponding variable. For instance we write $r[t_1(x, y), t_2(y, z)]$ for the pattern $r[t_1(x, y_1), t_2(y_2, z)]$ and the equality $(y_1 = y_2)$.

In the following example the so far discussed concepts about the XML data model and XML exchange are demonstrated. We start by giving an example of two DTDs and show two XML trees that conform to those DTDs. In addition, a schema mapping is presented that relates both XML documents. As already mentioned, we assume that attributes of a node have a specific order, even though it is not specified by DTDs. This allows us to assign values to attributes implicitly, and thus simplifies the notation.

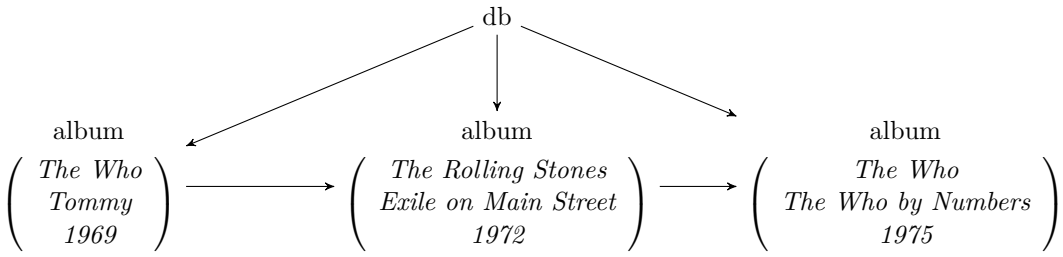


Figure 5.1: Source tree T_1

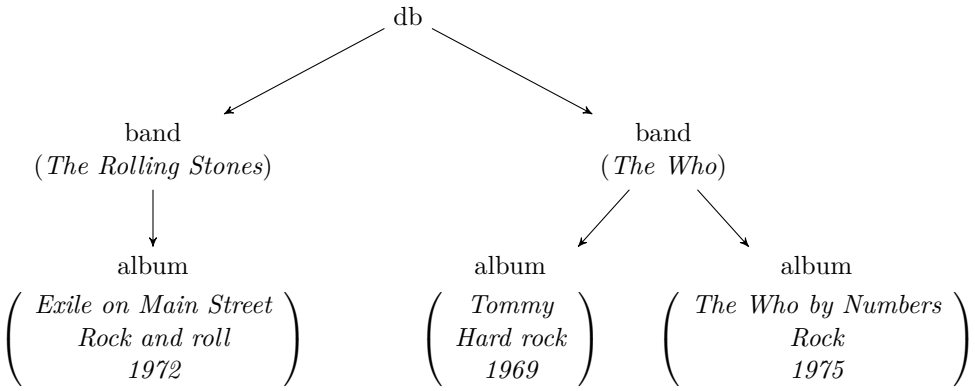


Figure 5.2: Target tree T_2

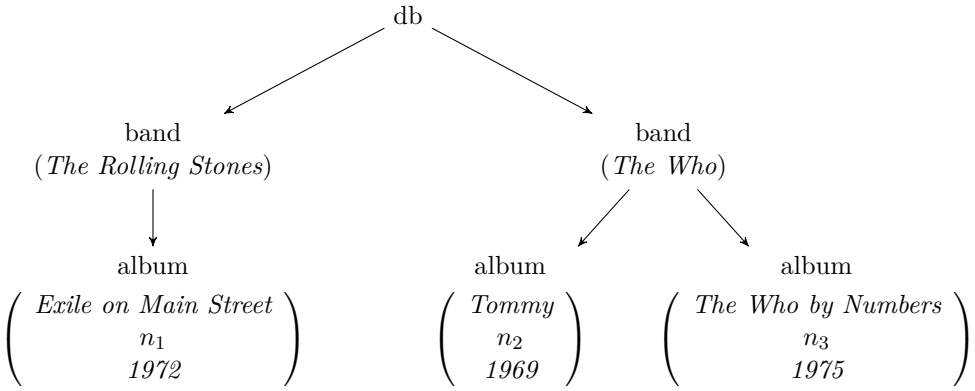


Figure 5.3: Target tree T_3

Example 5.3. Consider a DTD $D_1 = \langle db, P_{D_1}, A_{D_1} \rangle$ over $\Gamma_1 = \{db, album\}$ and $Att_1 = \{@title, @band, @year\}$ and a DTD $D_2 = \langle db, P_{D_2}, A_{D_2} \rangle$ over $\Gamma_2 = \{db, band, album\}$ and $Att_2 = \{@name, @title, @year, @genre\}$. The DTD D_1 is defined as follows:

$$\begin{aligned} P_{D_1} =& \{db \to album^*; \\ & \quad album \to \epsilon\} \\ A_{D_1}(db) =& \emptyset \\ A_{D_1}(album) =& \{@band, @title, @year\} \end{aligned}$$

The DTD D_2 is given by the following rules:

$$\begin{aligned} P_{D_2} =& \{db \to band^*; \\ & \quad band \to album^*; \\ & \quad album \to \epsilon\} \\ A_{D_2}(db) =& \emptyset \\ A_{D_2}(band) =& \{@name\} \\ A_{D_2}(album) =& \{@title, @genre, @year\} \end{aligned}$$

Figure 5.1 shows an XML tree T_1 that conforms to D_1. Analogously, tree T_2 and tree T_3 from Figure 5.2 and Figure 5.3, respectively, conform both to DTD D_2. Observe, that the attribute values of the attributes with label $@genre$ are represented in T_2 by elements from Const and in T_3 by elements from Var. Thus, tree T_2 is more specific compared to tree T_3. Let $\mathcal{M} = (D_1, D_2, \Sigma)$ be a schema mapping, where Σ is given by the following single std:

$$db/album(x, y, z) \rightarrow db/band(x)/album(y, w, z)$$

It is easy to confirm that $\langle T_1, T_2 \rangle$ as well as $\langle T_1, T_3 \rangle$ satisfy the std in Σ, and thus T_2 and T_3 are solutions for T_1 under $\mathcal{M}$. □

The ensuing example demonstrates how relational schema mappings can be transformed into an XML data exchange setting. Note that Example 5.4 extends an example from [4].

Example 5.4. Let $\mathcal{M}_1 = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12} \cup \Sigma_2)$ be a relational schema mapping, where $\mathbf{S}_1$ is composed of two binary relation symbols E and F, $\mathbf{S}_2$ consists of a single ternary relation symbol M, and $\Sigma_{12} = \{E(x, y) \wedge F(y, z) \rightarrow M(x, y, z)\}$.

We define now an XML schema mapping that represents the above relational data exchange setting. We start by defining the analogs to the schema $\mathbf{S}_1$ and $\mathbf{S}_2$. Assume a DTD $D_1 = \langle r, P_{D_1}, A_{D_1} \rangle$ with $P_{D_1} = \{r \rightarrow e, f; e \rightarrow d_1^*; f \rightarrow d_2^*; d_1 \rightarrow \epsilon; d_2 \rightarrow \epsilon\}$, and A_{D_1} results for r, e and f in the empty set, $A_{D_1}(d_1) = \{@x, @y_1\}$ and $A_{D_1}(d_2) = \{@y_2, @z\}$. Moreover, consider the DTD $D_2 = \langle r, P_{D_2}, A_{D_2} \rangle$, where $P_{D_2} = \{r \rightarrow g; g \rightarrow d_3^*; d_3 \rightarrow \epsilon\}$, A_{D_2} is for d_3 $\{@x, @y, @z\}$, and the empty set for r and g. Let Σ be the std given by

$$r[e[d_1(x, y_1)], f[d_2(y_2, z)]], y_1 = y_2 \rightarrow r[g[d_3(x, y_1, z)]].$$

It can be shown that $\mathcal{M}_2 = (D_1, D_2, \Sigma)$ reproduces $\mathcal{M}_1 = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$, i.e. for every pair of instances $(I, J) \in \mathcal{M}_1$, there exists a source tree T_1 and a target tree T_2, such that $\langle T_1, T_2 \rangle \in \mathsf{Inst}(\mathcal{M})$. □

5.1.2 Classification of XML Schema Mappings

The source-to-target dependencies given in Definition 5.2 are rather general and lead in the problems discussed in the following sections sometimes to high complexity bounds or even to undecidability. We will see that the high complexity bounds of the discussed problems are caused by different properties of stds and the structure of DTDs. This motivates us to isolate different characteristics of XML schema mappings and to provide a classification of schema mappings that utilize only subsets of those characteristics. We follow the notations from [4].

Tree patterns have four different axes for tree navigation. The navigational axes are referred as $\downarrow$ for the child axis, $\rightarrow$ for the next sibling axis, $\downarrow^*$ for the descendant axis, and $\rightarrow^*$ for the following child axis. It is denoted with

_ that a tree pattern contains wildcard symbols. Furthermore, the case that an std uses equalities and inequalities is denoted by $=$ and $\neq$, respectively. Moreover, we use the abbreviations $\Downarrow$ for $\{\downarrow, \downarrow^*, \_\}$, $\Rightarrow$ for $\{\rightarrow, \rightarrow^*\}$, and $\sim$ for $\{=, \neq\}$.

Definition 5.5. *Let $\sigma \subseteq \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =, \neq, \_, \Downarrow, \Rightarrow, \sim\}$ be a signature of features. Then* $\mathrm{SM}(\sigma)$ *describes the* class of XML schema mappings *defined by stds, which only use the features in σ. The class* $\mathrm{SM}^{\mathrm{nr}}(\sigma)$ *restricts* $\mathrm{SM}(\sigma)$ *further, such that the schema mappings are determined by nested relational DTDs. Moreover,* $\mathrm{SM}^{\circ}(\sigma)$ *is the subset of* $\mathrm{SM}(\sigma)$ *where the stds do not mention attribute values, i.e. all tree patterns are of the form $l[\lambda]$.*

XML data exchange was originally proposed for the class $\mathrm{SM}(\Downarrow)$ by Arenas and Libkin [9]. In the range of this work the existence of the child axis $\downarrow$ is always assumed. Although, the class $\mathrm{SM}^{\circ}(\sigma)$ is useless for exchanging actual data, it represents a useful tool for hardness proofs. On the other hand, the class $\mathrm{SM}^{\mathrm{nr}}(\sigma)$ is of practical importance, since it allows in many cases to lower complexity bounds. Moreover, a comprehensive empirical analysis has shown that nested relational DTDs are common in real life [13]. For instance, if relational databases are encoded as in Example 5.4, then relational schema mappings can be represented by XML schema mappings from the class $\mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$.

