Theoretical and Practical Methods for Consistent Query Answering in the Relational Data Model

Fabian Pijcke

fabian@pijcke.net

Acknowledgments

This document is the outcome of five years of work. During all that time, Jef Wijsen has been trusting me and encouraging me. In the end I'm very proud of the result, and all the words I could write here would not express enough how grateful I am for his support and constant motivation. Thank you, Jef!

When this project started, we were engaged, now we are married for better and for worse, but especially the better. Thank you Alexia for your happiness, your positiveness and also your diligent proofreading in the end. Thank you for making everything else than the thesis itself run smoothly. All this would have meant nothing without you.

Pierre Hauweele have taken the role of support during this adventure. Not only at the scientific level, but also for lending me his guest room so many times that I lost track of the counting, and for always being available when I needed help. Thank you so much, Pierre!

To my friends who've been patient all these years, and yet always present when I needed to take a break! Romain, Kelly, Émilie, Noémie, Romuald, Jonathan, Dany, thank you!

I have met many brilliant minds who never hesitated to share ideas, techniques, and advices. Thanks to Alexandre Decan, who shared his desk with me, Enela Pema, Sergio Greco, Paraschos Koutris, and the members of my jury; Floris Geerts, Stijn Vansummeren, Christian Michaux and Olivier Delgrange.

I would also like to address special thanks to one of my professors in college, Philippe Tilleuil, who transmitted to me his passion for mathematics.

The last sentences are addressed in French to my family. Chers parents, frères, sœurs, parrain, marraine, mamy, oncles, tantes, tante (qui se reconnaîtra !), vous avez fait de moi qui je suis aujourd'hui. Je suis très fier de mon parcours et vous doit tout ceci ! Merci !

Contents

1 Introduction **9**

2 Consistent Query Answering **17**

2.1 Preliminaries 17

2.1.1 Databases 17

2.1.2 Queries 18

2.1.3 Domain Relational Calculus 19

2.1.4 Tuple Relational Calculus 20

2.1.5 Constraints 22

2.1.6 Primary Keys 22

2.1.7 Functional dependencies 23

2.1.8 Inclusion Dependencies 25

2.1.9 Join Dependencies 25

2.2 Data Cleaning 26

2.3 Repairs 27

2.3.1 Inclusion-Based Repairs 28

2.3.2 Symmetric Difference-Based Repairs 29

2.3.3 Cardinality-Based Repairs 29

2.3.4 Tuple-Based or Value-Based 30

2.4 Consistent Query Answering 30

3 CQA in FO **33**

3.1 The problem **CERTAINTY**($\mathcal{Q}$) 33

3.2 Attack Graphs 40
3.3 Rewriting Function 44
3.3.1 Expected input/outputs 44
3.3.2 Function Rewrite 45
3.3.3 Function RewAtom 46
3.3.4 Function RewEmptyKey 47
3.3.5 Examples 48
3.4 Related Work 50

4 Presence of Satisfied Constraints 53
4.1 Motivation 53
4.2 Problem Statement 57
4.3 Extending Attack Graph 59
4.4 Construction of Consistent First-Order Rewritings 69
4.4.1 Attack Graph of $(\mathcal{Q}, \Sigma)$ 69
4.4.2 Free Variables 72
4.4.3 The Function $\mathbf{rewrite}^{\Sigma}(\mathcal{Q})$ 73
4.5 Conclusion 80

5 Under-Approximations of Consistent Query Answers 81
5.1 Introduction 82
5.2 Preliminaries 86
5.3 A Framework for Divulging Inconsistent Databases 86
5.3.1 The Language **CQAFO** 87
5.3.2 Restrictions on Data Complexity 89
5.3.3 Strategies 91
5.4 How to Construct Good Strategies? 92
5.4.1 Post-Processing by Unions Only 92
5.4.2 Post-Processing by Unions and Quantification 95
5.5 Simplifying Strategies 100
5.5.1 Problem Statement and Motivation 100
5.5.2 Attacks from Atoms to Variables 104

5.5.3 A New Attack Notion 105
5.5.4 Testing containment 107
5.6 Conclusion 116

6 On the Syntax of Consistent First-Order Rewritings 119
6.1 Introduction 120
6.2 Notations and Terminology 121
6.3 Naive Algorithm 123
6.4 Reducing the Number of Quantifier Blocks 126
6.5 Reducing the Quantifier Rank 129
6.6 Conclusion 134

7 Tools 135
7.1 Canswer 135
7.1.1 Core 136
7.1.2 Attack Graphs 138
7.1.3 Rewrite 140
7.1.4 Under-Approximations 144
7.2 The Canswer Language 146
7.3 Top level interface 148
7.4 Cansweb 148
7.4.1 Cansweb Interface 148
7.4.2 Cansweb Examples 150
7.5 Conclusion 156

8 Conclusion 157

A Notations 161

Chapter 1

Introduction

Inconsistent, incomplete and uncertain data is widespread in the internet and social media era. This has given rise to a new paradigm for query answering, called *Consistent Query Answering* (CQA) [ABC99]. This paradigm starts with the notion of *repair*, which is a new consistent database that minimally differs from the original inconsistent database. In general, an inconsistent database can have many repairs. In this respect, database repairing is different from data cleaning which aims at computing a unique cleaned database.

In this work, we assume that the only constraints are primary keys, one per relation. A repair of an inconsistent database **db** is a maximal subset of **db** that satisfies all primary key constraints. Primary keys will be underlined; for example, the notation $\mathtt{R}(\underline{\mathtt{a}, \mathtt{b}}, \mathtt{c})$ for a database fact bears within it the information that the first two positions of `R` constitute the primary key. For example, the database of Figure 1.1 stores information about a company, in particular license plates of the cars being leased to employees, and the hierarchy between its collaborators. We assume that the values in the **Employee** and **Boss** columns uniquely identify persons. Every car is leased to exactly one employee, and each employee is under orders of exactly one person. However, distinct tuples may agree on the primary key **LicensePlate** (in relation `Cars`) or **Employee** (in relation `Hierarchy`), because there can be uncertainty about leasing and hierarchy. In the database of Figure 1.1, there is uncertainty about

Cars	<u>**LicensePlace**</u>	**Employee**
	1 − 204 − OEC	No
	1 − 204 − OEC	Ante
	1 − 527 − JKL	Xiou

Hierarchy	<u>**Employee**</u>	**Boss**
	No	Ante
	No	No
	Kemily	Ante

Figure 1.1: Example database with primary key violations.

the employee using the car with the plate 1 − 204 − OEC (it can be No or Ante) and about the manager of No (which is either Ante or No herself). Each relation can be repaired in two ways: delete either Cars(1 − 204 − OEC, No) or Cars(1 − 204 − OEC, Ante) from the Cars relation; and either Hierarchy(No, Ante) or Hierarchy(No, No) from the Hierarchy relation. This database thus has four repairs. A maximal set of tuples of the same relation that agree on their primary key will be called a *block*; in Figure 1.1, each relation is made of two blocks, which are separated by dashed lines.

When database repairing results in multiple repairs, CQA shifts from standard semantics to certainty semantics. Given a query, the *consistent answer* (also called *certain answer*) is defined as the intersection of the answers on all repairs. That is, for a query $\mathcal{Q}$ on an inconsistent database $\mathbf{db}$, CQA replaces the standard query answer $\mathcal{Q}(\mathbf{db})$ with the consistent answer, defined by the following intersection:

$$\bigcap\Big\{\mathcal{Q}(r) \mid r \text{ is a repair of } \mathbf{db}\Big\}. \tag{1.1}$$

Thus, the certainty semantics exclusively returns answers that hold true in every repair. Given a query $\mathcal{Q}$, we will denote by $\lfloor\mathcal{Q}\rfloor$ the query that maps a database to the consistent answer defined by (1.1). In this thesis, $\mathcal{Q}$ will always belong to some *syntactically* defined query class (e.g., conjunctive queries without self-joins). On the other hand, $\lfloor\mathcal{Q}\rfloor$ is a query in the sense that it is a mapping that associates to each database a set of answer tuples; this map-

ping is *semantically* defined by (1.1). The mapping $\lfloor \mathcal{Q} \rfloor$, however, may not be expressible in some common database query language.

A practical obstacle to CQA is that the shift to certainty semantics involves a significant increase in complexity. When we refer to complexity in this thesis, we mean data complexity, i.e., the complexity in terms of the size of the database (for a fixed query) [AHV95, p. 422]. It is known for long [Mar02] that there exist conjunctive queries $\mathcal{Q}$ that join two relations such that the data complexity of $\lfloor \mathcal{Q} \rfloor$ is already **coNP**-hard. If this happens, CQA is completely impracticable.

On the other hand, it is known that for a significant class of queries, the problem of computing consistent answers belongs to the complexity class **FO**, which refers to the descriptive complexity class that captures all queries expressible in relational calculus [Imm99]. If consistent answers can be computed in **FO**, they can be obtained by executing a single SQL query in an existing RDBMS, taking advantage of its query optimizer.

This thesis focuses on computing consistent answers to conjunctive queries without self-joins, i.e., without repeated relation names. In SQL, these are SELECT-FROM-WHERE queries where the SELECT-clause is a list of (table-qualified) attribute names and constants, the FROM-clause is a list of distinct relation names, and the WHERE-clause is a conjunction of equalities. We now outline the different chapters of this thesis.

Chapter 2 recalls important notions of database theory and gives a broad overlook of what parts of CQA have been addressed lately. In that chapter we do not restrict ourselves to primary keys only, which leads to different variants of database repairs and CQA.

Chapter 3 addresses the following problem: Given a query $\mathcal{Q}$, decide whether the mapping $\lfloor \mathcal{Q} \rfloor$ can be expressed by a formula in first-order logic, and construct such a formula if it exists. A first-order formula that expresses $\lfloor \mathcal{Q} \rfloor$ is commonly called a *consistent first-order rewriting for* $\mathcal{Q}$. In Chapter 3, this problem is studied under the restriction that $\mathcal{Q}$ is a self-join-free conjunctive query and the only constraints are primary keys. This chapter is based

on earlier works by Wijsen [Wij12] and Koutris and Wijsen [KW17]. The idea of consistent first-order rewriting is illustrated next. Suppose that we want to retrieve the employees managed by Ante (call this query $\mathcal{E}$). As the database is uncertain, we use the certainty semantics. That is, we want to retrieve the employees that are managed by Ante in every repair of the database; that is, only Kemily.

More formally, our query $\mathcal{E}$ can be expressed in relational calculus as

$$\mathcal{E} = \left\{ n \mid \texttt{Employee}(\underline{n}, \mathsf{Ante}) \right\}.$$

The consistent answers of this query can be expressed by the following relational calculus query, which is thus a consistent first-order rewriting for $\mathcal{E}$:

$$\lfloor \mathcal{E} \rfloor = \left\{ n \mid \begin{array}{l} \exists m (\texttt{Employee}(\underline{n}, m) \wedge \\ \quad \forall m (\texttt{Employee}(\underline{n}, m) \Rightarrow m = \mathsf{Ante})). \end{array} \right\}.$$

This query asks for employees under the management of Ante only. This relational calculus query can then easily be translated into SQL as shown in Listing 1.1. The method that computes consistent answers to a query $\mathcal{Q}$ by expressing $\lfloor \mathcal{Q} \rfloor$ in first-order logic is denoted by different names in the literature: first-order [query] rewriting, certain first-order [query] rewriting, consistent first-order [query] rewriting. Moreover, the word "first-order" is often omitted if it is clear from the context that the target language is first-order logic.

Chapters 4-7 form the core of this work, which aims at improving the method initially proposed by Wijsen [Wij10a]

Chapter 4 tries to take advantage of satisfied functional dependencies and of a particular form of join dependencies to extend the class of queries that are rewritable. For example, using our database, suppose we want to find out whether some employee uses a customized[1] license plate showing his/her superior's name. This query, denoted by $\mathcal{F}$, can be expressed in relational

[1] In some countries, one can buy customized license plates for an additional fee.

```sql
SELECT Employee
FROM Hierarchy b
WHERE NOT EXISTS (
  SELECT *
  FROM Hierarchy e
  WHERE b.Employee = e.Employee
    AND e.Boss <> 'Ante'
)
```

Listing 1.1: $\lfloor \mathcal{E} \rfloor$ in SQL.

calculus as:

$$\mathcal{F} = \Big\{ () \mid \exists n \, \exists p \, \Big(\mathtt{Cars}(\underline{p}, n) \land \mathtt{Hierarchy}(\underline{n}, p) \Big) \Big\} .$$

Queries like $\mathcal{F}$ which return a tuple of zero arity are called *Boolean*: the empty answer $\{\}$ is interpreted as **false**, and the nonempty answer $\{()\}$ as **true**. It is known [Wij10b] that $\lfloor \mathcal{F} \rfloor$ cannot be expressed using relational calculus. However, if our database is known to always satisfy the functional dependency $\mathtt{Cars} : \textbf{Employee} \to \textbf{LicensePlate}$ (that is, if no employee can have more than one license plate), then $\lfloor \mathcal{F} \rfloor$ can be expressed as the Boolean query[2]

$$\begin{aligned}
&\exists p(\\
&\quad \exists n(\mathtt{Cars}(\underline{p}, n) \land \\
&\quad \forall n(\mathtt{Cars}(\underline{p}, n) \Rightarrow \\
&\qquad \exists p'(\mathtt{Hierarchy}(\underline{n}, p') \land \\
&\qquad \forall p'(\mathtt{Hierarchy}(\underline{n}, p') \Rightarrow p = p'))))).
\end{aligned}$$

Chapter 5 deals with queries that are not rewritable. For such a query $\mathcal{Q}$, we know that there exists no relational calculus query that expresses $\lfloor \mathcal{Q} \rfloor$.

[2]Note that we follow the convention from [End72, p. 78] fixing operator precedence to (tightest to loosest) negation, quantification, conjunction, disjunction, implication, equivalence (all operators are right-associative).

What we can do is build a relational calculus query which computes a subset as large as possible (in a language **CQAFO** defined in that chapter) of the consistent answer of $\mathcal{Q}$. For example, consider again query $\mathcal{F}$ but suppose we have no information about any satisfied functional dependency. There are three obvious queries which have a non-empty consistent answer only if $\lfloor\mathcal{F}\rfloor$ returns true (intuitively, because they are more "restrictive" than $\mathcal{F}$):

- $\{p \mid \exists n \left(\texttt{Cars}(\underline{p}, n) \wedge \texttt{Hierarchy}(\underline{n}, p)\right)\}$;
- $\{n \mid \exists p \left(\texttt{Cars}(\underline{p}, n) \wedge \texttt{Hierarchy}(\underline{n}, p)\right)\}$; and
- $\{() \mid \exists n \left(\texttt{Cars}(\underline{n}, n) \wedge \texttt{Hierarchy}(\underline{n}, n)\right)\}$.

These three queries, unlike $\mathcal{F}$, do have a consistent first-order rewriting. Let φ be a first-order sentence stating that at least one of these three queries has a non-empty consistent answer. Obviously, if φ evaluates to true on some database, then so does $\lfloor\mathcal{F}\rfloor$ (but if φ evaluates to false, then the consistent answer to $\mathcal{F}$ can still be **true**). In this case, we will say that φ is a first-order under-approximation of $\lfloor\mathcal{F}\rfloor$.

Chapter 6 is concerned with queries which are rewritable, and focuses on simplifying the relational calculus query generated. In particular, we look for rewritings with low quantifier depth and with few quantifier blocks.

As an example, consider the query $\mathcal{G}$ asking whether a car is leased to employee Xiou and whether some employee is managed by himself:

$$\exists p \, \exists n \left(\texttt{Cars}(\underline{p}, \textsf{Xiou}) \wedge \texttt{Hierarchy}(\underline{n}, n)\right).$$

This query can be expressed using a relational calculus query. The algorithm proposed by Wijsen [Wij10a] would generate the following relational calculus

query:

$$
\begin{aligned}
&\exists p(\\
&\quad \exists n(\texttt{Cars}(\underline{p}, n) \wedge\\
&\quad \forall n(\texttt{Cars}(\underline{p}, n) \Rightarrow n = \mathsf{Xiou} \wedge\\
&\qquad \exists n(\\
&\qquad\quad \exists n'(\texttt{Hierarchy}(\underline{n}, n') \wedge\\
&\qquad\quad \forall n'(\texttt{Hierarchy}(\underline{n}, n') \Rightarrow n = n')))))).
\end{aligned}
$$

Our revised algorithm would rather produce the following query, which is the intersection of two simple queries. The advantage is clear when using modern computers able to execute several tasks at once on distinct processors, or with distributed databases.

$$
\begin{aligned}
&\exists p(\\
&\quad \exists n(\texttt{Cars}(\underline{p}, n)) \wedge\\
&\quad \forall n(\texttt{Cars}(\underline{p}, n) \Rightarrow n = \mathsf{Xiou})) \wedge\\
&\exists n(\\
&\quad \exists n'(\texttt{Hierarchy}(\underline{n}, n')) \wedge\\
&\quad \forall n'(\texttt{Hierarchy}(\underline{n}, n') \Rightarrow n = n'))
\end{aligned}
$$

Finally, Chapter 7 presents the tools that have been developed as part of this thesis, grouped under the Canswer project. These tools allow, among others, deciding whether queries (of some restricted query class) have a consistent first-order rewriting, and constructing such a rewriting if it exists.