One of the key ideas behind restricting the schema mappings is to reduce the possibilities of how a tree pattern can be put on a tree. Consequently, more specific schema mappings permit in general more efficient algorithms. Diversity in the structure of trees can be prohibited by the use of nested relational DTDs. In addition, wildcards and descendants in tree patterns have to be disallowed to avoid guessing. The following definition restricts the use of them.

Definition 5.6. *A schema mapping is* fully specified *if the tree pattern in the conclusion of every std does not use _ or $\downarrow^*$.*

5.1.3 Solution Recognition

In this section the problem of deciding whether two XML trees form a solution for a schema mapping is examined. We discuss the data complexity and the combined complexity of this problem, i.e. the version of the problem where the schema mapping is considered as given and the version where the schema mapping is part of the input. The data complexity variant of recognizing solutions is defined as follows:

PROBLEM:	TESTSOLUTION($\mathcal{M}$)
INPUT:	trees T_1 and T_2
QUESTION:	is $\langle T_1, T_2 \rangle \in \mathsf{Sol}(\mathcal{M})$?

Amano, Libkin and Murlak [4] proved the following result.

Theorem 5.7. [4] *The* TestSolution($\mathcal{M}$) *problem for schema mappings in* SM($\Downarrow, \Rightarrow, \sim$) *is LOGSPACE-complete.*

Proof. Let $\mathcal{M}$ be defined by source DTD D_1, target DTD D_2 and by a set of stds Σ. It is shown in [60] that deciding, whether an XML tree conforms to a DTD is in LOGSPACE. The tree T_2 is a solution for T_1 if and only if $\langle T_1, T_2 \rangle$ satisfies all stds in Σ. Let $\sigma \in \Sigma$ be a single std given by $\pi(\mathbf{x}, \mathbf{y}), \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \pi'(\mathbf{x}, \mathbf{z}), \alpha'(\mathbf{x}, \mathbf{z})$, where $\mathbf{x} = x_1, \ldots, x_k$, $\mathbf{y} = y_1, \ldots, y_l$ and $\mathbf{z} = z_1, \ldots, z_m$. Moreover, let A_1 and A_2 be the set of data values from V which occur in the corresponding tree T_1 or T_2. According to Definition 5.2, the std σ is satisfied if for each $\mathbf{a} \in (A_1 \cup A_2)^k$ and each $\mathbf{b} \in A_1^l$, which satisfy $\pi(\mathbf{a}, \mathbf{b})$ and $\alpha_{=,\neq}(\mathbf{a}, \mathbf{b})$, exists $\mathbf{c} \in A_2^m$ such that $\pi'(\mathbf{a}, \mathbf{c})$ and $\alpha'_{=,\neq}(\mathbf{a}, \mathbf{c})$ are satisfied. Since k, l and m are not part of the input and are therefore considered as constants, $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ can be represented in logarithmic size with respect to the size of T_1 and T_2. Theorem 2.6 stated that tree patterns can be evaluated in DLOGSPACE. Moreover, it is folklore that conjunctions of equalities and inequalities can be verified in LOGSPACE. The resulting algorithm verifies first that T_1 and T_2 are a source and a target tree and then iterates over all possible combinations of $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ and checks if all stds in Σ are satisfied.

The LOGSPACE-hardness follows immediately from the fact that checking whether a tree pattern is satisfied in a tree is a LOGSPACE-hard problem. □

We look next at the TestSolution problem which is defined analogous to TestSolution($\mathcal{M}$), but considers the schema mapping as part of the input. The following result from [4] shows a notable increase in complexity compared to the data complexity of the problem. It turns out that the main source of complexity are the number of variables in the patterns.

Theorem 5.8. [4] *For schema mappings in* SM($\Downarrow, \Rightarrow, \sim$) *is the* TestSolution *problem* Π_2^p*-complete. If additionally the maximum number of variables per pattern is fixed, then* TestSolution *is solvable in PTIME.*

Proof. Assume that the schema mapping $\mathcal{M}$ is defined by source DTD D_1, target DTD D_2 and by a set of stds Σ.

We start with the Π_2^p membership by giving an algorithm for the complementary problem. This algorithm first guesses an std $\pi(\mathbf{x}, \mathbf{y}), \alpha(\mathbf{x}, \mathbf{y}) \rightarrow \pi'(\mathbf{x}, \mathbf{z}), \alpha'(\mathbf{x}, \mathbf{z})$ and tuples $\mathbf{a}$, $\mathbf{b}$, and then checks that $T_1 \models \pi(\mathbf{a}, \mathbf{b}), \alpha(\mathbf{a}, \mathbf{b})$ and $T_2 \not\models \pi'(\mathbf{a}, \mathbf{z}), \alpha'(\mathbf{a}, \mathbf{z})$. By Theorem 2.7, checking whether $T_1 \models \pi(\mathbf{a}, \mathbf{b})$ holds can be done in polynomial time. Moreover, also $\alpha(\mathbf{a}, \mathbf{b})$ can be validated in polynomial time. Deciding whether $T_2 \not\models \pi'(\mathbf{a}, \mathbf{z}), \alpha'(\mathbf{a}, \mathbf{z})$ holds is in coNP and can be determined by guessing a tuple $\mathbf{c}$ and checking in polynomial time whether $T_2 \models \pi'(\mathbf{a}, \mathbf{c}), \alpha'(\mathbf{a}, \mathbf{c})$ holds.

Π_2^p hardness is shown by a reduction from QSAT_2. QSAT_2 asks whether a Π_2 quantified Boolean formula (QBF), given by $\forall x_1 \cdots \forall x_m \exists x_{m+1} \cdots \exists x_n \bigwedge_{j=1}^{k} Z_j^1 \vee Z_j^2 \vee Z_j^3$, is satisfiable. The reduction has the following intuition. The source tree expresses that each variable in the QBF can be either 0 or

1. The target tree represents the variable assignments such that a clause is satisfied. Moreover, the QBF is translated into a single dependency. Source and target tree are defined as follows:

$$\begin{aligned}
T_1 &= r[v(0), v(1)] \\
T_2 &= r[v[v_1(0), v_2(0), v_3(1), \bar{v}_1(1), \bar{v}_2(1), \bar{v}_3(0)], \\
&\quad\; v[v_1(0), v_2(1), v_3(0), \bar{v}_1(1), \bar{v}_2(0), \bar{v}_3(1)], \\
&\quad\; v[v_1(0), v_2(1), v_3(1), \bar{v}_1(1), \bar{v}_2(0), \bar{v}_3(0)], \\
&\quad\; v[v_1(1), v_2(0), v_3(0), \bar{v}_1(0), \bar{v}_2(1), \bar{v}_3(1)], \\
&\quad\; v[v_1(1), v_2(0), v_3(1), \bar{v}_1(0), \bar{v}_2(1), \bar{v}_3(0)], \\
&\quad\; v[v_1(1), v_2(1), v_3(0), \bar{v}_1(0), \bar{v}_2(0), \bar{v}_3(1)], \\
&\quad\; v[v_1(1), v_2(1), v_3(1), \bar{v}_1(0), \bar{v}_2(0), \bar{v}_3(0)]]
\end{aligned}$$

Notice that the children of the root node in T_2 are of the form $v[v_1(a_1), v_2(a_2), v_3(a_3), \bar{v}_1(\bar{a}_1), \bar{v}_2(\bar{a}_2), \bar{v}_3(\bar{a}_3)]$, where $a_1, a_2, a_3 \in \{0, 1\}$ and $\bar{a}_i$ denotes the negation of a_i.

The set Σ consists of a single std defined as

$$r[v(x_1), v(x_2), \ldots, v(x_m)] \to r[v[\varphi_1^1, \varphi_1^2, \varphi_1^3], v[\varphi_2^1, \varphi_2^2, \varphi_2^3], \ldots, v[\varphi_k^1, \varphi_k^2, \varphi_k^3]],$$

where $\varphi_j^l = v_l(x_i)$ for $Z_j^l = x_i$ and $\varphi_j^l = \bar{v}_l(x_i)$ for $Z_j^l = \bar{x}_i$. We denote the left-hand side of the above std with $\pi(\mathbf{x})$ and the right-hand side with $\pi'(\mathbf{x}, \mathbf{y})$.

It remains to show that the QBF is satisfiable if and only if for each valuation $\mathbf{a}$ of $x_1, \ldots, x_m$, there exists a valuation $\mathbf{b}$ that extends $\mathbf{a}$, such that $T_1 \models \pi(\mathbf{a})$ and $T_2 \models \pi'(\mathbf{b})$. Therefore, we choose an arbitrary conjunct from the QBF, say $x_i \vee x_j \vee \bar{x}_k$. According to our definition π' contains the subpattern $v[v_1(x_i), v_2(x_j), \bar{v}_3(x_k)]$. Assume there exists a valuation $\mathbf{b}$, such that the subpattern is mapped into the subtree $v[v_1(a_1), v_2(a_2), v_3(a_3), \bar{v}_1(\bar{a}_1), \bar{v}_2(\bar{a}_2), \bar{v}_3(\bar{a}_3)]$. We have the valuation $x_1 = a_1$, $x_2 = a_2$ and $x_3 = \bar{a}_3$ and the values of the literals are $x_1 = a_1$, $x_2 = a_2$ and $\bar{x}_3 = a_3$. By the construction of T_2 at least one of a_1, a_2 or a_3 is 1, and thus the conjunct is satisfied.