The results of this thesis have already resulted in two journal publications and two international conference publications, as indicated in Table 1.1, which shows the correspondence between chapters and published articles. The chapters mentioned in the first column of Table 1.1 are to a large extent copies of the corresponding published articles in the second column. Since these articles are intended to be self-contained, the chapters derived from them contain some overlapping content. For reasons of readability, we have nevertheless uniformized the notation in the different chapters of this thesis, whereas the published articles exhibit differences in notation.

Chapter	**Publication**	**Venue**
Chapter 4	[GPW14]	Proceedings of the VLDB Endowment
Chapter 5	[GPW15]	Int. Conf. on Scalable Uncertainty Management (SUM 2015)
	[GPW17]	Int. Journal of Approximate Reasoning
Chapter 6	[DPW12]	Int. Conf. on Scalable Uncertainty Management (SUM 2012)

Table 1.1: Correspondence between chapters and published articles.

Chapter 2

Consistent Query Answering

The paradigm of Consistent Query Answering (CQA) relies on the notion of database repair. Different repair notions have been proposed in the literature. This chapter first introduces notions from database theory (schema, database, query, constraint), and then gives an overview of existing repair notions. Finally, the CQA paradigm is introduced and illustrated.

2.1 Preliminaries

2.1.1 Databases

We assume a denumerable set of relation names that is disjoint from a denumerable set of attributes. We assume a function **schema** that maps every relation name to a finite linearly ordered set of attributes. Let $\mathtt{R}$ be a relation name with $\mathbf{schema}(\mathtt{R}) = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$, where the attributes are listed in linear order. Then, the arity of $\mathtt{R}$ is said to be n. If $\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n$ are constants, then $\mathtt{R}(\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_n)$ is an $\mathtt{R}$-fact. If this $\mathtt{R}$-fact is denoted by F, then a_i is denoted by $F[i]$ or $F[\mathbf{A}_i]$.

We assume the existence of a countably infinite set **dom** of constants. A relational schema $\mathbf{R}$ is a finite set of relation names. Each relation name has a fixed arity $n \in \mathbb{N}$. A relational database **db** over $\mathbf{R}$ is a finite set of facts using only relation names in $\mathbf{R}$. If $\mathtt{R}$ is a relation name in the schema of **db**,

Employee

Emp	Salary	Birth	Hire
Ed	42000	1989	2007
Smith	24000	1989	2015

Team

Team	Emp	Leader?
accounting	Ed	Yes
accounting	Smith	No

Figure 2.1: A relational database composed of two relations `Employee` and `Team`.

then the set of all `R`-facts of **db** is denoted $\mathtt{R}^{\mathbf{db}}$ (or simply `R` if **db** is clear from the context) and called the relation `R` of **db**. It will always be clear from the context whether `R` refers to a relation name or a relation. The set of constants used in some database **db** is called its active domain, denoted **adom**(**db**).

Example 1. *Figure 2.1 shows the hypothetic database of some enterprise employing two people working in the enterprise's only team: accounting. This representation uses the named perspective [AHV95, p. 31]. Two relation names* `Employee` *and* `Team` *are used to model these facts. The arity of* `Employee` *is* 4 *while the arity of* `Team` *is* 3*. The cardinality of both relations (recall that relations are merely sets of facts) is* 2*.*

2.1.2 Queries

A query $\mathcal{Q}$ over some schema **S** is a total mapping from relational databases over **S** (seen as sets of facts over **dom**) to sets of tuples (over **dom**). We assume that the reader is familiar with Tuple Relational Calculus (TRC) [Mai83, Section 10.2] and Domain Relational Calculus (DRC) [Mai83, Section 10.5]. Both query languages are based on First-Order Logic (FOL) [End72] and have the same expressive power [Mai83, Sections 10.6, 10.7]. In this thesis, we denote by **FO** the complexity class of problems that take a database as input and that can be solved in first-order logic. **FO** is contained in the low circuit com-

plexity class $\mathbf{AC}^0$. We now fix notations and give examples for both DRC and TRC.

2.1.3 Domain Relational Calculus

We assume a set **var** of variables having no intersection with the set **dom** of constants. The set **sym** of symbols is $\mathbf{var} \cup \mathbf{dom}$. A DRC atom is either a Boolean value (**true** or **false**), an equality test between two symbols, or an atom which is a fact where in addition to constants, variables may be used. A DRC formula is a first-order formula over those atoms (conjunction $\wedge$, disjunction $\vee$, negation $\neg$, implication $\Rightarrow$, equivalence $\Leftrightarrow$, existential quantification $\exists$ and universal quantification $\forall$). A query is made of a query tuple and of a DRC formula. The query tuple is a tuple that can contain constants and that contains all (and only) the variables that have a free occurrence in the DRC formula. We require the queries to be domain independent [Top09]. That is, queries like $\{x \mid x = x\}$ are prohibited.

A *valuation* θ over a set X of variables is a mapping that associates a constant to each variable in X. Such a valuation over X is often understood to be the identity on all symbols not in X. Valuations extend to sequences and sets in the natural way: if $\vec{s} = (s_1, \ldots, s_n)$, then $\theta(\vec{s}) = (\theta(s_1), \ldots, \theta(s_n))$, and if S is a set, then $\theta(S) = \{\theta(s) \mid s \in S\}$. Given a DRC query $\mathcal{Q} = \{t \mid \phi\}$ and a database **db**, the answer $\mathcal{Q}(\mathbf{db})$ is the set of tuples $\vec{a}$, of the same arity as t, such that for some valuation θ over the free variables of ϕ, we have $\theta(t) = \vec{a}$ and $\theta(\phi)$ evaluates to true in **db**. Here, $\theta(\phi)$ is the closed formula obtained from ϕ by replacing each occurrence of each free variable x by $\theta(x)$. A fact DRC atom has the truth value **true** if and only if it occurs in **db**.

Example 2. *The query "Give all the employees' names" can be expressed by the following DRC query:*

$$\mathcal{Q} = \{n \mid \exists x\, \exists y\, \exists z\, (\texttt{Employee}(n, x, y, z))\}\,.$$

The query asking for the sum of the employees' salaries cannot be expressed as a first-order query, because we use first-order logic without arithmetic.

Given a set X of symbols, a sequence $\vec{s}$ of symbols, an atom F, a formula ϕ, or a query $\mathcal{Q}$, we write $\mathbf{vars}(\cdot)$ to denote the set of variables in these constructs: $\mathbf{vars}(X)$, $\mathbf{vars}(\vec{s})$, $\mathbf{vars}(F)$, $\mathbf{vars}(\phi)$, and $\mathbf{vars}(\mathcal{Q})$. Given a first-order logic formula ϕ, $\mathbf{free}(\phi)$ denotes the set of free variables of ϕ and $\phi_{x \to y}$ denotes the formula ϕ in which any free occurence of x has been replaced with y.

To alleviate the need of parentheses, we follow a usual convention fixing operator precedence to (tightest to loosest) negation, quantification, conjunction, disjunction, implication and equivalence. All operators are right-associative. For example, the formula $\forall x \forall y \neg(x = y) \vee (x = y)$ is equivalent to $(\forall x \forall y(\neg(x = y))) \vee (x = y)$.

An important class of queries is the class of conjunctive queries. In DRC this class can be characterized as the DRC queries that use only conjunction and existential quantifiers. It can be easily seen that such query can always be equivalently written in the following prenex normal form:

$$\{(s_1, s_2, \ldots, s_l) \mid \exists v_1 \, \exists v_2 \, \ldots \, \exists v_m \, (F_1 \wedge F_2 \wedge \cdots \wedge F_n)\}$$

where $F_1, F_2, \ldots, F_n$ are atoms (using only variables $s_1, s_2, \ldots, s_l, v_1, v_2, \ldots,$ v_m).

A conjunctive query $\mathcal{Q}$ is self-join-free if each relation name `R` is used at most once in $\mathcal{Q}$. We denote by **SJFCQ** the class of self-join-free conjunctive queries. A DRC query $\mathcal{Q}$ is Boolean if $\mathbf{free}(\mathcal{Q}) = \emptyset$.

2.1.4 Tuple Relational Calculus

We introduce a simple tuple relational calculus (TRC). This TRC is more restricted than TRCs found in the literature [Mai83], but suffices for our purposes. The main difference between DRC and TRC is that in DRC, variables range over atomic values, while in TRC, variables range over tuples. Variables in TRC play the same role as aliases in SQL.

For every relation name `R`, we assume denumerably many *tuple variables* of type `R`. No tuple variable can be of two distinct types. For every $n \in \{1, 2,$ $\ldots\}$, we assume a *free tuple* of arity n. Terms are defined as follows:

1. every constant is a term;

2. if r is a tuple variable of type $\mathtt{R}$ and $\mathbf{A} \in \mathbf{schema}(\mathtt{R})$, then $r[\mathbf{A}]$ is a term;

3. if f is a free tuple of arity n and $i \in 1, 2, \ldots, n$, then $f[i]$ is a term.

An atomic formula is an expression $t_1 = t_2$ where t_1 and t_2 are terms. Formulas are defined as follows:

1. every atomic formula is a formula;

2. if φ, φ_1, and φ_2 are formulas and r is a tuple variable of type $\mathtt{R}$, then $\exists r \in \mathtt{R}\,(\varphi)$, $\forall r \in \mathtt{R}\,(\varphi)$, $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are formulas.

We will use $f^{(n)}$ to denote a free tuple of arity n. A TRC query is an expression of the from

$$\left\{ f^{(n)} \mid \varphi \right\},$$

where φ is a formula in which no tuple variable has a free occurrence.

For example, "Give names and salaries of all leaders" can be stated in TRC as

$$\left\{ f^{(2)} \mid \exists t \in \mathtt{Team} \left(\exists n \in \mathtt{Employee} \left(\begin{array}{l} n[\mathbf{Emp}] = t[\mathbf{Emp}] \wedge \\ t[\mathbf{Leader?}] = \mathsf{Yes} \wedge \\ f[1] = n[\mathbf{Emp}] \wedge f[2] = n[\mathbf{Sal}] \end{array} \right) \right) \right\}$$

Every conjunctive query can be written as:

$$\left\{ f^{(n)} \mid \exists r_1 \in \mathtt{R_1}\,(\exists r_2 \in \mathtt{R_2}\,(\ldots \exists r_m \in \mathtt{R_m}\,(\varphi))) \right\},$$

where φ is a conjunction of atomic formulas.

We briefly discuss one important syntactic restriction of our TRC compared to the literature. In our TRC, each tuple variable is typed by a relation name and can only range over the tuples in that relation. This restriction will allow for an easier translation into SQL, in which an *alias* must also refer to a single relation. In the literature, one finds TRCs allowing queries like

$$\left\{ f^{(1)} \mid \exists t\,((\mathtt{R}(t) \vee \mathtt{S}(t)) \wedge t[1] = f[1]) \right\},$$

which associates t to both $\mathtt{R}$ and $\mathtt{S}$. In our TRC, this query can be expressed as:

$$\left\{f^{(1)} \mid \exists r \in \mathtt{R}\,(r[\mathbf{A}] = f[1]) \vee \exists s \in \mathtt{S}\,(s[\mathbf{B}] = f[1])\right\},$$

where it is assumed that $\mathbf{A}$ (resp. $\mathbf{B}$) is the first attribute of $\mathtt{R}$ (resp. $\mathtt{S}$) according to the linear order on attributes. Note incidentally that typed variables are not sufficient to guarantee *domain independence* [Top09], a property that should be possessed by all database queries. For example,

$$\left\{f^{(2)} \mid \exists r \in \mathtt{R}\,(r[\mathbf{A}] = f[1]))\right\}$$

is not domain independent, because $f[2]$ is not bound to a value. All TRC queries introduced later on in this thesis can be easily shown to be domain independent.

Given (linearly ordered) sets $X = \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ and $Y = \mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_n$ of attributes and two tuple variables s and t, the notation $s[X] = t[Y]$ stands for $s[\mathbf{X}_1] = t[\mathbf{Y}_1] \wedge s[\mathbf{X}_2] = t[\mathbf{Y}_2] \wedge \ldots \wedge s[\mathbf{X}_n] = t[\mathbf{Y}_n]$.

2.1.5 Constraints

It is nowadays considered fundamental that relational database management systems, in addition to effectively store and retrieve data, are able to ensure that some properties hold over the data. These properties are expressed under the form of Boolean queries. At any time, a database **db** under some constraint $\mathcal{C}$ has to satisfy $\mathcal{C}$. We introduce next some classes of constraints that play a central role in database systems.

2.1.6 Primary Keys

Every relation name $\mathtt{R}$ is associated with a unique *primary key*, which is a nonempty subset of **schema**($\mathtt{R}$). If the primary key of a relation name $\mathtt{R}$ is equal to **schema**($\mathtt{R}$), then $\mathtt{R}$ is called *full-key*.

An $\mathtt{R}$-relation r is said to satisfy its primary key if no two distinct tuples in r agree on all attributes of $\mathtt{R}$'s primary key. In other words, assuming that

$K = \mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_k$ is the key of R, then r satisfies its primary key if the following formula holds true:

$$\forall s \in R\, \forall t \in R\, (s[K] = t[K] \Rightarrow s = t)\,.$$

Henceforth, we will assume without loss of generality that in the linearly ordered set **schema**(R), the attributes of the primary key precede all other attributes. That is, if **schema**(R) is the linearly ordered set $\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$, then for all $1 \leq i \leq j \leq n$, if $\mathbf{A}_j$ belongs to R's primary key, then so does $\mathbf{A}_i$. It is also common practice to underline primary key attributes to distinguish them from other attributes.

In the unnamed perspective, every relation name is associated with a pair $[n, k]$ of positive integers where $n = |\mathbf{schema}(\mathtt{R})|$ and k is the cardinality of the primary key. The primary key of **schema**(R) then consists of the first k attributes in the linear order.

2.1.7 Functional dependencies

Functional dependencies (abbreviated **FD**) are a widespread form of constraints. Informally, in a relation R, a set of attributes B is functionally dependent on another set of attributes A if any pair of facts in R agreeing on the values of attributes in A also agree on the values of attributes in B.

A functional dependency is an expression of the form $R : A \rightarrow B$ where R is a relation name and $A, B \subseteq \mathbf{schema}(\mathtt{R})$. We say that a database **db** satisfies $\mathtt{R} : A \rightarrow B$, denoted $\mathbf{db} \models \mathtt{R} : A \rightarrow B$, if it satisfies the following Boolean query:

$$\forall s \in \mathtt{r}\, \forall t \in \mathtt{r}\, (s[A] = t[A] \Rightarrow s[B] = t[B])\,.$$

A functional dependency $\mathtt{R} : A \rightarrow B$ is called trivial if it is satisfied by every R-relation. It can be easily shown that $\mathtt{R} : A \rightarrow B$ is trivial if and only if $B \subseteq A$. A functional dependency $\mathtt{R} : A \rightarrow B$ is called a key dependency if $B = \mathbf{schema}(\mathtt{R})$. Obviously, if K is the primary key of R, then an R-

Models	**Id**	**MinimumAge**	**NumberOfParts**	**Motorized**
	5230	12	272	No
	1234	14	4242	No
	5243	14	2530	No
	2045	16	2025	Yes
	3752	16	1527	Yes

Figure 2.2: A relation `Models` storing data about construction game models.

relation satisfies its primary key if and only it satisfies the key dependency $\mathtt{R} : K \rightarrow \mathbf{schema}(\mathtt{R})$.

Example 3. *Consider the relation* `Models` *of Figure 2.2. The following functional dependencies are satisfied by the tuples of* `R`*:*

- $\mathtt{R} : \{\boldsymbol{Id}\} \rightarrow \{\boldsymbol{MinimumAge}, \boldsymbol{NumberOfParts}, \boldsymbol{Motorized}\}$*;*
- $\mathtt{R} : \{\boldsymbol{NumberOfParts}\} \rightarrow \{\boldsymbol{Id}\}$*;*
- $\mathtt{R} : \{\boldsymbol{MinimumAge}\} \rightarrow \{\boldsymbol{Motorized}\}$*.*

Note that the second functional dependency is satisfied by the relation of Figure 2.2, but is probably not a constraint that should be satisfied at all times, as two distinct construction game models could have the same number of parts.

The functional dependency $\mathtt{R} : \{\boldsymbol{Motorized}\} \rightarrow \{\boldsymbol{MimimumAge}\}$ *does not hold as the value* No *for attribute* ***Motorized*** *is associated to two distinct values for attribute* ***MinimumAge*** *(that are* 12 *and* 14*).*

Let Σ be a set of functional dependencies and let $A \rightarrow B$ be a functional dependency, all over (the attributes of) relation name `R`. We say that $A \rightarrow B$ is a logical consequence of Σ, denoted by $\Sigma \models A \rightarrow B$, if every `R`-relation that satisfies Σ also satisfies $A \rightarrow B$. It is well known that logical implication of functional dependencies is decidable in polynomial time.

Licensing	Id	License
	5230	LucasArts
	1234	LucasArts
	1234	NuclearBlast
	2045	Disney

Figure 2.3: A relation `Licensing` associating models from Figure 2.2 to license information.

2.1.8 Inclusion Dependencies

Inclusion dependencies express that attribute values found in the tuples of one relation, must re-occur in the tuples of another relation.

Formally, let $\mathtt{R}$ and $\mathtt{S}$ be two relation names taken from some schema $\mathbf{S}$ ($\mathtt{R}$ and $\mathtt{S}$ do not have to be distinct) Let $A = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$ be a subset of the attribute names of $\mathtt{R}$ and let $B = \{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n\}$ be a subset of the attribute names of $\mathtt{S}$ (note that $|A| = |B|$). Let **db** be a database over $\mathbf{S}$. Then **db** satisfies the inclusion dependency from $\mathtt{R}[A]$ to $\mathtt{S}[B]$, denoted $\mathtt{R}[A] \subset \mathtt{S}[B]$, if and only if **db** satisfies the following Boolean query:

$$\forall r \in \mathtt{R}\,(\exists s \in \mathtt{S}\,(s[B] = r[A]))\,.$$

Example 4. *The relations* `Models` *from Figure 2.2 and* `Licensing` *from Figure 2.3 satisfy the inclusion dependency* $\mathtt{Licensing}[\{\boldsymbol{Id}\}] \subset \mathtt{Models}[\{\boldsymbol{Id}\}]$.

On the other hand, $\mathtt{Models}[\{\boldsymbol{Id}\}] \subset \mathtt{Licensing}[\{\boldsymbol{Id}\}]$ *does not hold because model* 3752 *is not associated to any license.*

2.1.9 Join Dependencies

A join dependency involves n subsets of the attributes of some relation. It expresses that if n tuples over each of these subsets agree on their common attributes, then there must exist a tuple in the complete relation which agrees with each of the n tuples.

Preferences	Name	Likes	Dislikes
	Bart	SQL	NOSQL
	Bart	QBE	NOSQL
	Bart	SQL	nullvalues
	Bart	QBE	nullvalues
	Homer	post−its	Bart

Figure 2.4: A relation `Preferences` storing people's preferences and satisfying the join dependency `Preferences` : $\bowtie [\{\textbf{Name}, \textbf{Likes}\}, \{\textbf{Name}, \textbf{Dislikes}\}]$.

Let `R` be a relation name. Let $A_1, A_2, \ldots, A_n$ be subsets of (the attributes of) `R`. Let r be an `R`-relation. Then r satisfies the join dependency over $A_1, A_2, \ldots, A_n$, denoted `r` : $\bowtie [A_1, A_2, \ldots, A_n]$, if and only if r satisfies the following Boolean query:

$$
\begin{array}{l}
\forall r_1 \in \mathtt{R} \ldots \forall r_n \in \mathtt{R}(\\
\quad (\bigwedge_{1 \leq i < j \leq n} r_i[A_i \cap A_j] = r_j[A_i \cap A_j]) \Rightarrow \\
\quad \exists t \in \mathtt{R}\, (\bigwedge_{i=1}^{n} t[A_i] = r_i[A_i]) \\
)
\end{array}
$$

Example 5. *Consider the relation* `Preferences` *of Figure 2.4. This relation satisfies the join dependency* $\bowtie [\{\{\textbf{\textit{Name}}, \textbf{\textit{Likes}}\}, \{\textbf{\textit{Name}}, \textbf{\textit{Dislikes}}\}\}]$. *That is, if some tuple states that person p likes x, and another tuple states that p dislikes y, then there must be a tuple $\langle p, x, y \rangle$ in* `Preferences`*.*

A join dependency `R` : $\bowtie [A_1, A_2, \ldots, A_n]$ is called trivial if it is satisfied by every `R`-relation. It can be easily shown that `R` : $\bowtie [A_1, A_2, \ldots, A_n]$ is trivial if and only if for some $i \in 1, 2, \ldots, n$, A_i contains all attributes of `R`.

2.2 Data Cleaning

Given a schema $\mathbf{S}$, a set of constraints Σ over $\mathbf{S}$ and a database $\mathbf{db}$ over $\mathbf{S}$, $\mathbf{db}$ is said to satisfy Σ, denoted $\mathbf{db} \models \Sigma$, if for every constraint $\mathcal{C} \in \Sigma$, $\mathbf{db}$ satisfies

$\mathcal{C}$. A set of constraint Σ is satisfiable if there exists at least one database **db** such that **db** $\models \Sigma$.

A database may falsify one or more constraints it is supposed to satisfy. This situation can notably arise from the merging of several conflicting sources into one single relational databases, or from errors when data is copied. In such a situation, the usual solution is to ask an expert to change the data in the database so that it satisfies all the desired constraints. This way of dealing with inconsistent data is called data cleaning [RD00]. There are situations, however, where data cleaning is impractical, unaffordable or undesirable. For example, assume that a client database stores two different dates of birth for the same client. Data cleaning should restore the correct date of birth for that client. However, at data cleaning time, an expert may not be able to determine which date of birth is correct. Contacting the client may be unaffordable, and guessing the birth date may be undesirable.

2.3 Repairs

When it is not possible or desirable to clean the data, the database user has to work with inconsistent data. That is, a database for which constraints exist but are not satisfied. Such a database is called inconsistent. When dealing with such a database, we want queries to return only answers that hold true independently of the way the database is cleaned. This approach is called Consistent Query Answering (CQA) and depends on the constraints to be satisfied and on a given notion of repair.

Intuitively, a repair r of some inconsistent database **db** is a consistent database that minimally differs from **db**. The notion "minimally differs" can be defined in several ways, and this gives rise to several notions of repair. In the introduction of this thesis, we considered only primary key constraints and repairs that are maximal consistent subsets of the original database. We start by exploring this notion of repair in more details, then we introduce some other notions of repairs that have been addressed in the literature.

Votes

Question	Voter	Answer
Bestbeer	Francis	Cornet
Bestbeer	Clara	MaesRadler
Bestbeer	Chris	Cornet
Bestwine	Francis	Chablis
Bestwine	Chris	Chablis

Cred

CredibleVoter
Francis
Clara

Figure 2.5: A database storing votes and credible voters.

The repair semantics in the following sections are relative to a fixed set Σ of constraints. A database is consistent if it satisfies all constraints in Σ, and is inconsistent otherwise.

2.3.1 Inclusion-Based Repairs

Let **db** be a (possibly inconsistent) database. Under the inclusion semantics, a database $r' \subseteq \mathbf{db}$ is a repair for **db** if r' is consistent and there exists no consistent database r'' such that $r' \subsetneq r'' \subseteq \mathbf{db}$. Note that a consistent database has itself as unique repair.

Example 6. *Consider the inconsistent database from Figure 2.5. A fact* Votes(a, b, c) *means that voter* b *answered* c *to question* a*. Consider the following set of constraints:* $\Sigma = \{\texttt{Votes} : \{\textbf{\textit{Question}}\} \rightarrow \{\textbf{\textit{Answer}}\}\}$*. That is, a consistent database must associate at most one answer to each question.*

In each repair of this inconsistent database, a unique answer is associated to each of the two questions. As there are two possible answers for the Bestbeer *question and one possible answer for the* Bestwine *question, there are two repairs for this database. One of these repairs is shown in Figure 2.6.*

Votes	**Question**	**Voter**	**Answer**
	Bestbeer	Clara	MaesRadler
	Bestwine	Francis	Chablis
	Bestwine	Chris	Chablis

Cred	**CredibleVoter**
	Francis
	Clara

Figure 2.6: A repair under the inclusion semantics for the database of Figure 2.5 with $\Sigma = \{\texttt{Votes} : \{\textbf{Question}\} \rightarrow \{\textbf{Answer}\}\}$.

2.3.2 Symmetric Difference-Based Repairs

Given two arbitrary sets A and B, their symmetric difference, noted $A \Delta B$, is the set $(A \setminus B) \cup (B \setminus A)$. Given a (possibly inconsistent) database **db**, a database r is a repair for **db** under the symmetric difference semantics if r is consistent and there exists no consistent database r' such that $r' \Delta \mathbf{db} \subsetneq r \Delta \mathbf{db}$.

Example 7. *Consider again the database from Figure 2.5, but let* Σ *be the following set of constraints:* $\{\texttt{Votes}[\{\textbf{\textit{Voter}}\}] \subset \texttt{Cred}[\{\textbf{\textit{CredibleVoter}}\}]\}$.

Under the symmetric difference semantics, the database has two distinct repairs, which are obtained by either deleting all tuples about Chris *from the relation* `Votes`*, or by inserting* $\langle$Chris$\rangle$ *into the relation* `Cred`*.*

2.3.3 Cardinality-Based Repairs

Given a (possibly inconsistent) database **db**, a database r is a repair for **db** under the cardinality semantics if r is consistent and there exists no consistent database r' such that $|r' \Delta \mathbf{db}| < |r \Delta \mathbf{db}|$.

Example 8. *Consider again the database and constraints of Example 7. Under the cardinality semantics, there exists only one repair which is obtained by inserting the tuple* $\langle$Chris$\rangle$ *into the relation* `Cred`*.*

2.3.4 Tuple-Based or Value-Based

In our three repairs semantics above, databases are repaired by deleting entire tuples from the database or by inserting new tuples. This "tuple-based" style of repairing may not be desirable in all situations. An employee record, for example, may contain many correct data fields (first name, family name, salary, sex, address...) together with one incorrect field (e.g., year of birth 2090). In this situation, if we restore consistency by deleting the entire tuple, we also lose the correct data. In this situation, it seems more reasonable to restore consistency by means of updates, e.g., by changing the birth year 2090 into 1990 or Unknown. Such update-based (or value-based) repairing was introduced and studied in [Wij05].

As another example, consider again Example 7 about symmetric difference-based repairs. Under update-based repairing, a new repair pops up, in which each occurrence of Chris is replaced with Clara. In tuple-based repairing, this would not be a legal repair because it is "further away" from the original database than the repair obtained by deleting all tuples about Chris.

2.4 Consistent Query Answering

The paradigm of Consistent Query Answering (CQA) is defined relative to a fixed repair notion. Let $\mathbf{S}$ be a database schema and Σ be a set of constraints over $\mathbf{S}$. Let $\mathbf{db}$ be an inconsistent database over $\mathbf{S}$ with respect to Σ. Let $\mathcal{Q}$ be a query over $\mathbf{S}$. We are interested in finding the answers to $\mathcal{Q}$ that are true in every repair of $\mathbf{db}$. Such an answer is called consistent.

Example 9. *We work with the inclusion semantics for repairs. Consider again the database (which we call* $\mathbf{db}$*) from Figure 2.5 with* $\Sigma = \{\texttt{Votes} : \textbf{\textit{Question}} \rightarrow \textbf{\textit{Answer}}\}$*. From Example 6, we know that there exist two distinct repairs for* $\mathbf{db}$*. The first repair (which we call* r_1*) is given in Figure 2.6 and the second repair (*r_2*) is given in Figure 2.7. Let* $\mathcal{Q}$ *be the following query:*

$$\mathcal{Q} = \{(q, r) \mid \exists v\, (\texttt{Votes}(q, v, r) \wedge \texttt{Cred}(v))\},$$

Votes	Question	Voter	Answer
	Bestbeer	Francis	Cornet
	Bestbeer	Chris	Cornet
	Bestwine	Francis	Chablis
	Bestwine	Chris	Chablis

Cred	CredibleVoter
	Francis
	Clara

Figure 2.7: Another repair under the inclusion semantics for the database of Figure 2.5 with $\Sigma = \{\texttt{Votes} : \textbf{Question} \rightarrow \textbf{Answer}\}$.

expressing "What are the question-answer pairs associated to credible voters?"

Since $\mathcal{Q}(r_1) = \{(\mathsf{Bestbeer}, \mathsf{MaesRadler}), (\mathsf{Bestwine}, \mathsf{Chablis})\}$ *and* $\mathcal{Q}(r_2) = \{(\mathsf{Bestbeer}, \mathsf{Cornet}), (\mathsf{Bestwine}, \mathsf{Chablis})\}$*, the consistent answer to* $\mathcal{Q}$ *on* **db** *is the intersection* $\{(\mathsf{Bestwine}, \mathsf{Chablis})\}$*.*

In the remaining of this document, we restrict Σ to contain only primary keys, one per relation. It can be easily seen that under this restriction, different repair semantics (based on inclusion, symmetric difference, or cardinality) coincide.

We conclude this section with another example.

Example 10. *We consider the database* **db** *shown in Figure 2.8. A fact* $\texttt{Student}(\underline{n}, l)$ *means that student* n *works in lab* l*. A fact* $\texttt{Lab}(\underline{l}, r)$ *means that* r *is the rating of lab* l*. The primary keys of* `Student` *and* `Lab` *are* $\{\textbf{\textit{Name}}\}$ *and* $\{\textbf{\textit{Lab}}\}$ *respectively. The* `Student`*-relation has four repairs (because* Teemo *and* An *are associated to two labs), and the* `Lab`*-relation has two repairs. Consequently,* **db** *has eight repairs in total.*

Consider the query "Give the pairs of students and their associated rating," formally $\mathcal{Q} = \{(s, r) \mid \exists l(\texttt{Student}(\underline{s}, l) \wedge \texttt{Lab}(\underline{l}, r))\}$*. It can be easily verified that the consistent answer to* $\mathcal{Q}$ *on* **db** *is the set* $\{(\mathsf{Ed}, \mathsf{A}), (\mathsf{An}, \mathsf{A})\}$*.*

Student

Name	Lab
Ed	SSI
Frank	Robotics
Teemo	SSI
Teemo	Imagery
An	SSI
An	Hardware

Lab

Lab	Rating
SSI	A
Robotics	A
Robotics	B
Hardware	A

Figure 2.8: A database featuring students working in labs associated to ratings.

Chapter 3

Consistent Query Answering in First-Order Logic

This chapter first sets the context in which we will work, and then presents a method for solving the problem of consistent query answering under the inclusion semantics for repairs with respect to primary keys.

3.1 The problem CERTAINTY($\mathcal{Q}$)

As indicated in the end of the previous chapter, we fix the repair semantics to be the inclusion repair semantics. That is, given a database **db**, the set of repairs of **db**, denoted **repairs**(**db**), is the set of maximal consistent subsets of **db**. We will often use the term *uncertain database* at places where the term *database* would be sufficient and correct. In doing so, we want to emphasize that databases need not be consistent. In the database literature, the term uncertain/incomplete database often refers to some representation system that allows representing a set of possible [database] worlds [AHV95, Chapter 19]. In our setting, an uncertain database can also be seen as the representation of the set of its repairs.

We also restrict our study to primary key constraints, as stated in Section 2.1.6. Each relation name `R` is associated with a signature $[n, k]$ $(1 \leq$

$k \leq n$) where n is the arity of `R`, and k is the cardinality of `R`'s primary key. If F is an `R`-atom, then **keyvars**(F) denotes the set of variables occurring in $F[1], \ldots, F[k]$.

If φ is a closed first-order formula (which can express a database constraint or a Boolean query) and **db** is a database (which can be a repair), then we write $\mathbf{db} \models \varphi$ to denote that **db** satisfies φ according to standard first-order logic semantics. For a fixed first-order query $\mathcal{Q}(\vec{x})$ (over some fixed schema **S**), **CERTAINTY**$(\mathcal{Q}(\vec{x}))$ is the following problem:

Problem CERTAINTY$(\mathcal{Q}(\vec{x}))$

Input Uncertain database **db** over **S**

Output The consistent answer to $\mathcal{Q}(\vec{x})$, i.e., the set of tuples $\vec{a}$ such that $\forall r \in \mathbf{repairs}(\mathbf{db}), r \models \mathcal{Q}(\vec{a})$

Given a first-order query $\mathcal{Q}(\vec{x})$, we write $\lfloor \mathcal{Q}(\vec{x}) \rfloor$ for the mapping that takes as input an uncertain database and returns the consistent answer to $\mathcal{Q}(\vec{x})$. Note that $\lfloor \mathcal{Q}(\vec{x}) \rfloor$ is a query (in the sense of [Lib04, Definition 2.7]), even though it may not be expressible in first-order logic or some other common query language.

In this thesis, we will limit the input queries for **CERTAINTY**$(\mathcal{Q}(\vec{x}))$ to the class of self-join-free Boolean conjunctive queries. An important classification task is the following.

Problem Complexity classification task

Input Self-join-free Boolean conjunctive query $\mathcal{Q}$

Question Give lower and upper complexity bounds for the problem **CERTAINTY**$(\mathcal{Q})$

Notice that in the above complexity classification task, we assume that the input queries $\mathcal{Q}$ are Boolean, and that **CERTAINTY**($\mathcal{Q}$) asks whether $\mathcal{Q}$ is true in every repair. This allows us to restrict our attention to complexity classes for decision problems (in particular, **P** and **coNP**). However, most of our results extend to non-Boolean queries, as will become clear from Theorem 2.

Koutris and Wijsen [KW15, KW17] have solved this complexity classification task, as follows.

Theorem 1. *For every Boolean self-join-free conjunctive query* $\mathcal{Q}$*,*

- **CERTAINTY**($\mathcal{Q}$) *is either in* **P** *or* **coNP***-complete, and it can be decided in polynomial time (in the size of* $\mathcal{Q}$*) which of the two cases applies;*
- *it can be decided in polynomial time whether or not* **CERTAINTY**($\mathcal{Q}$) *is in* **FO***; and*
- *if* **CERTAINTY**($\mathcal{Q}$) *is in* **FO***, then a first-order query that solves* **CERTAINTY**($\mathcal{Q}$) *can be effectively constructed.*

If **CERTAINTY**($\mathcal{Q}(\vec{x})$) can be solved in first-order logic, then a first-order query that solves **CERTAINTY**($\mathcal{Q}(\vec{x})$) is called a consistent first-order rewriting for $\mathcal{Q}(\vec{x})$, as defined next.