Assume additionally that the maximum number of variables per pattern is fixed. We can reuse the algorithm described in the proof of Theorem 5.7. The PTIME membership follows immediately the fact that there are only polynomially many valuations. □

5.2 Static Analysis of XML Schema Mappings

Static analysis aims to determine certain properties of an XML data exchange setting. We focus here on properties concerning the existence of solutions. As we will see in Example 5.9, a source tree for an XML schema mapping defined by source-to-target dependencies need not necessarily have a solution. Moreover, there are schema mappings defined by stds which have no solution for any possible source tree. The following example, inspired from

[4], first presents a schema mapping, where every source tree has no solution and shows then a modification of the schema mapping, such that some but not all source trees have a solution.

Example 5.9. Let $D_1 = \langle r_1, P_{D_1}, A_{D_1} \rangle$ be a DTD over $\Gamma_1 = \{r_1, s\}$ and $Att_1 = \{@p\}$, and let $D_2 = \langle r, P_{D_2}, A_{D_2} \rangle$ be a DTD over $\Gamma_2 = \{r_2, t\}$ and $Att_2 = \{@q\}$. The DTDs are defined as follows:

$$\begin{aligned} P_{D_1} &= \{r_1 \to s*; & A_{D_1}(r_1) &= \emptyset \\ & \quad s \to \epsilon\} & A_{D_1}(s) &= \{@p\} \\ P_{D_2} &= \{r_2 \to t; & A_{D_2}(r_2) &= \emptyset \\ & \quad t \to \epsilon\} & A_{D_2}(t) &= \{@q\} \end{aligned}$$

Consider a schema mapping $\mathcal{M} = (D_1, D_2, \Sigma)$, where Σ is given by the std

$$r_1[s(x), s(y)], x \neq y \to r_2/t(x), x = y.$$

Since $x \neq y$ and $x = y$ can never be both fulfilled at the same time, there exists no tree T_1 conforming to D_1, which has a solution under $\mathcal{M}$. If Σ is defined instead as

$$\{r_1/s(x) \to r_2/t(x)\},$$

then there exist solutions only for some of the source trees. For instance, a source tree consisting of only one node labeled with s has solutions, while a source tree with two or more children at its root node has no solution. □

Two static analysis problems are considered here. The first static analysis problem, considered in Section 5.2.1, asks for a given schema mapping, whether there exists a source tree that has a solution under the schema mapping. The second static analysis problem, discussed in Section 5.2.2, asks for a given schema mapping whether every source tree has a solution.

5.2.1 Consistency

A schema mapping $\mathcal{M}$ is *consistent* if there exists at least one source tree T_1 that has a solution, i.e. $\mathsf{Sol}_{\mathcal{M}}(T_1) \neq \emptyset$. We consider in this section the following decision problem:

PROBLEM:	CONSISTENCY(σ)
INPUT:	a schema mapping $\mathcal{M} \in \mathrm{SM}(\sigma)$
QUESTION:	is $\mathsf{Sol}(\mathcal{M}) \neq \emptyset$?

This problem was first investigated for XML data exchange settings by Arenas and Libkin [9] for the class SM($\Downarrow$) of schema mappings. The consistency problem was then investigated for much broader classes by Amano et al. [4]. It turns out that CONSISTENCY(σ) has high complexity bounds and is even for many classes of schema mappings undecidable. The goal of this section is to examine the different sources of complexity rather than to present all complexity results exhaustively.

Recall, that $\mathrm{SM}^{\circ}(\sigma)$ is a class of schema mappings, where the dependencies do not use attribute values. The $\textsc{Consistency}^{\circ}(\sigma)$ problem is defined analogously to $\textsc{Consistency}(\sigma)$, but restricts the input to schema mappings from $\mathrm{SM}^{\circ}(\sigma)$.

We start the investigation of the $\textsc{Consistency}(\sigma)$ problem with $\mathrm{SM}(\Downarrow, \Rightarrow, \sim)$, the largest class considered in this section.

Theorem 5.10. [4] $\textsc{Consistency}(\Downarrow, \Rightarrow, \sim)$ *is undecidable.*

The undecidability of Theorem 5.10 can be shown by reduction from the halting problem of 2-register machines, i.e. for a 2-register machine a schema mapping can be constructed which is consistent if and only if the machine halts. To obtain decidability for the $\textsc{Consistency}(\sigma)$ problem, we have to impose restrictions. Thus we have to restrict either the DTDs used in the schema mapping or the dependencies. We start with the second and disallow equalities and negations.

Theorem 5.11. [4] *The* $\textsc{Consistency}(\Downarrow, \Rightarrow)$ *problem is EXPTIME-complete.*

Proof. *(Sketch)* The main observation of this proof is that $\textsc{Consistency}(\Downarrow, \Rightarrow)$ is not harder than $\textsc{Consistency}^{\circ}(\Downarrow, \Rightarrow)$. The hardness of $\textsc{Consistency}^{\circ}(\Downarrow, \Rightarrow)$ can be shown via tree automata techniques. We show here only the reduction from $\textsc{Consistency}(\Downarrow, \Rightarrow)$ to $\textsc{Consistency}^{\circ}(\Downarrow, \Rightarrow)$.

Assume an arbitrary tree pattern π. We denote with π° the tree pattern obtained by replacing in π every subpattern of the form $l(\mathbf{x})[\lambda]$ by $l[\lambda]$, where l is either a label or a wildcard. A schema mapping $\mathcal{M} = (D_1, D_2, \Sigma) \in \mathrm{SM}(\Downarrow, \Rightarrow)$ can be transformed into a schema mapping $\mathcal{M}' = (D_1, D_2, \Sigma^{\circ}) \in \mathrm{SM}^{\circ}(\Downarrow, \Rightarrow)$ by replacing every std of the form $\pi_1 \to \pi_2$ into a std of the form $\pi_1^{\circ} \to \pi_2^{\circ}$. It remains to show that $\mathcal{M}$ is consistent if and only if $\mathcal{M}'$ is consistent. It is easy to see that $\mathsf{Sol}(\mathcal{M}) \subseteq \mathsf{Sol}(\mathcal{M}')$ holds. Assume that $\mathcal{M}^{\circ}$ is consistent, i.e. there exists a tuple (T_1, T_2) such that $(T_1, T_2) \in \mathcal{M}^{\circ}$. Let (T_1', T_2') be a tuple of trees obtained by setting all attributes in (T_1, T_2) to the same value. Clearly, T_1' and T_2' are valid source and target trees. Moreover, (T_1', T_2') satisfies all dependencies, and thus also $\mathcal{M}$ is consistent. □

The next arising question is if we can lower the complexity bounds if we disallow in addition horizontal axes. However, Arenas and Libkin [9] showed that $\textsc{Consistency}(\Downarrow)$ remains EXPTIME-complete. A further question is if the combination of horizontal axes and data comparisons causes undecidability, or if data comparisons are alone the cause. The next result confirms the latter.

Theorem 5.12. [4] *The problem* $\textsc{Consistency}(\Downarrow, \sim)$ *is undecidable.*

The undecidability result of Theorem 5.12 is not surprising, since related problems showed undecidability as soon as data values comparisons are allowed [22, 21, 29, 42, 61]. The above theorem can be shown similar to Theorem 5.10, by a reduction from the halting problem of a 2-register machine.

We investigate next if the complexity of the CONSISTENCY(σ) problem can be influenced by considering less expressive DTDs. CONSISTENCY$^{\text{nr}}(\sigma)$ denotes a variant of the CONSISTENCY(σ) problem, that allows only schema mappings from SM$^{\text{nr}}(\sigma)$ as input, i.e. schema mappings where the source and the target DTDs are nested relational. The next result states that in the presence of data comparisons, vertical axes and horizontal axes the consistency problem remains undecidable even under the restriction to nested relational DTDs.

Theorem 5.13. [4] *The problem* CONSISTENCY$^{\text{nr}}(\Downarrow, \Rightarrow, \sim)$ *is undecidable.*

Theorem 5.13 can be shown by a modification of the proof for Theorem 5.10, i.e. by a reduction from the halting problem of a 2-register machine. Nevertheless, this modification cannot be repeated on the proof for Theorem 5.12. In fact Amano et al. [4] showed that the CONSISTENCY$^{\text{nr}}(\Downarrow, \sim)$ problem is actually decidable.

Theorem 5.14. [4] *The* CONSISTENCY$^{\text{nr}}(\Downarrow, \sim)$ *problem is NEXPTIME-complete.*

Moreover, the restriction to nested-relational data allows lowering the complexity bounds for schema mappings with vertical and horizontal axes.

Theorem 5.15. [4] CONSISTENCY$^{\text{nr}}(\Downarrow, \Rightarrow)$ *is PSPACE-complete.*

The membership of CONSISTENCY$^{\text{nr}}(\Downarrow, \Rightarrow)$ in PSPACE can be shown by a reduction to XPath satisfiability, which is studied in [18]. As already mentioned, the complexity boundaries of the CONSISTENCY$(\Downarrow, \Rightarrow)$ problem cannot be improved by disallowing horizontal axes. This changes for schema mappings defined by nested relational DTDs and allows one to find a tractable algorithm.

Theorem 5.16. [9] *The* CONSISTENCY$^{\text{nr}}(\Downarrow)$ *problem is solvable in polynomial time.*

Proof. *(Sketch)* Consider an arbitrary nested relational DTD D. We denote with D° the result of replacing in D each l^+ by l and each $l?$ and l^* by ϵ. Moreover, we denote with D^* the result of replacing in D each l^+, $l?$ and l^* by l. If D does not have attributes, then D° and D^* have both exactly one conforming tree.