Definition 1. *Let* $\mathcal{Q}(\vec{x})$ *be a first-order query. A first-order query* $\phi(\vec{x})$ *is said to be a* consistent first-order rewriting for $\mathcal{Q}(\vec{x})$ *if the following are equivalent for every uncertain database* **db** *and tuple* $\vec{a}$*:*

1. $\mathcal{Q}(\vec{a})$ *is true in every repair of* **db***; and*
2. $\phi(\vec{a})$ *is true in* **db***.*

The usage of the term "rewriting" is common in the literature on CQA, but can be considered a misnomer because in most cases $\mathcal{Q}$ and ϕ are not logically equivalent. As a matter of fact fact, in other applications (like query

rewriting under views or ontologies), "rewriting" commonly means "equivalent rewriting." Consistent first-order rewritings are of practical importance because, as illustrated next, they can be translated into SQL and executed on any SQL DBMS.

Example 11. *Consider again the database* **db** *from Figure 2.8 (where* ***Name*** *is the key of relation* Student *and* ***Lab*** *is the key of relation* Lab*) and the query* $\mathcal{Q}(s,r) = \exists l(\texttt{Student}(\underline{s}, l) \wedge \texttt{Lab}(\underline{l}, r))$*.*

It can be verified that the following query $\phi(s,r)$ *is a consistent first-order rewriting for* $\mathcal{Q}$*:*

$$\begin{aligned}\phi(s,r) = \ & \exists l(\texttt{Student}(\underline{s}, l) \wedge \\ & \quad \forall l(\texttt{Student}(\underline{s}, l) \Rightarrow (\\ & \quad\quad \texttt{Lab}(\underline{l}, r) \wedge \\ & \quad\quad\quad \forall r'(\texttt{Lab}(\underline{l}, r') \Rightarrow r' = r)))\end{aligned}$$

This chapter aims at giving a systematic way of constructing consistent first-order rewritings for self-join-free conjunctive queries whenever they exist. We will first argue why in the study of **CERTAINTY**($\mathcal{Q}$) for self-join-free conjunctive queries $\mathcal{Q}$, we can treat free variables as constants, and, as a consequence, restrict our attention to Boolean queries.

Definition 2. *Let* $\phi(x_1, \dots, x_n)$*, with* $n \geq 0$*, be a first-order formula with free variables* $x_1, \dots, x_n$*. Let* $\texttt{c}_1, \dots, \texttt{c}_n$ *be distinct constants. Then* $\phi_{x_1,\dots,x_n \mapsto \texttt{c}_1,\dots,\texttt{c}_n}$ *denotes the formula obtained from* ϕ *by replacing each free occurrence of* x_i *by* $\texttt{c}_i$ *(for each* $1 \leq i \leq n$*). Obviously, for every database* **db***,* $\mathbf{db} \models \phi_{x_1,\dots,x_n \mapsto \texttt{c}_1,\dots,\texttt{c}_n}$ *if and only if* $\mathbf{db} \models \phi(\texttt{c}_1, \dots, \texttt{c}_n)$*.*

The following definition, borrowed from [KW17], introduces typing in databases. Intuitively, in the technical treatment of self-join-free conjunctive queries $\mathcal{Q}$, one can assume that if two positions in $\mathcal{Q}$ contain distinct variables, then the corresponding "columns" in any database instance have no values in common. Furthermore, it can be assumed that these columns contain no constant that also occurs in $\mathcal{Q}$. For example, for the query $\mathcal{Q} = \{\texttt{R}(\underline{x}, y), \texttt{S}(\underline{y}),$

$\mathtt{T}(\underline{\mathtt{0}})\}$, where $\mathtt{0}$ is a constant, it can be assumed that whenever a database contains $\mathtt{R}(\underline{\mathtt{a}}, \mathtt{b})$ and $\mathtt{S}(\underline{\mathtt{c}})$, then it will be the case that $\mathtt{a} \neq \mathtt{b}$ and $\mathtt{a} \neq \mathtt{c}$ (but it may be that $\mathtt{b} = \mathtt{c}$). Furthermore, it can be assumed that $\mathtt{a} \neq \mathtt{0}$, $\mathtt{b} \neq \mathtt{0}$, and $\mathtt{c} \neq \mathtt{0}$. This simplifying assumption is no longer valid if self-joins are allowed (see Example 12 for an example), and is a major reason why most proofs of this thesis do not carry over to conjunctive queries with self-joins.

Definition 3 (Typed uncertain databases [KW17])**.** *For every variable x, we assume an infinite set of constants, denoted* **type**(x)*, such that $x \neq y$ implies* **type**$(x) \cap$ **type**$(y) = \emptyset$*. Let $\mathcal{Q}$ be a self-join-free conjunctive query and let* **db** *be an uncertain database. We say that* **db** *is* typed relative to $\mathcal{Q}$ *if for every atom $\mathtt{R}(x_1, \ldots, x_n)$ in $\mathcal{Q}$, for every $i \in \{1, \ldots, n\}$, if x_i is a variable, then for every fact $\mathtt{R}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$ in* **db***, $\mathtt{a}_i \in$* **type**(x_i) *and the constant $\mathtt{a}_i$ does not occur in $\mathcal{Q}$.*

An uncertain database **db** *can be trivially transformed into an uncertain database* **db**$'$ *that is typed relative to $\mathcal{Q}$ such that* **CERTAINTY**$(\mathcal{Q})$ *yields the same answer on problem instances* **db** *and* **db**$'$*. Indeed, we can take* **db**$'$ *to be the smallest database such that for every atom $\mathtt{R}(x_1, \ldots, x_n)$ in $\mathcal{Q}$, if* **db** *contains $\mathtt{R}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$, then* **db**$'$ *contains $\mathtt{R}(\mathtt{a}_1^{x_1}, \ldots, \mathtt{a}_n^{x_n})$. Here, each $\mathtt{c}^s$ denotes a constant such that 1. $\mathtt{c}_1^{s_1} = \mathtt{c}_2^{s_2}$ if and only if both $\mathtt{c}_1 = \mathtt{c}_2$ and $s_1 = s_2$, and 2. $\mathtt{c}^s = \mathtt{c}$ if and only if $\mathtt{c} = s$. Further, if x is a variable, then* **type**(x) *contains all (and only) constants of the form $\mathtt{c}^x$. Intuitively, occurrences of constants in* **db** *are "tagged" by the variable or constant that occurs at the same position in $\mathcal{Q}$, and these tags are unique because $\mathcal{Q}$ is self-join-free. Because of this transformation, the assumption that uncertain databases are typed can be made without loss of generality in the complexity classification task of* **CERTAINTY**$(\mathcal{Q})$ *for self-join-free conjunctive queries. This assumption is useful because it simplifies the technical treatment.*

In this thesis, most theoretical results on consistent first-order rewriting are stated for self-join-free conjunctive queries that are Boolean, i.e., that contain no free variables. We will now argue that this restriction is not a severe one, because free variables can be treated as constants. In the following

theorem, the notation x/a means "the variable symbol x interpreted as the constant symbol a." Before giving the theorem, we illustrate why the absence of self-joins is significant for the results in this thesis, and for Theorem 2 in particular.

Example 12. *Let $\mathcal{Q}(x) = \mathtt{R}(\underline{x}, \mathtt{0}) \wedge \mathtt{R}(\underline{\mathtt{c}}, \mathtt{1})$, where $\mathtt{c}$, $\mathtt{0}$, $\mathtt{1}$ are distinct constants. This query has a self-join. When we search for valuations θ of x that make $\mathcal{Q}$ true in every repair of some uncertain database, it is significant to distinguish between $\theta(x) = \mathtt{c}$ and $\theta(x) \neq \mathtt{c}$. Obviously, $\mathcal{Q}(\mathtt{c}) = \mathtt{R}(\underline{\mathtt{c}}, \mathtt{0}) \wedge \mathtt{R}(\underline{\mathtt{c}}, \mathtt{1})$ evaluates to false on every consistent database. It follows that $\mathcal{Q}(\mathtt{c})$ evaluates to false on every repair of any uncertain database, and thus $\mathtt{c}$ cannot be a consistent answer. On the other hand, for any constant $\mathtt{a}$ distinct from $\mathtt{c}$, $\mathcal{Q}(\mathtt{a}) = \mathtt{R}(\underline{\mathtt{a}}, \mathtt{0}) \wedge \mathtt{R}(\underline{\mathtt{c}}, \mathtt{1})$ can evaluate to true on a consistent database. It is correct to conclude that $\mathtt{c} \neq \mathtt{c}$ is a consistent first-order rewriting for $\mathcal{Q}(\mathtt{c})$, but $x \neq x$ is not a consistent first-order rewriting for $\mathcal{Q}(x)$.*

Theorem 2. *Let $\mathcal{Q}(x, \vec{y})$ be a self-join-free conjunctive query and let $\phi(x, \vec{y})$ be a first-order query such that no constant of $\mathbf{type}(x)$ occurs in $\mathcal{Q}$ or ϕ. Let a be a constant of $\mathbf{type}(x)$. Then, the following are equivalent:*

1. *$\phi(x, \vec{y})$ is a consistent first-order rewriting for $\mathcal{Q}(x, \vec{y})$; and*
2. *$\phi(x/\mathsf{a}, \vec{y})$ is a consistent first-order rewriting for $\mathcal{Q}(x/\mathsf{a}, \vec{y})$.*

Proof. It is obvious that $\boxed{1}$ implies $\boxed{2}$. For the opposite implication, assume that, when x is interpreted by the constant a, then $\phi(x, \vec{y})$ is a consistent first-order rewriting for $\mathcal{Q}(x, \vec{y})$. The desired result then holds by genericity of constants of $\mathbf{type}(x)$. In particular, if $\phi(x, \vec{y})$ contains an equality $x = \mathtt{c}$, then, since $\mathtt{c} \notin \mathbf{type}(x)$ by the hypothesis of the theorem, the equality $x = \mathtt{c}$ evaluates to false for *every* valuation of x by some constant in $\mathbf{type}(x)$. □

Theorem 2 suggests the following method to compute a consistent first-order rewriting ϕ for some non-Boolean self-join-free conjunctive query $\mathcal{Q}$:

- let $x_1, \ldots, x_n$ be the free variables of $\mathcal{Q}$;
- let $\mathcal{Q}'$ be $\mathcal{Q}(x_1/\mathsf{a_1}, \ldots, x_n/\mathsf{a_n})$, where each $\mathsf{a_i}$ is a fresh constant in $\mathbf{type}(x_i)$;
- let ϕ' be a consistent first-order rewriting for $\mathcal{Q}'$;
- let ϕ be ϕ' in which each occurrence of each constant $\mathsf{a_i}$ is replaced back to x_i.

Then ϕ is a consistent first-order rewriting for $\mathcal{Q}$.

This allows us to restrict our results to Boolean self-join-free conjunctive queries, without loss of generality. Note that ϕ does not depend on the constants $\mathsf{a}_1, \ldots, \mathsf{a}_n$. An example follows.

Example 13. *Consider again the query from Example 11:*

$$\mathcal{Q} = \left\{(s, r) \mid \exists l \left(\mathtt{Student}(\underline{s}, l) \land \mathtt{Lab}(\underline{l}, r)\right)\right\}.$$

The problem of finding a consistent first-order rewriting for $\mathcal{Q}$ now reduces to finding a consistent first-order rewriting for $\mathcal{Q}'$ where every free variable is replaced with a constant:

$$\mathcal{Q}' = \left\{() \mid \exists l \left(\mathtt{Student}(\underline{\mathsf{c}^s}, l) \land \mathtt{Lab}(\underline{l}, \mathsf{c}^r)\right)\right\}.$$

This is enough thanks to Theorem 2 which allows us to build a consistent first-order rewriting for $\mathcal{Q}$ given a consistent first-order rewriting for $\mathcal{Q}'$, which has the particularity of being Boolean.

Consider the following consistent first-order rewriting ϕ' for $\mathcal{Q}'$:

$$\begin{aligned} \phi'() = \quad & \exists l(\mathtt{Student}(\underline{\mathsf{c}^s}, l) \land \\ & \quad \forall l(\mathtt{Student}(\underline{\mathsf{c}^s}, l) \Rightarrow (\\ & \quad\quad \mathtt{Lab}(\underline{l}, \mathsf{c}^r) \land \\ & \quad\quad\quad \forall r'(\mathtt{Lab}(\underline{l}, r') \Rightarrow r' = \mathsf{c}^r)))). \end{aligned}$$

Then a consistent first-order rewriting for $\mathcal{Q}$ is

$$\begin{aligned} \phi'(s, r) = \quad & \exists l(\mathtt{Student}(\underline{s}, l) \land \\ & \quad \forall l(\mathtt{Student}(\underline{s}, l) \Rightarrow (\\ & \quad\quad \mathtt{Lab}(\underline{l}, r) \land \\ & \quad\quad\quad \forall r'(\mathtt{Lab}(\underline{l}, r') \Rightarrow r' = r)))). \end{aligned}$$

3.2 Attack Graphs

The construct of attack graph weas first introduced in [Wij10a] and is the main tool for establishing Theorem 1. Let $\mathcal{Q}$ be a self-join-free Boolean conjunctive query (denoted by its set of atoms). We define $\mathcal{K}(\mathcal{Q})$ as the following set of functional dependencies:

$$\mathcal{K}(\mathcal{Q}) := \{\mathbf{keyvars}(F) \rightarrow \mathbf{vars}(F) \mid F \in \mathcal{Q}\}.$$

For every atom $F \in \mathcal{Q}$, we define $F^{+,\mathcal{Q}}$ as the following set of variables:

$$F^{+,\mathcal{Q}} := \{x \in \mathbf{vars}(\mathcal{Q}) \mid \mathcal{K}(\mathcal{Q} \setminus \{F\}) \models \mathbf{keyvars}(F) \rightarrow x\}.$$

Here, the symbol $\models$ denotes standard logical entailment. The *attack graph of* $\mathcal{Q}$, denoted $\mathbf{attackgraph}(\mathcal{Q})$, is a directed graph whose vertices are the atoms of $\mathcal{Q}$. There is a directed edge from F to G ($F \neq G$) if there exists a sequence

$$F_0 \overset{z_1}{\frown} F_1 \overset{z_2}{\frown} F_2 \dots \overset{z_n}{\frown} F_n \tag{3.1}$$

where

- $F_0, \dots, F_n$ are atoms of $\mathcal{Q}$;
- $F_0 = F$ and $F_n = G$; and
- for all $i \in \{1, 2, \dots, n\}$, $z_i \in \mathbf{vars}(F_{i-1}) \cap \mathbf{vars}(F_i)$ and $z_i \notin F^{+,\mathcal{Q}}$.

A directed edge from F to G in the attack graph of $\mathcal{Q}$ is also called an *attack from F to G*, denoted by $F \overset{\mathcal{Q}}{\rightsquigarrow} G$. The sequence (3.1) is called a *witness* for the attack $F \overset{\mathcal{Q}}{\rightsquigarrow} G$. If $F \overset{\mathcal{Q}}{\rightsquigarrow} G$, then we also say that F *attacks* G (or that G is attacked by F).

Example 14. *Let* $\mathcal{Q} = \{\mathtt{R}(\underline{x}, y), \mathtt{S}(\underline{y}, z), \mathtt{T}(\underline{z}, x), \mathtt{U}(\underline{x}, u), \mathtt{V}(\underline{x, u}, v)\}$. *We have* $\mathtt{R}^{+,\mathcal{Q}} = \{x, u, v\}$. *A witness for* $\mathtt{R} \overset{\mathcal{Q}}{\rightsquigarrow} \mathtt{T}$ *is* $\mathtt{R} \overset{y}{\frown} \mathtt{S} \overset{z}{\frown} \mathtt{T}$. *Note that, by an abuse of notation, we write* R *to mean the* R*-atom of* $\mathcal{Q}$. *The complete attack graph is shown in Figure 3.1.*

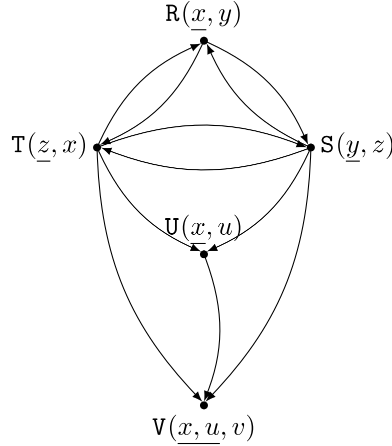

Figure 3.1: Attack graph of the query in Example 14.

The notion of attack graph is a syntactic notion. We now give some intuition behind the semantics of attacks, relative to some fixed self-join-free Boolean conjunctive query $\mathcal{Q}$. It is instructive to start with the meaning of *unattacked* atoms, i.e., atoms with zero indegree in the attack graph. One can show that if some atom F of $\mathcal{Q}$ is unattacked, then for every uncertain database **db**, the following property holds true: if $\mathcal{Q}$ is true in every repair of **db**, then there exists a valuation θ over **keyvars**(F) (where θ depends on **db**) such that $\theta(\mathcal{Q})$ is true in every repair of **db**. In other words, if F is unattacked and $\mathcal{Q}$ is true in every repair of **db**, then $\mathcal{Q}$ is true in every repair of **db** *for some fixed value of F's primary key.* Furthermore, one can show that this property precisely characterizes unattacked atoms, i.e., the property never holds true for attacked atoms. [1] Take, for example, the query $\mathcal{Q}_0 = \{\mathtt{R}(\underline{x}, y),\ \mathtt{S}(\underline{y})\}$. Let $\mathbf{db} = \{\mathtt{R}(\underline{\mathtt{a}}, 1),\ \mathtt{R}(\underline{\mathtt{a}}, 2),\ \mathtt{S}(\underline{1}),\ \mathtt{S}(\underline{2})\}$, which has two repairs: $r_1 = \{\mathtt{R}(\underline{\mathtt{a}}, 1),\ \mathtt{S}(\underline{1}),$ $\mathtt{S}(\underline{2})\}$ and $r_2 = \{\mathtt{R}(\underline{\mathtt{a}}, 2),\ \mathtt{S}(\underline{1}),\ \mathtt{S}(\underline{2})\}$. Both repairs satisfy $\mathcal{Q}_0$, but this is no longer true if we would fix the primary key of S: the first repair falsifies $\mathcal{Q}_{0_{y\mapsto 2}}$

[1] This observation led to the term "attack" [Wij]: atoms undergoing no attack can fix a primary key value independent of other atoms; attacked atoms, on the other hand, depend on the values chosen by their attackers.

and the the second repair falsifies $\mathcal{Q}_{0y\mapsto 1}$. It is then correct to conclude that the $\mathtt{S}$-atom is attacked. On the other hand, the primary key of $\mathtt{R}$ can be fixed in this example: both repairs satisfy $\mathcal{Q}_{0x\mapsto \mathsf{a}}$. Moreover, it is not hard to see that if all repairs of some uncertain database $\mathbf{db}'$ satisfy $\mathcal{Q}_0$, then there exists some value a' (which depends on $\mathbf{db}'$) such that all repairs satisfy $\mathcal{Q}_{0x\mapsto \mathsf{a}'}$. In view of this, it is correct to conclude that the $\mathtt{R}$-atom is unattacked. A consistent first-order rewriting for $\mathcal{Q}_0$ is

$$\exists x \left(\exists y \mathtt{R}(\underline{x}, y) \wedge \forall y \left(\mathtt{R}(\underline{x}, y) \implies \mathtt{S}(\underline{y})\right)\right).$$

The leading existential quantification ($\exists x$ in this example) captures that if all repairs of some uncertain database $\mathbf{db}$ satisfy $\mathcal{Q}_0$, then there exists a value a such that all repairs of $\mathbf{db}$ satisfy $\mathcal{Q}_{0x\mapsto \mathsf{a}}$.

Attack graphs allow us to determine whether a self-join-free Boolean conjunctive query has a consistent first-order rewriting and to compute such a rewriting if it exists.

Theorem 3 ([KW17]). *For every Boolean self-join-free conjunctive query $\mathcal{Q}$, the problem* **CERTAINTY**($\mathcal{Q}$) *is in* **FO** *if and only if $\mathcal{Q}$'s attack graph is acyclic. Moreover, if* **CERTAINTY**($\mathcal{Q}$) *is in* **FO**, *then a consistent first-order rewriting for $\mathcal{Q}$ can be effectively constructed.*

The if-part of the proof of Theorem 3, which can be found in [KW17], is constructive: if $\mathcal{Q}$'s attack graph is acyclic, then we can effectively construct a consistent first-order rewriting for $\mathcal{Q}$. In what follows, we explain the main ideas underlying this construction. To this extent, let $\mathcal{Q}$ be a Boolean self-join-free conjunctive query with an acyclic attack graph. We show, by induction on the number $|\mathcal{Q}|$ of atoms in $\mathcal{Q}$, the existence and the construction of a consistent first-order rewriting for $\mathcal{Q}$.

The proof of the if-part of Theorem 3 is obvious if $|\mathcal{Q}| = 0$; this is the base case of the induction. For the induction step, assume that $\mathcal{Q}$ with $|\mathcal{Q}| \geq 1$ has an acyclic attack graph. Then $\mathcal{Q}$'s attack graph must contain an atom with zero indegree. Let id denote the identity substitution. We distinguish two cases.

Case RewEmptyKey. The following is obvious. If $\mathcal{Q}$ contains $\mathtt{R}(\underline{\vec{a}}, \vec{y})$ with $\mathbf{vars}(\vec{a}) = \emptyset$, then for every uncertain database **db**, the following are equivalent:

1. $\mathcal{Q}$ is true in every repair of **db**; and
2. **db** contains an R-fact $\mathtt{R}(\underline{\vec{a}}, \vec{b})$ such that FOR EVERY atom $\mathtt{R}(\underline{\vec{a}}, \vec{c})$ in **db**, $\mathsf{id}_{\vec{y} \mapsto \vec{c}}$ is well-defined and $\mathsf{id}_{\vec{y} \mapsto \vec{c}}(\mathcal{Q}')$ is true in every repair of **db**, where $\mathcal{Q}' = \mathcal{Q} \setminus \{\mathtt{R}(\underline{\vec{a}}, \vec{y})\}$.

Note that the atom $\mathtt{R}(\underline{\vec{a}}, \vec{y})$ with $\mathbf{vars}(\vec{a}) = \emptyset$ will always have zero indegree in $\mathcal{Q}$'s attack graph.

Case RewAtom. The following result has been proven in [KW17]. If $\mathcal{Q}$'s attack graph contains an unattacked atom $\mathtt{R}(\underline{\vec{x}}, \vec{y})$ with $\mathbf{vars}(\vec{x}) \neq \emptyset$, then for every uncertain database **db**, the following are equivalent:

1. $\mathcal{Q}$ is true in every repair of **db**; and
2. THERE EXISTS $\vec{a}$ such that $\mathsf{id}_{\vec{x} \mapsto \vec{a}}$ is well-defined and $\mathsf{id}_{\vec{x} \mapsto \vec{a}}(\mathcal{Q})$ is true in every repair of **db**. Note that the R-atom in $\mathsf{id}_{\vec{x} \mapsto \vec{a}}(\mathcal{Q})$ contains no variable at a primary-key position, and hence Case RewEmptyKey applies.

Assume that the attack graph of $\mathcal{Q}$ contains an unattacked atom $\mathtt{R}(\underline{\vec{a}}, \vec{y})$ or $\mathtt{R}(\underline{\vec{x}}, \vec{y})$. Let $\vec{z}$ be a sequence of variables that contains exactly once each variable of this atom, and let $\vec{\mathsf{c}}$ be a sequence of distinct constants not occurring in $\mathcal{Q}$. It has been shown in [KW17] that $\mathcal{Q}'_{\vec{z} \mapsto \vec{\mathsf{c}}}$ has an acyclic attack graph, hence by Theorem 2 and the induction hypothesis, $\mathcal{Q}'(\vec{z})$ has a consistent first-order rewriting (call it $\phi'(\vec{z})$).

From what precedes, it is correct to conclude that if the attack graph of $\mathcal{Q}$ is acyclic, then we can construct a consistent first order-rewriting for $\mathcal{Q}$. Roughly, Case RewAtom results in an existential quantification expressing the existence of $\vec{a}$, and Case RewEmptyKey then results in a universal quantification over all the R-facts of the form $\mathtt{R}(\underline{\vec{a}}, \vec{c})$.

For example, if the unattacked atom is $\mathtt{R}(\underline{\mathsf{a}, x, x}, x, y, y, \mathsf{b})$, then the following is a consistent first-order rewriting:

$$\begin{aligned} &\exists x\,\exists y(\mathtt{R}(\underline{\mathsf{a}, x, x}, x, y, y, \mathsf{b})\wedge \\ &\quad \forall u\,\forall v\,\forall w(\mathtt{R}(\underline{\mathsf{a}, x, x}, u, y, v, w) \Rightarrow u = x \wedge v = y \wedge w = \mathsf{b} \wedge \phi'(x, y))). \end{aligned}$$

The conjunction $u = x \wedge v = y \wedge w = \mathsf{b}$ captures that $\mathsf{id}_{\vec{y}\mapsto\vec{c}}$ in Case RewEmptyKey must be well-defined. The following section studies in more depth the construction of consistent first-order rewritings.

3.3 Rewriting Function

In this section, we describe a function that takes as input a self-join-free conjunctive query $\mathcal{Q}$ with an acyclic attack graph and outputs a consistent first-order rewriting for $\mathcal{Q}$. The function is split up in several subfunctions constructed according to the proof of Theorem 3. As the functions recursively call themselves, we first give the expected input and outputs of the functions.

3.3.1 Expected input/outputs

Function Rewrite

Input $\mathcal{Q}$ is a query in **SJFCQ** that has a consistent first-order rewriting.

Output A consistent first-order rewriting for $\mathcal{Q}$.

Function RewAtom

Input $\mathcal{Q}$ is a Boolean self-join-free conjunctive query with an acyclic attack graph.

Input $\mathtt{R}$ is a relation name such that there exists an $\mathtt{R}$-atom in $\mathcal{Q}$ which is not attacked in the attack graph of $\mathcal{Q}$.

Output A consistent first-order rewriting for $\mathcal{Q}$.

Function RewEmptyKey

Input $\mathcal{Q}$ is a Boolean self-join-free conjunctive query with an acyclic attack graph.

Input R is a relation name such that there exists an R-atom in $\mathcal{Q}$ whose primary key contains no variables.

Output A consistent first-order rewriting for $\mathcal{Q}$.

We now introduce these three functions and argue that they are correct.

3.3.2 Function Rewrite

This function is the entry point of our rewriting process. Its role is just to find an atom that is not attacked and to call *Function RewAtom*. It also handles the case in which $\mathcal{Q}$ is empty.

Input: $\mathcal{Q}$ is a n-ary query in **SJFCQ** that has a consistent first-order rewriting.
Result: A consistent first-order rewriting ϕ for $\mathcal{Q}$.
if $\mathcal{Q}$ *is empty* **then**
  return true;
else
  Let $x_1, \ldots, x_n$ be the free variables of $\mathcal{Q}$;
  Let $\mathcal{Q}'$ be $\mathcal{Q}_{x_1,\ldots,x_n \to \mathsf{a}^{x_1},\ldots,\mathsf{a}^{x_n}}$;
  Let F be an unattacked R-atom of $\mathcal{Q}'$;
  Let ϕ be $RewAtom(\mathcal{Q}', \mathtt{R})$;
  return $\phi_{\mathsf{a}^{x_1},\ldots,\mathsf{a}^{x_n} \to x_1,\ldots,x_n}$;

Function Rewrite($\mathcal{Q}$)

It is clear that **true** is a consistent rewriting for the empty query. When a non-empty query $\mathcal{Q}$ is given as parameter to *Function Rewrite*, a non-attacked atom is randomly picked from $\mathcal{Q}$—note that the existence of such an atom is ensured by the acyclicity condition on the attack graph—and passed to the *Function RewAtom*.

The valuations $\mathsf{id}_{x_i \mapsto \mathsf{a}^{x_i}}$ and their inverses $\mathsf{id}_{\mathsf{a}^{x_i} \mapsto x_i}$ implement the rewriting strategy for non-Boolean queries outlined in the paragraph after the proof of Theorem 2. The constants a^{x_i} are as described in Definition 3.

3.3.3 Function RewAtom

This function removes the variables in the primary key of the selected unattacked atom, producing a query meeting the preconditions of Function RewEmptyKey, which requires that the primary key of the selected unattacked atom contains no variables.

Input: $\mathcal{Q}$ is a Boolean self-join-free conjunctive query with an acyclic attack graph.
Input: `R` is a relation name such that $\mathcal{Q}$ contains an `R`-atom that is unattacked in the attack graph of $\mathcal{Q}$.
Result: A consistent first-order rewriting ϕ for $\mathcal{Q}$.
Let F be the `R`-atom of $\mathcal{Q}$;
if keyvars(F) *is empty* **then**
  return $RewEmptyKey(\mathcal{Q}, \mathtt{R})$;
else
  Let x be a variable from **keyvars**(F);
  Let $\mathcal{Q}'$ be $\mathcal{Q}_{x \to \mathsf{a}^x}$;
  Let ϕ be $RewAtom(\mathcal{Q}', \mathtt{R})$;
  return $\exists x\, (\phi_{\mathsf{a}^x \to x})$;

Function RewAtom($\mathcal{Q}$, `R`)

It is obvious that the function terminates because the number of variables in the primary key of the `R`-atom decreases at each recursive call (until it reaches zero, at which point Function RewEmptyKey is called, whose execution will remove the `R`-atom).

The correctness of Function RewAtom is supported by **Case RewAtom** of Theorem 3.

3.3.4 Function RewEmptyKey

This last function is to be applied on an R-atom whose primary key is a sequence $\vec{a}$ without variables. It constructs a first-order formula in which a universal quantification ranges over all R-facts with primary key value $\vec{a}$.

Input: $\mathcal{Q}$ is a Boolean self-join-free conjunctive query with an acyclic attack graph.
Input: R such that $\mathcal{Q}$ contains an R-atom $F = \mathtt{R}(\underline{x_1, x_2, \ldots, x_k}, x_{k+1}, x_{k+2}, \ldots, x_n)$ satisfying **keyvars**$(F) = \emptyset$.
Result: A consistent first-order rewriting for $\mathcal{Q}$.

$C \leftarrow$ **true**;
$V \leftarrow \emptyset$;
$\mathcal{Q}' \leftarrow \mathcal{Q} \setminus \{F\}$;
for $i = k+1$ **to** n **do**
  $\mathcal{Q}' \leftarrow \mathcal{Q}'_{x_i \rightarrow \mathtt{a}^{x_i}}$;
  if x_i *is a variable that does not occur in* $\{x_{k+1}, x_{k+2}, \ldots, x_{i-1}\}$ **then**
    y_i is x_i;
    $V \leftarrow V \cup \{x_i\}$;
  else
    y_i is the fresh variable $v_i^{\mathtt{R}}$;
    $C \leftarrow C \land v_i^{\mathtt{R}} = x_i$;
Let ϕ' be *Rewrite*$(\mathcal{Q}')$;
foreach $x_i \in V$ **do**
  $\phi' \leftarrow \phi'_{\mathtt{a}^{x_i} \rightarrow x_i}$;
Let ϕ be $\forall y_{k+1} \ldots \forall y_n (\mathtt{R}(\underline{x_1, \ldots, x_k}, y_{k+1}, \ldots, y_n) \Rightarrow C \land \phi')$;
return $\exists V(F) \land \phi$;

Function RewEmptyKey($\mathcal{Q}$, R)

The correctness of this function is supported by **Case RewEmptyKey** of Theorem 3. The main idea can be illustrated by taking $\mathtt{R}(\underline{\vec{a}}, x, x, \mathtt{c})$ as an example. In this case, the rewriting must verify that whenever an uncertain

database contains $\mathtt{R}(\underline{\vec{a}}, x, y_2, y_3)$, then it must be the case that $y_2 = x$ and $y_3 = \mathsf{c}$:

$$\forall x \forall y_2 \forall y_3 \left(\mathtt{R}(\underline{\vec{a}}, x, y_2, y_3) \Rightarrow (y_2 = x \wedge y_3 = \mathsf{c}) \right)$$

Note that we introduce two fresh variables y_2 and y_3, but we must keep the variable x (i.e., there is no fresh variable y_1), because x can re-occur in the atoms that have not yet been rewritten. Thus, Function RewEmptyKey traverses the non-primary-key positions of F from left to right, and whenever it finds a variable that has been encountered before or a constant, it adds an appropriate equality to C.

3.3.5 Examples

We now illustrate the application of Function Rewrite by means of an example.

Let $\mathcal{Q} = \{\mathtt{R}(\underline{x}, x, y, y, z, v, \mathsf{a}), \mathtt{S}(\underline{v}, w)\}$. We name the R-atom F and the S-atom G. Note that we can give $\mathcal{Q}$ as input to Function Rewrite as its attack graph is acyclic (it is contains a single attack from F to G).

The initial call to *Function Rewrite*$(\mathcal{Q})$ recognizes that $\mathcal{Q}$ is not empty, finds out that F is not attacked, and returns the result of $RewAtom(\mathcal{Q}, \mathtt{R})$. At this point Function RewAtom will call itself recursively as the key of F contains the variable x. The query argument will be $\{\mathtt{R}(\underline{\mathsf{a}^x}, \mathsf{a}^x, y, y, z, v, \mathsf{a}), \mathtt{S}(\underline{v}, w)\}$.