Let $\mathcal{M} = (D_1, D_2, \Sigma)$ be an arbitrary input of the CONSISTENCY$^{\text{nr}}(\Downarrow)$ problem. Clearly, $\mathcal{M}$ can be transformed in linear time into a schema mapping $\mathcal{M}' = (D_1^\circ, D_2^*, \Sigma)$. As already shown in the proof sketch of Theorem 5.11, we can assume without loss of generality that the stds in Σ do not mention attributes. Therefore, we are also allowed to assume that D_1 and D_2 do not have attributes. Furthermore, it can be shown that $\mathcal{M}$ is consistent if and only if $\mathcal{M}'$ is consistent. It suffices to evaluate the stds in Σ on $\langle T_1, T_2 \rangle$, where T_1 and T_2 conform to D_1° and D_2^*, respectively. This can be done in time $O(nm^2)$, where n is the size of D_1 and D_2, and m is the size of Σ. □

5.2.2 Absolute Consistency

The previous section highlighted the problem of deciding whether for a given schema mapping there exists at least one source tree that has a solution. In this section the stronger notion of absolute consistency is considered. We say a schema mapping is *absolutely consistent* if for every source tree there exists a solution. More formally:

PROBLEM:	ABSOLUTECONSISTENCY(σ)
INPUT:	$\mathcal{M} = (D_1, D_2, \Sigma) \in \mathrm{SM}(\sigma)$
QUESTION:	is $\mathsf{Sol}_{\mathcal{M}}(T_1) \neq \emptyset$ for all T_1 with $T_1 \models D_1$?

Clearly, the notion of absolute consistency is more restrictive than the notion of consistency. For instance, Example 5.9 shows a schema mapping, which is consistent but not absolutely consistent. It turns out that even for vertical axes ABSOLUTECONSISTENCY($\Downarrow$) is significantly harder than the CONSISTENCY($\Downarrow$) problem. Whereas the CONSISTENCY($\Downarrow$) problem can be reduced to the CONSISTENCY$^\circ$($\Downarrow$) problem, a similar reduction for the absolute consistency problem does not exist. This is due to the fact that the possible number of occurrences of attribute values has to be taken into account. The following result provided by Amano et al. [4] states that the absolute consistency problem for schema mappings in SM($\Downarrow$) can be solved in double-exponential time.

Theorem 5.17. [4] *The* ABSOLUTECONSISTENCY($\Downarrow$) *problem is in EXSPACE and NEXPTIME-hard.*

We have seen that the complexity of the CONSISTENCY(σ) problem can be reduced by using nested relational DTDs. However, this alone is not sufficient for schema mappings in $\mathrm{SM}^{\mathrm{nr}}(\Downarrow)$.

Theorem 5.18. [4] *The* ABSOLUTECONSISTENCY($\Downarrow$) *problem remains NEXPTIME-hard even if its restricted to nested relational DTDs.*

To receive lower complexity bounds further restrictions on the dependencies have to be imposed. The following result shows that the absolute consistency problem is tractable for the class $\mathrm{SM}^{\mathrm{nr}}(\downarrow)$ of schema mappings, i.e if we additionally disallow the descendant axis and wildcards in stds. We denote with ABSOLUTECONSISTENCY$^{\mathrm{nr}}$(σ) the absolute consistency problem under the restriction to nested-relational DTDs.

Theorem 5.19. [4] *For fully specified schema mappings the* ABSOLUTECONSISTENCY$^{\mathrm{nr}}$($\Downarrow$) *problem is solvable in polynomial time.*

5.3 Query Answering

The most central problem in XML data exchange is answering a query posed over the target with respect to a schema mapping and a source tree. Since a

schema mapping might have infinitely many solutions for a source tree, it is not obvious how such a answer should look like. We have already encountered this issue in Section 3.3 for relational data exchange. There we considered the answer of a query over the target schema as those tuples that are contained in every solution of the schema mapping and the given source instance. The concept of these certain answers can be adapted for XML data exchange.

Definition 5.20. *Consider a schema mapping $\mathcal{M}$, a source tree T_1 and a query q. The* certain answers *of q with respect to T_1 under $\mathcal{M}$ are given by*

$$\mathsf{certain}_{\mathcal{M}}(q, T_1) = \bigcap \{q(T_2) | T_2 \in \mathsf{Sol}_{\mathcal{M}}(T_1)\}.$$

The associated decision problem asks, whether a tuple a is included in the certain answers of a query q with respect to the source tree T_1 under the schema mapping $\mathcal{M}$. More formally:

PROBLEM:	CERTAIN$(\mathcal{M}, q)$
INPUT:	a tree T_1, a tuple $\mathbf{a}$
QUESTION:	is $\mathbf{a} \in \mathsf{certain}_{\mathcal{M}}(q, T_1)$?

In the complexity analysis of the CERTAIN$(\mathcal{M}, q)$ problem we will restrict the features of queries, analogous to Section 5.1.2. Thus, $\mathbf{CTQ}(\sigma)$ and $\mathbf{UCTQ}(\sigma)$ denote the classes of queries which contain queries from **CTQ** and **UCTQ**, which only use the features in σ, where $\sigma \subseteq \{\downarrow, \downarrow^*, \rightarrow, \rightarrow^*, =, \neq, \_, \Downarrow, \Rightarrow, \sim\}$.

We begin the complexity analysis of the CERTAIN$(\mathcal{M}, q)$ problem with an upper bound, i.e. for schema mappings from SM$(\Downarrow, \Rightarrow, \sim)$ and queries from **UCTQ**. Afterwards, we show that there exist restrictions of features such that the CERTAIN$(\mathcal{M}, q)$ problem becomes tractable. We conclude this section with the result that adding more features leads quickly to intractability.

Theorem 5.21. [3] *Let $\mathcal{M}$ be a schema mapping from* SM$(\Downarrow, \Rightarrow, \sim)$*, and let q be a query from* $\mathbf{UCTQ}(\Downarrow, \Rightarrow, \sim)$*. Then the* CERTAIN$(\mathcal{M}, q)$ *problem is in coNP.*

Proof. *(Sketch)* Let the schema mapping $\mathcal{M}$ be defined by a set of stds Σ and the source and target DTDs D_1 and D_2. Moreover, let T_1 be a tree conforming to D_1. As in Section 3.3 we assume without loss of generality that q is a Boolean query.

We consider the complementary problem to show membership in coNP. The certain answer of q is false if and only if there exists a target tree T_2, such that (T_1, T_2) satisfies all dependencies in Σ, but T_2 does not satisfy q. Assume that such a tree T_2 exists. Since T_2 is possibly exponential in the size of the input, we cannot simply guess T_2 and check if T_2 satisfies Σ and dissatisfies q.

By construction, every solution satisfies each sentence of

$$\Delta = \{\psi(\mathbf{a}) \mid \varphi(\mathbf{x}, \mathbf{y}) \rightarrow \psi(\mathbf{x}) \in \Sigma \text{ and } (T_1, T_2) \models \varphi(\mathbf{a}, \mathbf{b})\}.$$

Due to Theorem 2.6, the set Δ can be computed in PTIME by iterating over all possible valuations $\mathbf{a}$ and $\mathbf{b}$. The intuition of this proof is that many of the possibly exponentially many nodes in tree T_2 can be trimmed, such that T_2 becomes polynomial in the size of T_1 and still satisfies Σ, dissatisfies q and conforms to DTD D_1. In other words, we show if T_2 exists then there exists always a polynomial sized tree which is a solution for T_1 but does not satisfy q. Therefore, we fix the nodes in T_2 that witness Δ. Consider two nodes u and v from T_2 that satisfy the same set of first order formulae. We can merge u and v and omit the nodes between them, if afterwards T_2 still conforms to DTD D_2 and all affected nodes are not witnesses of Δ. It can be shown that this cutting technique can be repeated horizontally and vertically until T_2 is polynomial. □

Alternatively, the above theorem could be shown by a reduction to the problem, where an incomplete tree I and a tuple $\mathbf{a}$ is given and the question is if $\mathbf{a}$ is a certain answer of a query q over I. Membership in coNP was shown for this problem in [15].

We are now interested in restrictions that allows to solve the $\textsc{Certain}(\mathcal{M}, q)$ problem in polynomial time. Therefore, we will discuss an algorithm, proposed by Arenas and Libkin [9], that computes in polynomial time a canonical solution $T_2^\star$ for a source tree T_1 under a fully specified schema mapping $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ over which the evaluation of an arbitrary query $q \in \mathbf{UCTQ}(\Downarrow, =)$ results in $\mathsf{certain}_{\mathcal{M}}(q, T)$. We start with some preliminary notations.

Definition 5.22. *Let e be a regular expression over an alphabet Γ and let $L(e)$ be the language described by e. Then $\mathsf{perm}(e)$ denotes the set of all permutations of the strings in $L(e)$.*

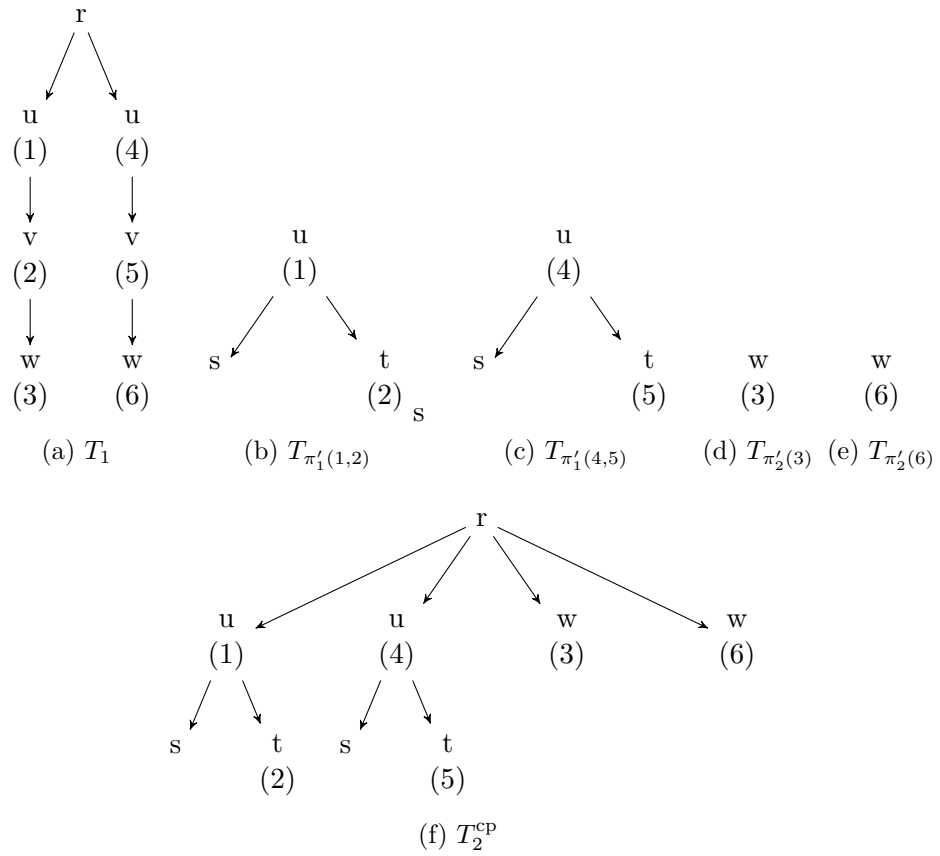
For instance, $\mathsf{perm}(ab)$ contains the string ab and ba. The next concept defines inductively the minimal tree that satisfies a tree pattern under a given variable assignment.