In turn, this will trigger the execution of *Function RewEmptyKey*, which will finish the treatment of the R-atom.

$$\begin{aligned}
\lfloor \mathcal{Q} \rfloor &= \mathit{Rewrite}(\mathcal{Q}) \\
&= \mathit{RewAtom}(\mathcal{Q}, \mathtt{R}) \\
&= \exists x(\phi_{\mathsf{a}^x \to x}), \mathit{where} \\
\phi &= \mathit{RewAtom}(\mathtt{R}(\underline{\mathsf{a}^x}, \mathsf{a}^x, y, y, z, v, \mathsf{a}) \wedge \mathtt{S}(\underline{v}, w), \mathtt{R}) \\
&= \mathit{RewEmptyKey}(\mathtt{R}(\underline{\mathsf{a}^x}, \mathsf{a}^x, y, y, z, v, \mathsf{a}) \wedge \mathtt{S}(\underline{v}, w), \mathtt{R}) \\
&= \exists y \, \exists z \, \exists v(\mathtt{R}(\underline{\mathsf{a}^x}, \mathsf{a}^x, y, y, z, v, \mathsf{a}) \wedge \\
&\qquad \forall v_1^{\mathtt{R}} \, \forall y \, \forall v_3^{\mathtt{R}} \, \forall z \, \forall v \, \forall v_6^{\mathtt{R}} (\mathtt{R}(\underline{\mathsf{a}^x}, v_1^{\mathtt{R}}, y, v_3^{\mathtt{R}}, z, v, v_6^{\mathtt{R}}) \Rightarrow (\\
&\qquad\quad v_1^{\mathtt{R}} = \mathsf{a}^x \wedge v_3^{\mathtt{R}} = y \wedge v_6^{\mathtt{R}} = \mathsf{a} \wedge \psi_{\mathsf{a}^y, \mathsf{a}^z, \mathsf{a}^v \to y, z, v}))), \mathit{where} \\
\psi &= \mathit{Rewrite}(\mathtt{S}(\underline{\mathsf{a}^v}, w))
\end{aligned}$$

The expansion of the subquery goes as follows:

$$\begin{aligned}
\psi &= \mathit{Rewrite}(\mathtt{S}(\underline{\mathsf{a}^v}, w)) \\
&= \mathit{RewAtom}(\mathtt{S}(\underline{\mathsf{a}^v}, w), \mathtt{S}) \\
&= \mathit{RewEmptyKey}(\mathtt{S}(\underline{\mathsf{a}^v}, w), \mathtt{S}) \\
&= \exists w(\mathtt{S}(\underline{\mathsf{a}^v}, w) \wedge \\
&\qquad \forall w(\mathtt{S}(\underline{\mathsf{a}^v}, w) \Rightarrow (\\
&\qquad\quad \mathit{Rewrite}(\emptyset)))).
\end{aligned}$$

Summing everything up and applying remaining substitutions, we obtain

$$\begin{aligned}
\psi =& \ \exists w(\mathtt{S}(\underline{\mathsf{a}^v}, w)) \wedge \forall w(\mathtt{S}(\underline{\mathsf{a}^v}, w) \Rightarrow \mathbf{true}) \\
\phi =& \ \exists y \, \exists z \, \exists v(\mathtt{R}(\underline{\mathsf{a}^x}, \mathsf{a}^x, y, y, z, v, \mathsf{a})) \wedge \forall v_1^{\mathtt{R}} \, \forall y \, \forall v_3^{\mathtt{R}} \, \forall z \, \forall v \, \forall v_6^{\mathtt{R}} (\\
&\quad \mathtt{R}(\underline{\mathsf{a}^x}, v_1^{\mathtt{R}}, y, v_3^{\mathtt{R}}, z, v, v_6^{\mathtt{R}}) \Rightarrow v_1^{\mathtt{R}} = \mathsf{a}^x \wedge v_3^{\mathtt{R}} = y \wedge v_6^{\mathtt{R}} = \mathsf{a} \wedge \\
&\qquad \exists w(\mathtt{S}(\underline{v}, w)) \wedge \forall w(\mathtt{S}(\underline{v}, w) \Rightarrow \mathbf{true})) \\
\lfloor \mathcal{Q} \rfloor =& \ \exists x(\\
&\quad \exists y \, \exists z \, \exists v(\mathtt{R}(\underline{x}, x, y, y, z, v, \mathsf{a})) \wedge \forall v_1^{\mathtt{R}} \, \forall y \, \forall v_3^{\mathtt{R}} \, \forall z \, \forall v \, \forall v_6^{\mathtt{R}} (\\
&\qquad \mathtt{R}(\underline{x}, v_1^{\mathtt{R}}, y, v_3^{\mathtt{R}}, z, v, v_6^{\mathtt{R}}) \Rightarrow v_1^{\mathtt{R}} = x \wedge v_3^{\mathtt{R}} = y \wedge v_6^{\mathtt{R}} = \mathsf{a} \wedge \\
&\qquad\quad \exists w(\mathtt{S}(\underline{v}, w)) \wedge \forall w(\mathtt{S}(\underline{v}, w) \Rightarrow \mathbf{true}))).
\end{aligned}$$

Obviously, further syntactic simplifications can be applied. For example, $\forall w(\mathtt{S}(\underline{v}, w) \Rightarrow \mathbf{true})$ can be replaced by **true**. Such syntactic simplifications will be studied in Chapter 6.

Non-Boolean queries can be treated as explained in the paragraph following the proof of Theorem 2. Take, for example, $\mathcal{Q}(x) = \mathtt{R}(\underline{x}, x)$. Only Function Rewrite is concerned with free variables. The function call $\mathit{Rewrite}(\mathtt{R}(\underline{x}, x))$ will expand to $\mathit{RewAtom}(\mathtt{R}(\underline{\mathtt{a}^x}, \mathtt{a}^x), \mathtt{R})_{\mathtt{a}^x \to x}$. The final output will then be $\mathtt{R}(\underline{x}, x) \wedge \forall v_1^{\mathtt{R}}(\mathtt{R}(\underline{x}, v_1^{\mathtt{R}}) \Rightarrow v_1^{\mathtt{R}} = x)$, which is the expected output.

3.4 Related Work

The investigation of the problem **CERTAINTY**($\mathcal{Q}$) was pioneered by Fuxman and Miller [FM05, FM07], who defined a large subclass of **SJFCQ** such that every query $\mathcal{Q}$ in the subclass possesses a consistent first-order rewriting. This result has later on been improved by Wijsen [Wij10a, Wij12], who developed an effective method to decide, given an acyclic query $\mathcal{Q}$ in **SJFCQ**, whether $\mathcal{Q}$ has a consistent first-order rewriting. Finally, decidability of the existence of consistent first-order rewritings for the entire class **SJFCQ** was settled by Koutris and Wijsen [KW17].

In their conclusion of [FM07], Fuxman and Miller [FM05, FM07] raised the question whether **SJFCQ** contains queries $\mathcal{Q}$ such that **CERTAINTY**($\mathcal{Q}$) is in **P** but not first-order expressible. This question was answered affirmatively by Wijsen [Wij10b].

Kolaitis and Pema [KP12] showed that for every Boolean query $\mathcal{Q}$ in **SJFCQ** with exactly two atoms, **CERTAINTY**($\mathcal{Q}$) is either in **P** or **coNP**-complete, and it is decidable which of the two is the case. More recently, for all Boolean queries $\mathcal{Q}$ in **SJFCQ**, an effective complexity classification of **CERTAINTY**($\mathcal{Q}$) into three classes (**FO**, **P**, **coNP**-complete) was established by Koutris and Wijsen [KW17].

All aforementioned results assume queries without self-join. For queries $\mathcal{Q}$ with self-joins, only fragmentary results about the complexity of the problem **CERTAINTY**($\mathcal{Q}$) are known [CM05, Wij09].

The counting variant of **CERTAINTY**($\mathcal{Q}$), written $\sharp$**CERTAINTY**($\mathcal{Q}$), takes as input an uncertain database **db** and asks to determine the number of

repairs of **db** that satisfy Boolean query $\mathcal{Q}$. As shown in [Wij13], this problem is intimately related to query answering in block-independent-disjoint (BID) probabilistic databases [DRS09, DRS11]. Maslowski and Wijsen [MW13] have proved that for every Boolean query $\mathcal{Q}$ in **SJFCQ**, the counting problem $\sharp$**CERTAINTY**($\mathcal{Q}$) is either in **FP** or $\sharp$**P**-complete, and it is decidable which of the two is the case.

In the past, the paradigm of CQA has been implemented in expressive formalisms, such as Disjunctive Logic Programming [GGZ03] and Binary Integer Programming (BIP) [KPT13]. In these formalisms, it is relatively easy to express an algorithm that computes consistent answers to conjunctive queries under primary key constraints. The drawback is that these algorithms may, in the worst case, take exponential time in cases where, in theory, consistent answers are computable in polynomial time or expressible in first-order logic. In the latter case, the consistent answer can be computed by a single SQL query using standard database technology, including query optimization. In [Ber11, page 38], the author mentions that logic programs for CQA cannot compete with solutions in first-order logic when they exist. Likewise, in an experimental comparison of EQUIP [KPT13] and ConQuer [FFM05], the authors of the former system found that BIP never outperformed solutions in SQL.

Chapter 4

Presence of Satisfied Constraints

In this chapter, we introduce the problem **CERTAINTY**($\mathcal{Q}$) in the presence of a set Σ of dependencies. The problem **CERTAINTY**($\mathcal{Q}, \Sigma$) takes as input an uncertain database **db** that satisfies Σ, and asks whether every repair of **db** satisfies $\mathcal{Q}$.

This chapter extends our article [GPW14] in the following way: while the results in [GPW14] assume that the input conjunctive query $\mathcal{Q}$ is acyclic (in the sense of [BFMY83]), no such assumption is made in the current chapter.

4.1 Motivation

Consider the relations `Employee` and `Place` in Figure 4.1. Employees are uniquely identified by their first and last name. The relation `Place` gives touristic appreciations for cities in terms of Michelin stars. There is uncertainty about the salary and city of Ed Smith, about the touristic value of the city of Acri, and about the country of Mons.

All existing works on **CERTAINTY**($\mathcal{Q}$) have assumed that primary keys are the only integrity constraints involved, and that they can be violated at any one time. However, in practice, some primary keys or some other

Employee

First	Last	Birth	Salary	City	Country
Ed	Smith	1960	50 000	Acri	Italy
Ed	Smith	1960	60 000	Mons	Belgium
An	Allen	1970	40 000	Mons	Belgium

Place

City	Country	Stars
Acri	Italy	**
Acri	Italy	***
Mons	Belgium	***
Mons	France	***

Figure 4.1: Uncertain database satisfying `Employee` : **City** → **Country**.

constraints may well be satisfied. This happens, for example, when some (but not all) constraints are enforced by the database system. In this chapter, we study the problem **CERTAINTY**($\mathcal{Q}$) *in the presence of a set* Σ *of functional and join dependencies.* The problem **CERTAINTY**($\mathcal{Q}, \Sigma$) takes as input an uncertain database **db** *that satisfies* Σ, and asks whether $\mathcal{Q}$ evaluates to true on every repair of **db**. We next illustrate the interest of this problem by two examples.

Example 15. *Although the relation* `Employee` *in Figure 4.1 violates its primary key, it can be observed that birth years are unique for each employee, and that the functional dependency (FD)* `Employee` : ***City*** → ***Country*** *(call it* σ_{15}*) holds (but* `Place` : ***City*** → ***Country*** *is violated).*

Consider the conjunctive query $\mathcal{Q}_{15}$ *asking whether some employees live in three-star cities:*

$$\mathcal{Q}_{15} = \exists u\, \exists v\, \exists w\, \exists x\, \exists y\, \exists z \left(\texttt{Employee}(\underline{u, v}, x, y, z) \wedge \texttt{Place}(\underline{y}, z, ***)\right)$$

From Theorem 3, it follows that **CERTAINTY**($\mathcal{Q}_{15}$) *is not in* **FO** *(and from [KW17, Theorem 3.2], it follows that the problem is* **coNP***-complete).*

Nevertheless, the results developed in the current chapter will demonstrate that $\mathbf{CERTAINTY}(\mathcal{Q}_{15}, \{\sigma_{15}\})$ *is first-order expressible and hence in the low complexity class* **FO**. *In practice, this means that the problem can be solved by a single SQL query, which is shown near the end of the chapter in Listing 4.1. This example shows that in the presence of an FD that is satisfied, the complexity of consistent query answering can significantly decrease, from intractable to highly tractable, even if that FD is not a key dependency.*

Example 16. *The relation* `Employee` *of Figure 4.1 contains two tuples about* Ed Smith. *One may want to consider that the first tuple contains the correct salary (*50 000*), and the second tuple the correct domicile (*Mons*). This is possible in value-based repairing but impossible in tuple-based repairing, where either the first or the second tuple has to be selected in a repair.*

As advocated in [Wij06], to simulate value-based repairing, we may want to chase uncertain databases with join dependencies (JDs) to "distribute" uncertain values. Figure 4.2 shows the result of chasing `Employee` *with the following JD (call it* σ_{16}*):*

$$\texttt{Employee} :\bowtie \left[\begin{array}{l} \{\boldsymbol{First}, \boldsymbol{Last}, \boldsymbol{Birth}\}, \\ \{\boldsymbol{First}, \boldsymbol{Last}, \boldsymbol{Salary}\}, \\ \{\boldsymbol{First}, \boldsymbol{Last}, \boldsymbol{City}, \boldsymbol{Country}\} \end{array}\right].$$

The effect is that we obtain for each employee all combinations of salaries and cities. A repair can now select the third tuple, stating that Ed *earns* 50 000 *and lives in* Mons.

On the other hand, the attributes ***City*** *and* ***Country*** *are considered as one composite attribute in* σ_{16}*, that is, even though* Ed *is associated with both city* Acri *(first row of* `Employee`*) and country* Belgium *(second row of* `Employee`*), we do not want to consider the combination* Acri Belgium.

The three components of the above JD σ_{16} *share the primary key* $\{\boldsymbol{First}, \boldsymbol{Last}\}$ *of* `Employee`. *We will use the term* key join dependency *(KJD) for JDs in which all two distinct components share the primary key (and share no other attribute). In general, if we chase a database with a KJD* σ *prior to*

First	Last	Birth	Salary	City	Country
Ed	Smith	1960	50 000	Acri	Italy
Ed	Smith	1960	60 000	Acri	Italy
Ed	Smith	1960	50 000	Mons	Belgium
Ed	Smith	1960	60 000	Mons	Belgium
An	Allen	1970	40 000	Mons	Belgium

Figure 4.2: Relation that results from the chase of `Employee` with JD σ_{16}.

consistent query answering, we obtain a database that satisfies σ. This means that for any Boolean query $\mathcal{Q}$, we shift from **CERTAINTY**$(\mathcal{Q})$ *to the problem* **CERTAINTY**$(\mathcal{Q}, \{\sigma\})$*, using the information that σ holds.*

In this chapter, we study the complexity of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ when $\mathcal{Q}$ is an acyclic Boolean conjunctive query without self-join, and Σ is a set of FDs and KJDs, containing at most one KJD per relation name. In particular, we show that it is decidable to determine whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible (and hence in the low complexity class **FO**). This contribution is of practical relevance, because it allows us to decide which cases of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ can be solved by standard "first-order" SQL capabilities.

Conceptually, our work adds a new flavor to consistent query answering [ABC99, Ber11]. Existing works in this field have always assumed that all constraints can be potentially violated. In our approach, we distinguish two classes of constraints: those that may be violated (primary keys in our setting), and those that are known to be satisfied. In practice, one may learn the satisfied constraints in various ways: one may simply look at the data and "mine" dependencies that hold; one may know that the database system enforces some constraints; or one may apply a KJD to simulate value-based repairing [Wij06], as illustrated by Example 16. In any way, the knowledge about satisfied constraints can be favorably exploited in consistent query answering.

This chapter is organized as follows. Section 4.2 first introduces some theoretical notions, and then states the problem. Section 4.3 extends the notion of attack graph to deal with FDs and KJDs, and shows that it is decidable, given $\mathcal{Q}$ and Σ, whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible. Section 4.4 explains how to effectively construct a first-order definition of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ if it exists. Finally, Section 4.5 concludes the chapter.

4.2 Problem Statement

Let R be a relation name with signature $[n, k]$. A *key join dependency* (KJD) has the form $\mathtt{R} :\bowtie [K_1, K_2, \ldots, K_l]$ with $l \geq 1$ such that

1. for $1 \leq i \leq l$, $K_i \subseteq \{1, 2, \ldots, n\}$;
2. $K_1 \cup K_2 \cup \ldots \cup K_l = \{1, 2, \ldots, n\}$;
3. for every $1 \leq i < j \leq l$, we have $K_i \neq K_j$ and $K_i \cap K_j = \{1, 2, \ldots, k\}$.

A *functional dependency* (FD) has the form

$$\mathtt{R} : i_1, i_2, \ldots, i_m \rightarrow l$$

where $1 \leq i_1 \leq i_2 \leq \ldots \leq i_m \leq n$ and $1 \leq l \leq n$, $l \notin \{i_1, i_2, \ldots, i_m\}$. We will assume $m \geq 1$, although the technical treatment can be easily extended to treat FDs with an empty left-hand side. The following definition of satisfaction of KJDs and FDs is standard (see, e.g., [AHV95, page 159]). Let **db** be an uncertain database with some relation R. We say that two R-facts $\mathtt{R}(\underline{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k}, \mathsf{a}_{k+1}, \mathsf{a}_{k+2}, \ldots, \mathsf{a}_n)$ and $\mathtt{R}(\underline{\mathsf{b}_1, \mathsf{b}_2, \ldots, \mathsf{b}_k}, \mathsf{b}_{k+1}, \mathsf{b}_{k+2}, \ldots, \mathsf{b}_n)$ *agree on position i* if $\mathsf{a}_i = \mathsf{b}_i$, where $i \in \{1, 2, \ldots, n\}$. We say that **db** *satisfies* the KJD $\mathtt{R} : \bowtie [K_1, K_2, \ldots, K_l]$ if whenever $A_1, A_2, \ldots, A_l$ are key-equal R-facts of **db**, then there exists an R-fact $B \in \mathbf{db}$ such that for all $i \in \{1, 2, \ldots, l\}$, B and A_i agree on all positions in K_i. We say that **db** *satisfies* $\mathtt{R} : i_1, i_2, \ldots, i_m \rightarrow l$ if for all R-facts $A, B \in \mathbf{db}$, if A and B agree on all positions among $i_1, i_2, \ldots, i_m$, then they agree on position l. If σ is an FD or a KJD, then we write $\mathbf{db} \models \sigma$ to denote that **db** satisfies σ.