Definition 5.23. *Consider a tree pattern $\pi(\mathbf{x})$ and a variable assignment $\mathbf{a}$ for $\mathbf{x}$. A tree $T_{\pi(\mathbf{a})}$ is* associated *with $\pi(\mathbf{a}) = l(\mathbf{a}_0)[\pi_1(\mathbf{a}_1), \ldots, \pi_n(\mathbf{a}_n)]$, if its root node has the label l, attributes $\mathbf{a}_0$ and the subtrees $T_{\pi_1(\mathbf{a}_1)}, \ldots, T_{\pi_n(\mathbf{a}_n)}$ as children.*

The algorithm for computing the canonical tree $T_2^\star$ is composed of three components. The first component is related to the chase procedure from Section 3.2.1 and creates an intermediate solution.

Definition 5.24. *Consider a fully specified schema mapping $\mathcal{M} = (D_1, D_2, \Sigma)$ from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ as fixed and a source tree T_1 as input. DTD D_2 is defined by (r, P_D, A_D). For every std $\pi(\mathbf{x}, \mathbf{y}) \rightarrow r[\pi'_1(\mathbf{x}, \mathbf{z}), \ldots, \pi'_k(\mathbf{x}, \mathbf{z})] \in \Sigma$ and each pair of tuples $\mathbf{a}$ and $\mathbf{b}$ from Const, such that $T_1 \models \pi(\mathbf{a}, \mathbf{b})$, we introduce a tuple $\mathbf{c}$ of unused variables from Var and create k trees $T_{\pi'_1(\mathbf{a},\mathbf{c})} \ldots T_{\pi'_k(\mathbf{a},\mathbf{c})}$ associated with $\pi'_1(\mathbf{a}, \mathbf{c}), \ldots, \pi'_k(\mathbf{a}, \mathbf{c})$. The* canonical pre-solution T_2^{cp} *is a tree with root node r and all created trees $T_{\pi'_i(\mathbf{a},\mathbf{c})}$ as distinct children.*

The following example illustrates the above specified concept of canonical pre-solutions.



(a) T_1 (b) $T_{\pi_1'(1,2)}$ (c) $T_{\pi_1'(4,5)}$ (d) $T_{\pi_2'(3)}$ (e) $T_{\pi_2'(6)}$

(f) T_2^{cp}

Figure 5.4: Source tree T_1, the unordered trees $T_{\pi_1'(1,2)}$ and $T_{\pi_1'(4,5)}$ associated with $r/u(x)[s, t(y)]$, the unordered trees $T_{\pi_2'(3)}$ and $T_{\pi_2'(6)}$ associated with $r/w(z)$, and the canonical pre-solution T_2^{cp} for T_1.

Example 5.25. Consider the schema mapping $\mathcal{M} = (D_1, D_2, \Sigma)$ from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$. The DTDs $D_1 = \langle r, P_{D_1}, A_{D_1}\rangle$ and $D_2 = \langle r, P_{D_2}, A_{D_2}\rangle$ are defined as follows:

$$
\begin{aligned}
P_{D_1} = \{&r \to u^* && A_{D_1}(r) = \emptyset\\
&u \to v && A_{D_1}(u) = \{@o\}\\
&v \to w && A_{D_1}(v) = \{@p\}\\
&w \to \epsilon\} && A_{D_1}(w) = \{@q\}
\end{aligned}
$$

$$
\begin{aligned}
P_{D_2} = \{&r \to u^* v w^* && A_{D_2}(r) = \emptyset\\
&u \to st && A_{D_2}(u) = \{@o\}\\
&v \to \epsilon && A_{D_2}(v) = \emptyset\\
&w \to \epsilon && A_{D_2}(w) = \{@q\}\\
&s \to \epsilon && A_{D_2}(s) = \{@m\}\\
&t \to \epsilon\} && A_{D_2}(t) = \{@p\}
\end{aligned}
$$

The dependencies in Σ are given by two stds defined as

$$r/u(x)/v(y) \rightarrow r/u(x)[s, t(y)]$$
$$r/u/v/w(z) \rightarrow r/w(z).$$

Suppose that T_1 is the tree conforming to D_1 depicted in Figure 5.4 a). We construct now the canonical pre-solution for T_1. Therefore, we have to create for each variable assignment $\mathbf{a}$ that satisfies the left-hand side of an std, in our case, a single intermediate tree $T_{\pi'(\mathbf{a})}$, such that the right-hand side of the std is satisfied under $\mathbf{a}$ by the tree with root node r and the single child $T_{\pi'(\mathbf{a})}$. This step results in four of such intermediate trees $T_{\pi'_1(1,2)}$, $T_{\pi'_1(4,5)}$, $T_{\pi'_2(3)}$ and $T_{\pi'_2(6)}$ depicted in Figure 5.4 b), c), d) and e), respectively. Figure 5.4 f) shows the canonical pre-solution T_2^{cp} obtained by subsuming all created trees under a single root node r. Note that although T_2^{cp} satisfies all dependencies in Σ, it does not conform to D_2. □

It is easy to see that canonical pre-solutions can be computed in polynomial time. As already demonstrated in Example 5.25, a canonical pre-solution does not necessarily conform to the target DTD. We show now how a canonical pre-solution can be transformed into a canonical tree by repairing all constraint violations of the target DTD. Therefore, we have to deal with two sorts of violations in canonical pre-solutions. In particular, a node of a canonical pre-solution T_2^{cp} represents a violation of DTD D_2, either because its attributes are defined differently in the DTD, or its children do not have the appropriate types. More formally, node s of T_2^{cp} violates D_2 if $\{a \mid \rho_a(s) \text{ is defined in } T_2^{\text{cp}}\} \neq A_D(\mathsf{lab}(s))$ or if $\mathsf{lab}(\mathsf{children}(s)) \notin \mathsf{perm}(P_D(\mathsf{lab}(s)))$, where $\mathsf{children}(s)$ is a function returning the node ids of the children of s. We will define the functions $\mathsf{ChangeAtt}(D_2, T, s)$ and $\mathsf{ChangeReg}(D_2, T, s)$, which iteratively repair one violation at the time until T_2^{cp} conforms to D_2. Prior to that we continue Example 5.25 and illustrate both kinds of violations.

Example 5.26. Recall Example 5.25. There we created a canonical pre-solution T_2^{cp}, depicted in Figure 5.4 f), for a schema mapping $\mathcal{M} = (D_1, D_2, \Sigma)$ and a source tree T_1. Tree T_2^{cp} does not conform to D_2. DTD D_2 requires that nodes labeled with s have an attribute $@m$, but the nodes in T_2^{cp} labeled with s have no attribute. Moreover, DTD D_2 specifies that a the root node must have a child node labeled v. □

The function $\mathsf{ChangeAtt}(D_2, T, s)$ attempts to repair the attributes of a node. If the node has missing attributes, then the function defines them. If the node has on the other hand an attribute that the DTD does not specify, then the function fails. More formally:

Definition 5.27. *Function* $\mathsf{ChangeAtt}(D, T, s)$ *has as input parameter a DTD* $D = (r, P_D, A_D)$*, a tree* T *and a node id* s*, such that* $\{a \mid \rho_a(s)$ is defined in $T\} \neq A_D(\mathsf{lab}(s))$*. If there is an attribute* a*, such that* $\rho_a(s)$ *is defined in* T *but* $a \notin A_D(\mathsf{lab}(s))$*, then the function fails. Otherwise, for every* a*, such that* $\rho_a(s)$ *is undefined and* $a \in A_D(\mathsf{lab}(s))$*, define* $\rho_a(s) = v$*, where* v *is a unused element from* Var*.*

The function $\mathsf{ChangeReg}(D_2, T, s)$ is applied if the types of the children of a node s do not coincide with the regular expressions from a DTD. If the violations are not reparable then the function fails. Otherwise, it returns a modified tree, such that $\mathsf{lab}(\mathsf{children}(s)) \in \mathsf{perm}(P_D(\mathsf{lab}(s)))$. Before $\mathsf{ChangeReg}(D_2, T, s)$ can be specified, we have to fix some terminology and motivate how violations are handled.

Suppose that w is either a string over symbols from $\Gamma \subset \mathit{Labels}$. The set of symbols mentioned in w is denoted with $\mathsf{alph}(w)$. Analogously, $\mathsf{alph}(e)$ gives the set of different symbols used in a regular expression e. The number of occurrences of a symbol $b \in \mathsf{alph}(w)$ in w is given by $\#_b(w)$. We denote with $w \preceq w'$ if $\#_b(w) \leq \#_b(w')$ holds for every $a \in \mathsf{alph}(w)$. Furthermore, we need a preference relation that allows us to compare replacements w_1, w_2 for w. We prefer w_2 to w_1 if w_2 excludes fewer symbols from w than w_1 and if w_2 adds less new symbols to w than w_1. More formally, $w_1 \preceq_w w_2$ if $\#_b(w_2) \geq \min\{\#_b(w_1), \#_b(w)\}$ for all $b \in \mathsf{alph}(w)$ and $\mathsf{alph}(w_2) \backslash \mathsf{alph}(w) \subseteq \mathsf{alph}(w_1) \backslash \mathsf{alph}(w)$ holds. Definition 5.28 specifies, for a string w and a regular expression e, those elements from $\mathsf{perm}(e)$ with the smallest number of different symbols.