We say that a set Σ of KJDs and FDs is *jd-singular* if it does not contain two distinct KJDs with the same relation name; the number of FDs per relation name is not restricted.

Let $\mathcal{Q}$ be a Boolean query in **SJFCQ**. Let Σ be a jd-singular set of KJDs and FDs. The problem of consistent query answering in the presence of constraints is the following.

Problem CERTAINTY$(\mathcal{Q}, \Sigma)$

Input Uncertain database **db** such that **db** $\models \Sigma$

Question Does every repair of **db** satisfy $\mathcal{Q}$?

If $\Sigma = \emptyset$, then **CERTAINTY**$(\mathcal{Q}, \Sigma)$ and **CERTAINTY**$(\mathcal{Q})$ are the same problem. On the other extreme, if Σ captures all primary key constraints, then input databases contain no primary key violations, and **CERTAINTY**$(\mathcal{Q}, \Sigma)$ asks, given consistent database **db**, whether $\mathcal{Q}$ evaluates to true on **db**. The problem **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is in **coNP**, because if **CERTAINTY**$(\mathcal{Q}, \Sigma)$ has "no" as its answer, then a "no"-certificate is a repair of **db** that satisfies Σ and falsifies $\mathcal{Q}$. We are interested in deciding its complexity for varying $\mathcal{Q}$ and Σ, in particular, given $\mathcal{Q}$ and Σ:

1. Is it decidable to determine whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible?
2. Is it decidable to determine whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is in **P**?
3. Is it decidable to determine whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is **coNP**-hard?

Saying that **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible is tantamount to saying that there exists a first-order sentence φ such that for every uncertain database **db** that satisfies Σ, the following are equivalent:

- every repair of **db** satisfies $\mathcal{Q}$;
- **db** $\models \varphi$.

As in Chapter 3 for **CERTAINTY**($\mathcal{Q}$), we call such a first-order sentence φ a first-order definition of **CERTAINTY**($\mathcal{Q}, \Sigma$), or alternatively, a *consistent first-order rewriting of* $\mathcal{Q}$ *relative to* Σ. Again its practical interest is obvious: φ can be encoded in SQL and executed on any uncertain database by means of standard database technology.

The following Theorem is the main result of this chapter.

Theorem 5. *Given* $\mathcal{Q}$ *and* Σ*, it is decidable whether* **CERTAINTY**($\mathcal{Q}, \Sigma$) *is first-order expressible, where*

- $\mathcal{Q}$ *is a Boolean self-join-free conjunctive query, and*
- Σ *is a jd-singular set of KJDs and FDs.*

Moreover, if **CERTAINTY**($\mathcal{Q}, \Sigma$) *is first-order expressible, then a first-order definition of* **CERTAINTY**($\mathcal{Q}, \Sigma$) *can be effectively constructed.*

Theorem 5 is a fairly deep result of practical interest. Its proof will be developed from Section 4.3 on. The proof relies on the hypothesis that Σ contains at most one KJD per relation name. Nevertheless, for the intended purpose of simulating update-based repairing (cf. Example 16), one KJD per relation suffices.

4.3 Extending Attack Graph

Attack graphs have been defined in Section 3.2. In this section, given a Boolean query $\mathcal{Q}$ in **SJFCQ** and a set Σ of dependencies, we compute a new query denoted $\mathcal{Q} \otimes \Sigma$. This new query will be conjunctive and self-join-free. The main result will be that **CERTAINTY**($\mathcal{Q}, \Sigma$) is first-order expressible if and only if the attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic (this is Theorem 4).

The operator $\otimes$ transforms query atoms and database facts according to FDs and KJDs. We provide an example before giving the technical definition.

Example 17. *Assume an atom* $F = \mathtt{R}(\underline{\mathtt{a}_1, \mathtt{a}_2}, \mathtt{a}_3, \mathtt{a}_4, \mathtt{a}_5)$ *with signature* $[5, 2]$.

- *An FD* $\mathtt{R} : 2, 3 \rightarrow 4$ *(call it* σ*) will add to* F *two new atoms* $\mathtt{R}_1^{\sigma}(\underline{\mathtt{a}_2, \mathtt{a}_3}, \mathtt{a}_4)$ *and* $\mathtt{R}_2^{\sigma}(\underline{\mathtt{a}_2, \mathtt{a}_3}, \mathtt{a}_4)$. *Here,* $\mathtt{R}_1^{\sigma}$ *and* $\mathtt{R}_2^{\sigma}$ *are two new relation names which depend on* σ. *The two atoms differ in their relation names but are otherwise identical.*

- *A KJD* $\mathtt{R} : \bowtie [\{1, 2, 3\}, \{1, 2, 4, 5\}]$ *will replace the atom* F *with three atoms* $\mathtt{R}_{\circ}^{\bowtie}(\underline{\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_4, \mathtt{a}_5})$, $\mathtt{R}_1^{\bowtie}(\underline{\mathtt{a}_1, \mathtt{a}_2}, \mathtt{a}_3)$, *and* $\mathtt{R}_2^{\bowtie}(\underline{\mathtt{a}_1, \mathtt{a}_2}, \mathtt{a}_4, \mathtt{a}_5)$. *Here,* $\mathtt{R}_{\circ}^{\bowtie}$ *is a new relation name that is all-key, and* $\mathtt{R}_1^{\bowtie}$, $\mathtt{R}_2^{\bowtie}$ *are new relation names corresponding to the first and the second component of the KJD.*

In the following, we denote by $\mathbf{qschema}(\mathcal{Q})$ the set of relation names used in $\mathcal{Q}$.

Definition 4. *Let* $\mathcal{Q}$ *be a Boolean query in* ***SJFCQ***. *Let* $\mathbf{db}$ *be an uncertain database such that all relation names of* $\mathbf{db}$ *belong to* $\mathbf{qschema}(\mathcal{Q})$.

Let $\Sigma = \Sigma_1 \cup \Sigma_2$ *where* Σ_1 *is a jd-singular set of KJDs and* Σ_2 *is a set of FDs (with zero or more FDs per relation name). The Boolean self-join-free conjunctive query* $\mathcal{Q} \otimes \Sigma$ *and the uncertain database* $\mathbf{db} \otimes \Sigma$ *are defined as follows.*

For every atom $F = \mathtt{R}(\underline{s_1, s_2, \ldots, s_k}, s_{k+1}, s_{k+2}, \ldots, s_n)$ *of* $\mathcal{Q}$:

1. *If* Σ_1 *contains no KJD for* $\mathtt{R}$, *then* $\mathcal{Q} \otimes \Sigma$ *contains* F *and* $\mathbf{db} \otimes \Sigma$ *contains all* $\mathtt{R}$*-facts of* $\mathbf{db}$.

2. *If* Σ_1 *contains a KJD for* $\mathtt{R}$, *then* $\mathtt{R}_{\circ}^{\bowtie}(\underline{s_1, s_2, \ldots, s_n})$ *is an atom of* $\mathcal{Q} \otimes \Sigma$ *where* $\mathtt{R}_{\circ}^{\bowtie}$ *is a new relation name with signature* $[n, n]$. *That is,* $\mathtt{R}_{\circ}^{\bowtie}$ *is all-key.*

   For every $\mathtt{R}(\underline{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_k}, \mathtt{a}_{k+1}, \mathtt{a}_{k+2}, \ldots, \mathtt{a}_n)$ *of* $\mathbf{db}$, *it is the case that* $\mathbf{db} \otimes \Sigma$ *contains* $\mathtt{R}_{\circ}^{\bowtie}(\underline{\mathtt{a}_1, \mathtt{a}_2, \ldots, \mathtt{a}_n})$.

3. *If* Σ_1 *contains KJD* $\mathtt{R} : \bowtie [K_1, K_2, \ldots, K_l]$, *then for each* $i \in \{1, 2, \ldots, l\}$, *the query* $\mathcal{Q} \otimes \Sigma$ *contains a new* $\mathtt{R}_i^{\bowtie}$*-atom.*

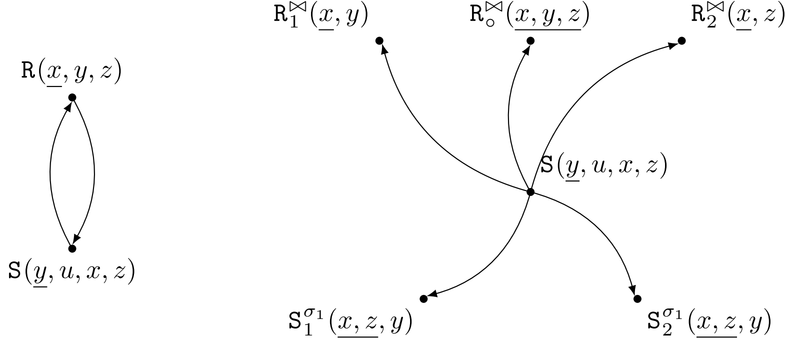

Figure 4.3: Attack graphs of $\mathcal{Q}_1$ (left) and $\mathcal{Q}_1 \otimes \Sigma_1$ (right).

If $K_i = \{1, 2, \ldots, k, j_1, j_2, \ldots, j_m\}$ *with* $1 < 2 < \cdots < k < j_1 < j_2 < \cdots < j_m \leq n$ *then the new* $\mathtt{R}_i^{\bowtie}$*-atom is* $\mathtt{R}_i^{\bowtie}(\underline{s_1, s_2, \ldots, s_k}, s_{j_1}, s_{j_2}, \ldots, s_{j_m})$*, where* $\mathtt{R}_i^{\bowtie}$ *is a new relation name of signature* $[k+m, k]$*.*

For every $\mathtt{R}(\underline{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k}, \mathsf{a}_{k+1}, \mathsf{a}_{k+2}, \ldots, \mathsf{a}_n)$ *of* **db***, it is the case that* $\mathbf{db} \otimes \Sigma$ *contains* $\mathtt{R}_i^{\bowtie}(\underline{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k}, \mathsf{a}_{j_1}, \mathsf{a}_{j_2}, \ldots, \mathsf{a}_{j_m})$*.*

4. *If* Σ_2 *contains some FD* $\mathtt{R} : i_1, i_2, \ldots, i_m \to l$ *(call it* σ*), then* $\mathcal{Q} \otimes \Sigma$ *contains two new atoms* $\mathtt{R}_1^{\sigma}(\underline{s_{i_1}, s_{i_2}, \ldots, s_{i_m}}, l)$ *and* $\mathtt{R}_2^{\sigma}(\underline{s_{i_1}, s_{i_2}, \ldots, s_{i_m}}, l)$*, where* $\mathtt{R}_1^{\sigma}$ *and* $\mathtt{R}_2^{\sigma}$ *are two new relation names with signature* $[m+1, m]$*.*

   For every $\mathtt{R}(\underline{\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_k}, \mathsf{a}_{k+1}, \mathsf{a}_{k+2}, \ldots, \mathsf{a}_n) \in \mathbf{db}$*, it is the case that* $\mathbf{db} \otimes \Sigma$ *contains* $\mathtt{R}_1^{\sigma}(\underline{\mathsf{a}_{i_1}, \mathsf{a}_{i_2}, \ldots, \mathsf{a}_{i_m}}, \mathsf{a}_l)$ *and* $\mathtt{R}_2^{\sigma}(\underline{\mathsf{a}_{i_1}, \mathsf{a}_{i_2}, \ldots, \mathsf{a}_{i_m}}, \mathsf{a}_l)$*.*

5. $\mathcal{Q} \otimes \Sigma$ *contains no other atoms than those specified in this definition;* $\mathbf{db} \otimes \Sigma$ *contains no other facts than those specified in this definition.*

We refer to relation names $\mathtt{R}_\circ^{\bowtie}$*,* $\mathtt{R}_i^{\bowtie}$*,* $\mathtt{R}_1^{\sigma}$*, and* $\mathtt{R}_2^{\sigma}$ *as* spurious relation names*. Atoms with a spurious relation name are called* spurious atoms*.*

Example 18. *Let* $\mathcal{Q}_1 = \{\mathtt{R}(\underline{x}, y, z), \mathtt{S}(\underline{y}, u, x, z)\}$*. Let* Σ_1 *be the set of dependencies containing KJD* $\mathtt{R} : \bowtie [\{1, 2\}, \{1, 3\}]$ *and FD* $\mathtt{S} : 3, 4 \to 1$ *(call it* σ_1*). The query* $\mathcal{Q}_1 \otimes \Sigma_1$ *contains the following atoms:*

- $\mathtt{R}_{\circ}^{\bowtie}(\underline{x,y,z})$ *where* $\mathtt{R}_{\circ}^{\bowtie}$ *has signature* $[3,3]$;
- $\mathtt{R}_{1}^{\bowtie}(\underline{x},y)$ *where* $\mathtt{R}_{1}^{\bowtie}$ *has signature* $[2,1]$ *and corresponds to the first component of the KJD;*
- $\mathtt{R}_{2}^{\bowtie}(\underline{x},z)$ *where* $\mathtt{R}_{2}^{\bowtie}$ *has signature* $[2,1]$ *and corresponds to the second component of the KJD;*
- $\mathtt{S}(\underline{y},u,x,z)$. *Notice that* Σ_1 *contains no KJD for* $\mathtt{S}$; *and*
- $\mathtt{S}_1^{\sigma_1}(\underline{x,z},y)$ *and* $\mathtt{S}_2^{\sigma_1}(\underline{x,z},y)$, *both of signature* $[3,2]$.

The attack graphs of $\mathcal{Q}_1$ *and* $\mathcal{Q}_1 \otimes \Sigma_1$ *are shown on Figure 4.3.*

The following lemma states that the operator $\otimes$ is first-order expressible.

Lemma 1. *Let* $\mathcal{Q}$ *be a Boolean query in* ***SJFCQ***. *Let* Σ *be a jd-singular set of KJDs and FDs. For every spurious relation name* $\mathtt{S}$ *in* $\mathcal{Q} \otimes \Sigma$, *there exists a first-order query* $\psi_{\mathtt{S}}$ *such that for every uncertain database* **db** *over* **qschema**$(\mathcal{Q})$, *the set of* $\mathtt{S}$*-facts of* $\mathbf{db} \otimes \Sigma$ *is equal to* $\psi_{\mathtt{S}}(\mathbf{db})$.

Proof. For every spurious relation name $\mathtt{S}$, the query $\psi_{\mathtt{S}}$ is (by Definition 4) a projection. □

Lemma 2. *Let* $\mathcal{Q}$ *be a Boolean query in* ***SJFCQ***. *Let* $\Sigma = \Sigma_1 \cup \Sigma_2$ *where* Σ_1 *is a jd-singular set of KJDs and* Σ_2 *is a set of FDs. The following statements are equivalent for every uncertain database* **db** *over* **qschema**$(\mathcal{Q})$ *that satisfies* Σ:

1. *every repair of* **db** *satisfies* $\mathcal{Q}$;
2. *every repair of* $\mathbf{db} \otimes \Sigma$ *satisfies* $\mathcal{Q} \otimes \Sigma$.

Proof. For every uncertain database **db** over **qschema**$(\mathcal{Q})$, let $[\![\mathbf{db} \otimes \Sigma]\!]$ be the smallest subset of $\mathbf{db} \otimes \Sigma$ such that $[\![\mathbf{db} \otimes \Sigma]\!]$ contains all facts with spurious relation names of the form $\mathtt{R}_{\circ}^{\bowtie}$ or $\mathtt{R}_{i}^{\sigma}$, where $\mathtt{R} \in$ **qschema**$(\mathcal{Q})$, $\sigma \in \Sigma_2$, and $i \in \{1,2\}$. Note that $[\![\mathbf{db} \otimes \Sigma]\!]$ does not contain facts with relation names of the form $\mathtt{R}_{i}^{\bowtie}$.

Let f be the function with domain **repairs**(**db**) mapping each repair r of **db** to $f(r) := (r \otimes \Sigma) \cup [\![\mathbf{db} \otimes \Sigma]\!]$. We show that for every uncertain database **db** over **qschema**($\mathcal{Q}$) such that $\mathbf{db} \models \Sigma$,

$$\mathbf{repairs}(\mathbf{db} \otimes \Sigma) = \{f(r) \mid r \in \mathbf{repairs}(\mathbf{db})\} \tag{4.1}$$

$\boxed{\supseteq}$ Let r be a repair of **db** with $\mathbf{db} \models \Sigma$. We need to show that $f(r)$ is a repair of $\mathbf{db} \otimes \Sigma$. Since relation names of the form $\mathtt{R}_{\circ}^{\bowtie}$ are all-key and since $\mathbf{db} \models \Sigma_2$, it follows that $[\![\mathbf{db} \otimes \Sigma]\!]$ is consistent. Consequently, it suffices to show that $f(r) \setminus [\![\mathbf{db} \otimes \Sigma]\!]$ is a maximal consistent subset of $(\mathbf{db} \otimes \Sigma) \setminus [\![\mathbf{db} \otimes \Sigma]\!]$. The set $f(r) \setminus [\![\mathbf{db} \otimes \Sigma]\!] = (r \otimes \Sigma) \setminus [\![\mathbf{db} \otimes \Sigma]\!]$ contains two types of relation names.