Definition 5.28. *Consider a string w and a regular expression e, such that $\mathsf{alph}(w) \subseteq \mathsf{alph}(e)$ and $w \notin \mathsf{perm}(e)$. The set of* preferred repairs *of w, denoted as $\mathsf{rep}(w, e)$, is given by*

$$\mathsf{rep}(w, e) = \max_{\preceq_w} \bigcup_{w' \preceq w, \mathsf{alph}(w') = \mathsf{alph}(w)} \min_{\preceq}\{w' \mid w' \in \mathsf{perm}(e), w \preceq w'\}.$$

We emphasize now the intuition of the above introduced concept of preferred repairs. Therefore, the expression of $\mathsf{rep}(w, e)$ is surveyed from right to left. We start with

$$\mathsf{minext}(w, e) = \min_{\preceq}\{w' \mid w' \in \mathsf{perm}(e), w \preceq w'\}.$$

The set $\mathsf{minext}(w, e)$ refers to the minimal extensions of w, i.e. those $w' \in \mathsf{perm}(e)$ with substring w that contain the fewest additional symbols. The minimal extensions define how to repair strings with too few symbols of a particular kind. For example, $\mathsf{rep}(a, aab^+) = \{aab, aba, baa\}$. On the other hand, if a symbol occurs too many times, then the set of minimal extensions does not provide a repair. Consider therefore the example $\mathsf{rep}(aaabb, aab^+) = \emptyset$. To repair such a case, we have to exclude some of those symbols that occur too often. This can be done by considering minimal extensions of a substring of w, i.e. a string w' such that $w' \preceq w$ and $\mathsf{alph}(w') = \mathsf{alph}(w)$ holds. This behavior is captured with

$$\bigcup_{w' \preceq w, \mathsf{alph}(w') = \mathsf{alph}(w)} \min_{\preceq}\{w' \mid w' \in \mathsf{perm}(e), w \preceq w'\}.$$

In the above example the minimal extensions for the substrings ab, aab, $aaab$, abb, abb, $aabb$, and $aaabb$ are given by $\{aab, aba, baa, aabb, abab, baab, abba, baba, bbaa\}$. We prefer from this set those repairs that exclude as few known

symbols and introduce as few new symbols as possible, i.e. $\{aabb, abab, baab, abba, baba, bbaa\}$. This is captured with the $\max_{\preceq_w}$ expression. The $\mathsf{ChangeReg}(D, T, s)$ function is defined as follows:

Definition 5.29. *Function* $\mathsf{ChangeReg}(D, T, s)$ *receives as input parameters a DTD* D*, a tree* T *and a node id* s*, such that* $\mathsf{lab}(\mathsf{children}(s)) \notin \mathsf{perm}(P_D(\mathsf{lab}(s)))$*. Suppose that* $w = \mathsf{lab}(\mathsf{children}(s))$ *and* $e = P_D(\mathsf{lab}(s))$*. The function fails if* $\mathsf{rep}(w, e)$ *is empty.*

Otherwise, the function chooses an arbitrary string $w' \in \mathsf{rep}(w, e)$ *and transforms the children of* s *according to* w' *as follows. For every* $b \in \mathsf{alph}(e)$*, let* $p = \#_b(w)$ *and let* $q = \#_b(w')$*. If* $p < q$ *then* $(q-p)$ *new children labeled with* b *without attributes and subtrees are added to* s*. If* $p > q$*, then the* p *children labeled with* b *are replaced by a single fresh node* s' *with label* b*. Moreover, every subtree of the replaced nodes* $s_1, \ldots, s_p$ *is attached to* s'*.*

The function $\mathsf{ChangeReg}(D, T, s)$ *fails, if there are two nodes with subtrees* $s_i, s_j \in s_1, \ldots, s_p$ *which have an attribute* a*, where* $\rho_a(s_i) = \mathsf{Const}$*,* $\rho_a(s_j) = \mathsf{Const}$ *and* $\rho_a(s_i) \neq \rho_a(s_j)$*. If not, then* $\mathsf{ChangeReg}(D, T, s)$ *returns* T*.*

Observe, $\mathsf{ChangeReg}(D, T, s)$ replaces the nodes $s_1, \ldots, s_p$ by a single node s' having no attributes and no children if $p > q$. Thus, node s' can itself violate constraints from the target DTD again, which are resolved by applying the functions $\mathsf{ChangeAtt}(D, T, s')$ and $\mathsf{ChangeReg}(D, T, s')$ on s'. The construction of the canonical pre-solution, $\mathsf{ChangeAtt}(D, T, s)$ and $\mathsf{ChangeReg}(D, T, s)$ are the three components needed for specifying the algorithm that computes the canonical tree for a schema mapping and a source tree.

Definition 5.30. *The algorithm* $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ *starts by computing a canonical pre-solution* T_2^{cp} *for the fully specified schema mapping* $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ *and the source tree* T_1*. Afterwards, the functions* $\mathsf{ChangeAtt}(D_2, T_2^{\mathrm{cp}}, s)$ *and* $\mathsf{ChangeReg}(D_2, T_2^{\mathrm{cp}}, s)$ *are applied in an arbitrary order until either the resulting tree* $T^\star$ *conforms to the target DTD* D_2*, or one of the function calls fails, in which case also* $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ *fails.*

The following example concludes our running example started at Example 5.25 and illustrates the application of the $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ algorithm.

Example 5.31. Example 5.25 presented a schema mapping $\mathcal{M} = (D_1, D_2, \Sigma)$ and a source tree T_1, shown in Figure 5.4 a). Moreover, it was demonstrated how the canonical pre-solution T_2^{cp} is constructed. As already mentioned, the canonical pre-solution T_2^{cp}, depicted in Figure 5.4 f), does not conform to DTD D_2. Example 5.26 has located in T_2^{cp} two kinds of violations of the constraints imposed by D_2, namely nodes with label s do not have an attribute and the root node has no node with label v.

We discuss now the further application of the $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ algorithm. Let u_1 and u_2 be the node ids of the children with label s of the nodes with attribute "1" and "4", respectively. The function calls $\mathsf{ChangeAtt}(D_2, T_2^{\mathrm{cp}}, u_1)$ and $\mathsf{ChangeAtt}(D_2, T_2^{\mathrm{cp}}, u_2)$ add to u_1 and u_2 an attribute $@m$

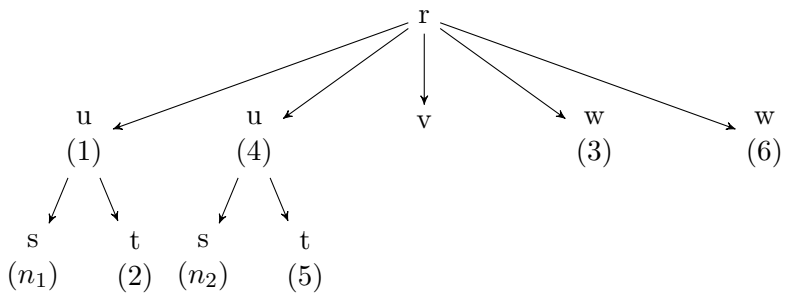


Figure 5.5: The canonical solution $T_2^\star$ for the source tree T_1 shown in Figure 5.4 a)

with the corresponding value n_1 and n_2, where $n_1, n_2 \in \mathsf{Var}$. Moreover, let u_3 be the node id of the root node. The application of the $\mathsf{ChangeReg}(D, T_2^{\text{cp}}, u_3)$ function adds a new child with label v to the root node. Since there are no violations left, the algorithm terminates successfully and thus the resulting XML tree, depicted in Figure 5.5 and denoted as $T_2^\star$, conforms to D_2. Furthermore, $T_2^\star$ is the canonical solution for T_1 and $\mathcal{M}$. □

It remains to show the correctness of the $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ algorithm. Observe that the above algorithm is essentially the chase. Accordingly, the following Lemma is the analog to Theorem 3.27, Lemma 3.24 and Theorem 3.26 from Section 3.2.1.

Lemma 5.32. [9] *Let $\mathcal{M} = (D_1, D_2, \Sigma)$ be a schema mapping from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ and let T_0' be the canonical pre-solution for the source tree T_1. Moreover, let $T_0', \ldots, T_n'$ be a sequence of trees created by applying $\mathsf{ChangeAtt}(D_2, T_{i-1}', s)$ and $\mathsf{ChangeReg}(D_2, T_{i-1}', s)$ until one of them fails or both cannot be applied anymore.*

(1) The sequence $T_0', \ldots, T_n'$ is finite.

(2) There exists a homomorphism $h_i : T_i' \to T_2$ for every solution $T_2 \in \mathsf{Sol}_{\mathcal{M}}(T_1)$ and for every $i \in [0, n]$.

(3) If either $\mathsf{ChangeAtt}(D_2, T_n', s)$ or $\mathsf{ChangeReg}(D_2, T_n', s)$ can be applied but fails, then there exists no solution for T_1 under $\mathcal{M}$.

Theorem 5.33 states that if there are solutions for a source tree and a schema mapping, then one of them is canonical.

Theorem 5.33. [9] *Suppose that $\mathcal{M}$ is a fully specified schema mapping from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ and that T_1 is a source tree. If there exists a solution $T_2 \in \mathsf{Sol}_{\mathcal{M}}(T_1)$, then there exists also a canonical solution $T_2^\star$ for T_1.*

Proof. Assume there is a tree $T_2 \in \mathsf{Sol}_{\mathcal{M}}(T_1)$, but there exists no canonical solution. This means that at some point $\mathsf{ChangeAtt}(D_2, T', s)$ or $\mathsf{ChangeReg}(D_2, T', s)$ fails. Lemma 5.32 states that in such a case $\mathsf{Sol}_{\mathcal{M}}(T_1)$ is empty, which is a contradiction. □

The following theorem continues Theorem 5.33 and shows that the $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ algorithm is correct, i.e. it results truly in a canonical solution if one exists.

Theorem 5.34. [9] *Let $\mathcal{M}$ be a fully specified schema mapping from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ and let T_1 be a source tree. The $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ procedure computes in polynomial time a canonical solution or fails if none exists.*

Proof. *(Sketch)* Let $\mathcal{M}$ be defined by the source DTD D_1, the target DTD D_2, and by a set of stds Σ. The $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ procedure is composed of tree components, namely the construction of the pre-solution and the functions $\mathsf{ChangeAtt}(D, T, s)$ and $\mathsf{ChangeReg}(D, T, s)$. Clearly, the pre-solution as well as the function $\mathsf{ChangeAtt}(D, T, s)$ can be computed in polynomial time. Showing tractability for the $\mathsf{ChangeReg}(D, T, s)$ function comes down to demonstrating, whether the problem of testing $\mathsf{rep}(w, e) \neq \emptyset$ is tractable, and if it is, whether a string $w' \in \mathsf{rep}(w, e)$ can be computed in polynomial time.

The $\mathsf{CanonicalTree}(\mathcal{M}, T_1)$ procedure starts by computing a canonical pre-solution T'_0. By applying $\mathsf{ChangeAtt}(D_2, T'_{i-1}, s)$ and $\mathsf{ChangeReg}(D_2, T'_{i-1}, s)$ in a depth-first search manner we compute a sequence of trees $T'_0, \ldots, T'_n$, such that neither $\mathsf{ChangeAtt}(D_2, T'_{i-1}, s)$ nor $\mathsf{ChangeReg}(D_2, T'_{i-1}, s)$ can be applied to T'_n, or an application of one of them would fail. In the latter case we have that $T'_n \not\models D_2$, and due to Lemma 5.32 we know that there is no solution for T_1 under $\mathcal{M}$, and thus, there exists also no canonical solution. If in contrast $T'_n \models D_2$, then T'_n is the canonical solution. □

The next theorem shows that canonical solutions can indeed be used for computing the certain answers.

Theorem 5.35. [9] *Consider a fully specified schema mapping $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$, a source tree T_1, a query $q \in \mathbf{UCTQ}(\Downarrow, =)$, and a tuple $\mathbf{a}$ from Const. Moreover, assume there exists a canonical solution $T_2^\star$ for T_1. Then $\mathbf{a} \in \mathsf{certain}_{\mathcal{M}}(q, T_1)$ if and only if $T_2^\star \models q(\mathbf{a})$.*

Proof. Assume that $T_2^\star \models q(\mathbf{a})$. Let T_2 be an arbitrary solution from $\mathsf{Sol}_{\mathcal{M}}(T_1)$. We know from Lemma 5.32 that there exists a homomorphism $h : T_2^\star \to T_2$. It is easy to see that if a tree T satisfies a tree pattern π and there is a homomorphism from T to T', then also T' satisfies π. It follows that $T_2 \models q(\mathbf{a})$, and since T_2 is an arbitrary solution, we have that $T_2 \models q(\mathbf{a})$ holds for every solution $T_2 \in \mathsf{Sol}_{\mathcal{M}}(T_1)$. We conclude that $\mathbf{a} \in \mathsf{certain}_{\mathcal{M}}(q, T_1)$.

Assume that $T_2^\star \not\models q(\mathbf{a})$. Since $T_2^\star$ is a solution for T_1 under $\mathcal{M}$, we have that $\mathbf{a} \notin \mathsf{certain}_{\mathcal{M}}(q, T_1)$. □

The following result concludes the discussion of the algorithm provided by Arenas and Libkin [9] and immediately follows by Theorem 5.34 and Theorem 5.35.

Corollary 5.36. [9] *Suppose that $\mathcal{M}$ is a fully specified schema mapping from $\mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ and that q is a query from $\mathbf{UCTQ}(\Downarrow, =)$. Then the $\textsc{Certain}(\mathcal{M}, q)$ problem can be solved in polynomial time.*

Arenas and Libikin [9] and Amano et al. [3] made several attempts to find a larger class of schema mappings or query classes such that the $\textsc{Certain}(\mathcal{M}, q)$ problem remains tractable. It has turned out that this problem is highly non-trivial. Theorem 5.37 shows various coNP-hardness results obtained by allowing further features in the schema mappings or queries compared to Corollary 5.36. Recall, that Theorem 5.21 provided an upper bound for $\textsc{Certain}(\mathcal{M}, q)$.

Theorem 5.37. [9, 4] *The $\textsc{Certain}(\mathcal{M}, q)$ problem is coNP-complete if*

(1) $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\Downarrow, =)$ *and* $q \in \mathbf{CTQ}(\Downarrow, =)$,

(2) $\mathcal{M} \in \mathrm{SM}(\Downarrow, =)$ *is fully specified and* $q \in \mathbf{CTQ}(\Downarrow, =)$,

(3) $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow, =)$ *is fully specified and* $q \in \mathbf{CTQ}(\downarrow, =, \neq)$,

(4) $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow)$ *is fully specified and* $q \in \mathbf{CTQ}(\downarrow, \rightarrow, =)$,

(5) $\mathcal{M} \in \mathrm{SM}^{\mathrm{nr}}(\downarrow)$ *is fully specified and* $q \in \mathbf{CTQ}(\downarrow, \rightarrow^*, =)$,

CHAPTER 6

Conclusion

We have studied data exchange over relational data, knowledge bases and XML data. The corresponding schema mappings are constructed in such a way that knowledge exchange as well as XML data exchange represented generalizations of the relational data exchange. Thus, the complexity results in the relational case represent a lower complexity bound for the corresponding problems in knowledge exchange and XML data exchange. This allowed us to exclude problem settings that have already been shown to be undecidable in the relational data scenario from further investigation under the other two data models.

The first issue that has to be dealt with, is to find a proper logical formalism that allows efficient algorithms, but is also powerful enough to express meaningful queries. We highlighted this investigation for the relational data exchange problem. As anticipated, dependencies defined by unrestricted first order rules showed undecidability results even for the simplest problems. A promising attempt to find the right formalism was to base the dependencies on conjunctive queries, but nevertheless undecidability results could be proven for basic problems in the presence of target dependencies. The further restriction of weakly acyclic target dependencies provided then the preferred properties.

The most crucial computational problem concerning data exchange is to verify whether a target instance is a solution for a source instance. The discussion of finding suitable dependencies for relational data exchange implied that a solution can be verified in polynomial time. The complexity is much higher for knowledge exchange, since a knowledge base can represent infinitely many relational instances. Hence, the complexity of verifying a knowledge base solution rises to $\Delta_2^P[O(\log n)]$. The problem of verifying solutions in XML data exchange is even for the full structure of XML documents well behaved and is solvable in LOGSPACE.

Clearly, the main task in data exchange is to materialize a solution for a given source instance. The fact that there are in general infinitely many solutions

gives rise to the question whether there are preferred solutions. In relational data exchange such solutions exist and are called universal solutions. They are defined as those solutions that have a homomorphism to every other solution. Universal solutions can be computed in polynomial time with the well-known chase procedure. The concept of universality can also be applied to knowledge base solutions, but has the disadvantage that they prohibit implicit knowledge, and thus, universal knowledge base solutions degenerate to relational solutions. More sophisticated are minimal knowledge base solutions which have similar positive properties as universal knowledge base solutions, but also allow implicit knowledge. On the other hand, for XML data exchange there exists in general no single solution that is representative for all other solutions. Nevertheless, we have seen an algorithm that materializes canonical solutions in our discussion of query answering for restricted XML schema mappings.

We looked next at the problem of answering queries posed over the target schema with respect to a source instance. The semantics are given by the notion of certain answers, that defined the result of a query as those tuples which occur in the answers for all solutions. In relational data exchange the certain answers of a union of conjunctive queries can be computed over the universal solution in polynomial time. As soon as we added inequalities to the conjunctive queries the complexity increased to coNP-completeness. In knowledge exchange the certain answers of queries can be directly computed over universal knowledge base solutions. Computing the certain answers in XML data exchange has an upper bound in coNP for schema mappings reasoning over the full structure of XML trees. We showed a tractable algorithm that computes for restricted schema mappings a canonical tree, over which the certain answers for unions of conjunctive tree queries can be calculated.

The last studied fundamental issue regarding data exchange was their composition. The associated problem is the composition query, which asks if a pair of instances is contained in the composition of two schema mappings. It turns out that the composition of two schema mappings cannot always be defined with the usual first-order dependencies for relational data exchange and knowledge exchange, and hence, the composition query is NP-complete. We introduced alternative dependencies based on second order logic, that possess good properties for data exchange and allow to specify the composition of two schema mappings in polynomial time. We did not discuss composition for XML data exchange but it has already been studied by [4].

Data exchange has become one of the central problems in database theory. This master thesis aimed to assess the state of the art of data exchange on the basis of three different data models. It has shown that while relational data exchange is already well understood, there are still many open questions in knowledge exchange and XML data exchange. Nevertheless, the discussion of exchanging incomplete data has shown that there are assumptions in data exchange that can and should be questioned. Moreover, with the fast change of technologies there will most likely emerge new data models that require the exchange of data.

Bibliography

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of Answering Queries Using Materialized Views. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '98)*, pages 254–263. ACM, 1998.

[2] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the Representation and Querying of Sets of Possible Worlds. In *Proceedings of the Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference (SIGMOD '87)*, pages 34–48. ACM, 1987.

[3] Shun'ichi Amano, Claire David, Leonid Libkin, and Filip Murlak. On the Tradeoff between Mapping and Querying Power in XML Data Exchange. In *Proceedings of the 13th International Conference on Database Theory (ICDT '10)*, pages 155–164. ACM, 2010.

[4] Shun'ichi Amano, Leonid Libkin, and Filip Murlak. XML schema mappings. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '09)*, pages 33–42. ACM, 2009.

[5] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Tree pattern query minimization. *The VLDB Journal*, 11(4):315–331, December 2002.

[6] Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} Worlds and Beyond: Efficient Representation and Processing of Incomplete Information. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE'07)*, pages 606–615. IEEE, 2007.

[7] Marcelo Arenas, Elena Botoeva, and Diego Calvanese. Knowledge Base Exchange. In *Proceedings of the 24th International Workshop on Description Logics (DL '11)*, volume 745 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.

[8] Marcelo Arenas, Elena Botoeva, Diego Calvanese, Vladislav Ryzhikov, and Evgeny Sherkhonov. Exchanging Description Logic Knowledge Bases. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 13th International Conference (KR '12)*. AAAI Press, 2012.

[9] Marcelo Arenas and Leonid Libkin. XML Data Exchange: Consistency and Query Answering. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '05)*, pages 13–24. ACM, 2005.

[10] Marcelo Arenas, Jorge Pérez, and Juan L. Reutter. Data Exchange beyond Complete Data. In *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '11)*, pages 83–94. ACM, 2011.

[11] Marcelo Arenas, Jorge Pérez, Juan L. Reutter, and Cristian Riveros. Foundations of Schema Mapping Management. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '10)*, pages 227–238. ACM, 2010.

[12] Marcelo Arenas, Jorge Pérez, and Cristian Riveros. The Recovery of a Schema Mapping: Bringing Exchanged Data Back. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '08)*, pages 13–22. ACM, 2008.

[13] Denilson Barbosa, Laurent Mignet, and Pierangelo Veltri. Studying the XML Web: Gathering Statistics from an XML Sample. *World Wide Web*, 8(4):413–438, December 2005.

[14] Pablo Barceló. Logical Foundations of Relational Data Exchange. *SIGMOD Record*, 38(1):49–58, June 2009.

[15] Pablo Barceló, Leonid Libkin, Antonella Poggi, and Cristina Sirangelo. XML with Incomplete Information: Models, Properties, and Query Answering. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '09)*, pages 237–246. ACM, 2009.

[16] Pablo Barceló, Jorge Pérez, and Juan L. Reutter. Schema Mappings and Data Exchange for Graph Databases. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings*, pages 189–200. ACM, 2013.

[17] Catriel Beeri and Moshe Y. Vardi. A Proof Procedure for Data Dependencies. *Journal of the ACM*, 31(4):718–741, September 1984.

[18] Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '05)*, pages 25–36. ACM, 2005.

[19] Philip A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Online Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR '03)*. www.cidrdb.org, 2003.

[20] Henrik Björklund, Wim Martens, and Thomas Schwentick. Conjunctive Query Containment over Trees. In *Proceedings of the 11th International*

Symposium on Database Programming Languages (DBPL '07), volume 4797 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007.

[21] Henrik Björklund, Wim Martens, and Thomas Schwentick. Optimizing Conjunctive Queries over Trees Using Schema Information. In *Proceedings of the 33rd International Symposium Mathematica onl Foundations of Computer Science (MFCS '08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 132–143. Springer, 2008.

[22] Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '06)*, pages 10–19. ACM, 2006.

[23] Mikolaj Bojanczyk, Leszek Aleksander Kolodziejczyk, and Filip Murlak. Solutions in XML data exchange. In *Proceedings of the 14th International Conference on Database Theory (ICDT '11)*, pages 102–113. ACM, 2011.

[24] Elena Botoeva. Description Logic Knowledge Base Exchange. In *Web Reasoning and Rule Systems - 6th International Conference (RR '12)*, volume 7497, pages 266–271. Springer, 2012.

[25] Samuel R. Buss and Louise Hay. On Truth-Table Reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.

[26] Edgar F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[27] Edgar F. Codd. Relational Completeness of Data Base Sublanguages. *In: R. Rustin (ed.): Database Systems: 65-98, Prentice Hall and IBM Research Report RJ 987*, 1972.

[28] Edgar F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[29] Claire David. Complexity of Data Tree Patterns over XML Documents. In *Proceedings of the 33rd International Symposium on Mathematical Foundations of Computer Science (MFCS '08)*, volume 5162 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2008.

[30] Claire David, Leonid Libkin, and Filip Murlak. Certain Answers for XML Queries. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '10)*, pages 191–202. ACM, 2010.

[31] Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The Chase Revisited. In *Proceedings of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '08)*, pages 149–158. ACM, 2008.

[32] Alin Deutsch and Val Tannen. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Proceedings of 8th International Workshop on Database Programming Languages (DBPL '01)*, volume 2397 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2001.

[33] Alin Deutsch and Val Tannen. Reformulation of XML Queries and Constraints. In *Proceedings of the 9th International Conference on Database Theory (ICDT '03)*, volume 2572 of *Lecture Notes in Computer Science*, pages 225–241. Springer, 2003.

[34] Trevor Evans. Embeddability and the Word Problem. *Journal LMS*, 28:76–80, 1953.

[35] Ronald Fagin. Horn Clauses and Database Dependencies. In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC '80)*, pages 123–134. ACM, 1980.

[36] Ronald Fagin. Inverting Schema Mappings. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '06)*, pages 50–59. ACM, 2006.

[37] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data Exchange: Semantics and Query Answering. In *Proceedings of the 9th International Conference on Database Theory (ICDT '03)*, volume 2572 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2003.

[38] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa. Data Exchange: Getting to the Core. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '03)*, pages 90–101. ACM, 2003.

[39] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Composing Schema Mappings: Second-Order Dependencies to the Rescue. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '04)*, pages 83–94. ACM, 2004.

[40] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Quasi-inverses of Schema Mappings. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '07)*, pages 123–132. ACM, 2007.

[41] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. Reverse Data Exchange: Coping with Nulls. In *Proceedings of the 28th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '09)*, pages 23–32. ACM, 2009.

[42] Wenfei Fan and Leonid Libkin. On XML Integrity Constraints in the Presence of DTDs. In *Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '01)*. ACM, 2001.

[43] Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. On Reconciling Data Exchange, Data Integration, and Peer Data Management. In *Proceedings of the 26th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '07)*, pages 133–142. ACM, 2007.

[44] Georg Gottlob. Computing Cores for Data Exchange: New Algorithms and Practical Solutions. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '05)*, pages 148–159. ACM, 2005.

[45] Georg Gottlob and Alan Nash. Data Exchange: Computing Cores in Polynomial Time. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '06)*, pages 40–49. ACM, 2006.

[46] Yuri Gurevich. The Word Problem for Certain Classes of Semigroups. *Algebra and Logic*, 5:25–35, 1966.

[47] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema Mediation in Peer Data Management Systems. In *Proceedings of the 19th International Conference on Data Engineering (ICDE '03)*, pages 505–516. IEEE Computer Society, 2003.

[48] Jan Hidders. Satisfiability of XPath Expressions. In *Proceedings of the 9th International Symposium on Database Programming Languages (DBPL '03)*, volume 2921 of *Lecture Notes in Computer Science*, pages 21–36. Springer, 2003.

[49] Tomasz Imielinski and Witold Lipski Jr. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.

[50] Phokion G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '05)*, pages 61–75. ACM, 2005.

[51] Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan. The Complexity of Data Exchange. In *Proceedings of the 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '06)*, pages 30–39. ACM, 2006.

[52] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '02)*, pages 233–246. ACM, 2002.

[53] Jayant Madhavan and Alon Y. Halevy. Composing Mappings Among Data Sources. In *Proceedings of 29th International Conference on Very Large Data Bases (VLDB '03)*, pages 572–583. VLDB Endowment, 2003.

[54] Aleksander Madry. Data exchange: On the complexity of answering queries with inequalities. *Information Processing Letters*, 94(6):253–257, 2005.

[55] David Maier, Jeffrey D. Ullman, and Moshe Y. Vardi. On the Foundations of the Universal Relation Model. *ACM Transactions on Database Systems*, 9(2):283–308, 1984.

[56] Wim Martens, Frank Neven, and Thomas Schwentick. Simple off the shelf abstractions for XML schema. *SIGMOD Record*, 36(3):15–22, 2007.

[57] Michael Meier, Michael Schmidt, and Georg Lausen. Stop the Chase: Short Contribution. In *Proceedings of the 3rd Alberto Mendelzon International Workshop on Foundations of Data Management (AMW '09)*, volume 450 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[58] Alan Nash, Philip A. Bernstein, and Sergey Melnik. Composition of Mappings Given by Embedded Dependencies. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '05)*, pages 172–183. ACM, 2005.

[59] David A. Plaisted. Complete Problems in the First-Order Predicate Calculus. *Journal of Computer and System Sciences*, 29(1):8–35, 1984.

[60] Luc Segoufin. Typing and querying XML documents: some complexity bounds. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '03)*, pages 167–178. ACM, 2003.

[61] Luc Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Proceedings of the 20th International Workshop on Computer Science Logic (CSL '06)*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006.

[62] Nan C. Shu, Barron C. Housel, Robert W. Taylor, Sakti P. Ghosh, and Vincent Y. Lum. EXPRESS: A Data EXtraction, Processing, amd RE-Structuring System. *ACM Transactions on Database Systems*, 2(2):134–174, 1977.

[63] Jeffrey D. Ullman. *Principles of Database Systems, 2nd Edition*. Computer Science Press, 1982.

[64] Ron van der Meyden. Logical Approaches to Incomplete Information: A Survey. In *Logics for Databases and Information Systems*, pages 307–356. Kluwer, 1998.

[65] Moshe Y. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC '82)*, pages 137–146. ACM, 1982.

[66] Klaus W. Wagner. More Complicated Questions About Maxima and Minima, and Some Closures of NP. *Theoretical Computer Science*, 51:53–80, 1987.

[67] Klaus W. Wagner. Bounded Query Classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.