Relation names $\mathtt{R} \in \mathbf{qschema}(\mathcal{Q}) \cap \mathbf{qschema}(\mathcal{Q} \otimes \Sigma)$.

Then, Σ_1 contains no KJD for $\mathtt{R}$. Then **db** and $\mathbf{db} \otimes \Sigma$ contain the same set of $\mathtt{R}$-facts. Likewise, for every repair r of **db**, we have that r and $r \otimes \Sigma$ contain the same set of $\mathtt{R}$-facts. Clearly, the set of $\mathtt{R}$-facts in $r \otimes \Sigma$ is a repair of the set of $\mathtt{R}$-facts in $\mathbf{db} \otimes \Sigma$.

Relation names $\mathtt{R}_i^{\bowtie} \in \mathbf{qschema}(\mathcal{Q} \otimes \Sigma)$ with $\mathtt{R} \in \mathbf{qschema}(\mathcal{Q})$.

Assume $\mathtt{R}$ has signature $[n, k]$. We can assume a KJD $\bowtie [K_1, K_2, \dots, K_l]$ in Σ_1 such that $K_i = \{1, 2, \dots, k, j_1, j_2, \dots, j_m\}$ with $1 < 2 < \cdots < k < j_1 < j_2 < \cdots < j_m \leq n$. Assume $\mathbf{db} \otimes \Sigma$ contains $\mathtt{R}_i^{\bowtie}(\underline{\mathsf{a}_1, \mathsf{a}_2, \dots, \mathsf{a}_k}, \mathsf{b}_{k+1}, \mathsf{b}_{k+2}, \dots, \mathsf{b}_m)$. Then, **db** contains some fact $F = \mathtt{R}(\underline{\mathsf{a}_1, \mathsf{a}_2, \dots, \mathsf{a}_k}, \mathsf{a}_{k+1}, \mathsf{a}_{k+2}, \dots, \mathsf{a}_n)$ such that for $i \in \{1, 2, \dots, m\}$, we have $\mathsf{a}_{j_i} = \mathsf{b}_i$. Since r contains exactly one $\mathtt{R}$-fact that is key-equal to F, it follows that $r \otimes \Sigma$ contains exactly one $\mathtt{R}_i^{\bowtie}$-fact that is key-equal to $\mathtt{R}(\underline{\mathsf{a}_1, \mathsf{a}_2, \dots, \mathsf{a}_k}, \mathsf{b}_{k+1}, \mathsf{b}_{k+2}, \dots, \mathsf{b}_m)$. Consequently, the set of $\mathtt{R}_i^{\bowtie}$-facts in $r \otimes \Sigma$ is a repair of the set of $\mathtt{R}_i^{\bowtie}$-facts in $\mathbf{db} \otimes \Sigma$.

We conclude that $f(r)$ is a repair of $\mathbf{db} \otimes \Sigma$.

$\boxed{\subseteq}$ Let r' be a repair of $\mathbf{db} \otimes \Sigma$ with $\mathbf{db} \models \Sigma$. We specify the construction of a (unique) repair r of **db** such that $r' = f(r)$. For every relation name

$\mathtt{R} \in \mathbf{qschema}(\mathcal{Q})$ we proceed as follows. If Σ_1 contains no KJD for R, then we include in r all R-facts of r'. Assume next that Σ_1 contains a KJD $\mathtt{R} : \bowtie [K_1, \ldots, K_l]$ (call it σ), where the signature of R is $[n, k]$. Since $\mathbf{db} \models \sigma$, it is the case that for every $i \in \{1, \ldots, l\}$, for every $\mathtt{R}_i^{\bowtie}$-fact A_i of r', there exists a (unique) R-fact A in $\mathbf{db}$ such that $A_i \in \{A\} \otimes \{\sigma\} \subseteq r'$; we include every such A in r. Note that $\{A\} \otimes \{\sigma\}$ contains a number l of distinct facts that agree on positions $\{1, \ldots, k\}$ and whose join is A. Since Σ is jd-singular, the set r so constructed is consistent. It is now obvious that r is a repair of $\mathbf{db}$ such that $r' = f(r)$. This concludes the proof of (4.1).

To conclude the proof, it suffices to note the following easy equivalences for every $r \in \mathbf{repairs}(\mathbf{db})$:

$$r \models \mathcal{Q} \quad \Leftrightarrow \quad r \otimes \Sigma \models \mathcal{Q} \otimes \Sigma$$
$$r \otimes \Sigma \models \mathcal{Q} \otimes \Sigma \quad \Leftrightarrow \quad f(r) \models \mathcal{Q} \otimes \Sigma.$$

□

The following examples illustrate that in the proof of Lemma 2, it is important to require that $\mathbf{db} \models \Sigma$.

Example 19. *Let* $\mathcal{Q} = \{\mathtt{R}(\underline{x}, y, z)\}$ *and let* Σ *be the singleton containing KJD* $\mathtt{R} : \bowtie [\{1, 2\}, \{1, 3\}]$. *We have* $\mathcal{Q} \otimes \Sigma = \{\mathtt{R}_\circ^{\bowtie}(\underline{x, y, z}), \mathtt{R}_1^{\bowtie}(\underline{x}, y), \mathtt{R}_2^{\bowtie}(\underline{x}, z)\}$.

Let $\mathbf{db} = \{\mathtt{R}(\underline{\mathsf{a}}, \mathsf{b}, \mathsf{b}'), \mathtt{R}(\underline{\mathsf{a}}, \mathsf{c}, \mathsf{c}')\}$, *which falsifies the KJD in* Σ. *We have* $\mathbf{db} \otimes \Sigma = \{\mathtt{R}_\circ^{\bowtie}(\underline{\mathsf{a}, \mathsf{b}, \mathsf{b}'}), \mathtt{R}_\circ^{\bowtie}(\underline{\mathsf{a}, \mathsf{c}, \mathsf{c}'}), \mathtt{R}_1^{\bowtie}(\underline{\mathsf{a}}, \mathsf{b}), \mathtt{R}_1^{\bowtie}(\underline{\mathsf{a}}, \mathsf{c}), \mathtt{R}_2^{\bowtie}(\underline{\mathsf{a}}, \mathsf{b}'), \mathtt{R}_2^{\bowtie}(\underline{\mathsf{a}}, \mathsf{c}')\}$.

Clearly, every repair of $\mathbf{db}$ *satisfies* $\mathcal{Q}$. *However,* $\{\mathtt{R}_\circ^{\bowtie}(\underline{\mathsf{a}, \mathsf{b}, \mathsf{b}'}), \mathtt{R}_\circ^{\bowtie}(\underline{\mathsf{a}, \mathsf{c}, \mathsf{c}'}), \mathtt{R}_1^{\bowtie}(\underline{\mathsf{a}}, \mathsf{b}), \mathtt{R}_2^{\bowtie}(\underline{\mathsf{a}}, \mathsf{c}')\}$ *is a repair of* $\mathbf{db} \otimes \Sigma$ *that falsifies* $\mathcal{Q} \otimes \Sigma$.

Example 20. *Let* $\mathcal{Q} = \{\mathtt{R}(\underline{x}, y, z)\}$ *and let* Σ *be the singleton containing the FD* $\mathtt{R} : 2 \rightarrow 3$ *(call it* σ*). We have* $\mathcal{Q} \otimes \Sigma = \{\mathtt{R}(\underline{x}, y, z), \mathtt{R}_1^{\sigma}(\underline{y}, z), \mathtt{R}_2^{\sigma}(\underline{y}, z)\}$.

Let $\mathbf{db} = \{\mathtt{R}(\underline{\mathsf{a}}, \mathsf{c}, \mathsf{d}), \mathtt{R}(\underline{\mathsf{b}}, \mathsf{c}, \mathsf{e})\}$, *which falsifies* σ. *We have* $\mathbf{db} \otimes \Sigma = \{\mathtt{R}(\underline{\mathsf{a}}, \mathsf{c}, \mathsf{d}), \mathtt{R}(\underline{\mathsf{b}}, \mathsf{c}, \mathsf{e}), \mathtt{R}_1^{\sigma}(\underline{\mathsf{c}}, \mathsf{d}), \mathtt{R}_1^{\sigma}(\underline{\mathsf{c}}, \mathsf{e}), \mathtt{R}_2^{\sigma}(\underline{\mathsf{c}}, \mathsf{d}), \mathtt{R}_2^{\sigma}(\underline{\mathsf{c}}, \mathsf{e})\}$.

Clearly, every repair of $\mathbf{db}$ *satisfies* $\mathcal{Q}$. *However,* $\{\mathtt{R}(\underline{\mathsf{a}}, \mathsf{c}, \mathsf{d}), \mathtt{R}(\underline{\mathsf{b}}, \mathsf{c}, \mathsf{e}), \mathtt{R}_1^{\sigma}(\underline{\mathsf{c}}, \mathsf{d}), \mathtt{R}_2^{\sigma}(\underline{\mathsf{c}}, \mathsf{e})\}$ *is a repair of* $\mathbf{db} \otimes \Sigma$ *that faslifies* $\mathcal{Q} \otimes \Sigma$.

Theorem 4. *Let $\mathcal{Q}$ be a Boolean query in **SJFCQ**. Let Σ be a jd-singular set of KJDs and FDs. The following statements are equivalent:*

1. **CERTAINTY**$(\mathcal{Q}, \Sigma)$ *is first-order expressible.*
2. *The attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic.*

Proof. $\boxed{1 \Rightarrow 2}$ Proof by contraposition. Assume the attack graph of $\mathcal{Q} \otimes \Sigma$ is cyclic. By [KW17, Lemma 3.6], the attack graph of $\mathcal{Q} \otimes \Sigma$ contains two atoms, say $\tilde{F}$ and $\tilde{G}$, that mutually attack each other. It can be easily verified that:

1. if $\tilde{F}$ is an $\mathtt{S}_i^{\bowtie}$-atom and $\tilde{G}$ a $\mathtt{T}_j^{\bowtie}$-atom, where relation names $\mathtt{S}_i^{\bowtie}$ and $\mathtt{T}_j^{\bowtie}$ come from KJDs for $\mathtt{S}$ and $\mathtt{T}$ respectively, then $\mathtt{S} \neq \mathtt{T}$;
2. the attack graph of $\mathcal{Q} \otimes \Sigma$ contains no attacks starting from an $\mathtt{R}_\circ^{\bowtie}$-atom, where the relation name $\mathtt{R}_\circ^{\bowtie}$ comes from a KJD for $\mathtt{R}$, because $\mathtt{R}_\circ^{\bowtie}$ is all-key; and
3. the attack graph of $\mathcal{Q} \otimes \Sigma$ contains no attacks starting from $\mathtt{R}_i^{\sigma}$-atoms, where the relation name $\mathtt{R}_i^{\sigma}$ comes from an FD $\sigma \in \Sigma$ and $i \in \{1, 2\}$.

Consequently, we can assume distinct relation names $\mathtt{S}$, $\mathtt{T}$ and positive integers i, j such that $\tilde{F}$ is either an $\mathtt{S}$-atom or an $\mathtt{S}_i^{\bowtie}$-atom, and $\tilde{G}$ is either a $\mathtt{T}$-atom or a $\mathtt{T}_j^{\bowtie}$-atom.

Let F and G be the $\mathtt{S}$-atom and $\mathtt{T}$-atom of $\mathcal{Q}$ respectively. Notice that F and $\tilde{F}$ agree on all primary-key positions. Likewise, G and $\tilde{G}$ agree on all primary-key positions.

Note incidentally that from $F^{+,\mathcal{Q}} \subseteq \tilde{F}^{+,\mathcal{Q}\otimes\Sigma}$ and $G^{+,\mathcal{Q}} \subseteq \tilde{G}^{+,\mathcal{Q}\otimes\Sigma}$, it follows that F and G mutually attack each other in the attack graph of $\mathcal{Q}$.

We will show that **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is **L**-hard (and hence not in **FO**). The proof is a first-order reduction from **CERTAINTY**$(\mathcal{Q}_0)$, with $\mathcal{Q}_0 = \{\mathtt{R}_0(\underline{x}, y), \mathtt{S}_0(\underline{y}, x)\}$, to **CERTAINTY**$(\mathcal{Q}, \Sigma)$. The result then follows from **L**-hardness of **CERTAINTY**$(\mathcal{Q}_0)$ [KW17, Lemma 4.2].

We denote by f the first-order reduction from **CERTAINTY**$(\mathcal{Q}_0)$ to **CERTAINTY**$(\mathcal{Q} \otimes \Sigma)$, as specified in the proof of [KW17, Lemma 4.2]. If

db is a legal input to **CERTAINTY**($\mathcal{Q}_0$), then $f(\mathbf{db})$ is a legal input to **CERTAINTY**($\mathcal{Q} \otimes \Sigma$) with the following properties:

- The only relations in $f(\mathbf{db})$ that can be inconsistent are the relations corresponding to $\tilde{F}$ and $\tilde{G}$.
- For every KJD $\mathtt{R} : \bowtie [K_1, \dots, K_l]$ in Σ, among all relations $\mathtt{R}_1^{\bowtie}, \dots, \mathtt{R}_l^{\bowtie}$ in $f(\mathbf{db})$,[1] our construction ensures that at most one will be inconsistent, which happens if $\tilde{F}$ or $\tilde{G}$ has a relation name in $\mathtt{R}_1^{\bowtie}, \dots, \mathtt{R}_l^{\bowtie}$. Since we can always renumber indexes, we can assume without loss of generality that for every $i \in \{1, \dots, l-1\}$, $\mathtt{R}_i^{\bowtie}$ is consistent (but $\mathtt{R}_l^{\bowtie}$ may be inconsistent). It is straightforward to show the following variant of Heath's theorem: if $K, A_1, A_2, \dots, A_l$ are mutually disjoint sets of attributes, then $\{K \to A_1, K \to A_2, \dots, K \to A_{l-1}\} \models \bowtie [KA_1, KA_2, \dots, KA_{l-1}, KA_l]$. Note that the set of functional dependencies does not contain $K \to A_l$. It follows that the relation $\mathtt{R}_\circ^{\bowtie}$ in $f(\mathbf{db})$ will satisfy the KJD $\bowtie [K_1, \dots, K_l]$.
- Whenever Σ contains an FD σ on $\mathtt{R}$, then $\mathtt{R}_1^{\sigma}$ and $\mathtt{R}_2^{\sigma}$ are both distinct from $\tilde{F}$ and distinct from $\tilde{G}$, and thus their relations will be consistent in $f(\mathbf{db})$.

It is now straightforward to construct, in **FO** complexity, a database $\mathbf{db}_0$ that satisfies Σ such that that $\mathbf{db}_0 \otimes \Sigma = f(\mathbf{db})$. Indeed, $\mathbf{db}_0$ deletes from $f(\mathbf{db})$ all $\mathtt{R}_i^{\sigma}$-facts where σ is an FD, deletes from $f(\mathbf{db})$ all $\mathtt{R}_i^{\bowtie}$-facts with $i \geq 1$, and replaces every fact $\mathtt{R}_\circ^{\bowtie}(\underline{\mathtt{a}_1, \dots, \mathtt{a}_n})$ with $\mathtt{R}(\underline{\mathtt{a}_1, \dots, \mathtt{a}_k}, \mathtt{a}_{k+1}, \dots, \mathtt{a}_n)$ where the signature of $\mathtt{R}$ is $[n, k]$. From Lemma 2, it follows that the following are equivalent:

1. every repair of $f(\mathbf{db}) = \mathbf{db}_0 \otimes \Sigma$ satisfies $\mathcal{Q} \otimes \Sigma$;
2. every repair $\mathbf{db}_0$ satisfies $\mathcal{Q}$.

Since first-order reductions compose, it is correct to conclude that there exists a first-order reduction from **CERTAINTY**($\mathcal{Q}_0$) to **CERTAINTY**($\mathcal{Q}, \Sigma$).

[1] By an abuse of terminology, when we refer to relation $\mathtt{R}_1^{\bowtie}$ in $f(\mathbf{db})$, we mean the set of $\mathtt{R}_1^{\bowtie}$-facts in $f(\mathbf{db})$.

$\boxed{2 \Rightarrow 1}$ Assume the attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic. By Theorem 4, **CERTAINTY**$(\mathcal{Q} \otimes \Sigma)$ is first-order expressible. We can assume a first-order formula ψ such that for every uncertain database $\tilde{\mathbf{db}}$ over **qschema**$(\mathcal{Q} \otimes \Sigma)$, we have that ψ evaluates to true on $\tilde{\mathbf{db}}$ if and only if every repair of $\tilde{\mathbf{db}}$ satisfies $\mathcal{Q} \otimes \Sigma$.

For every uncertain database **db** over **qschema**$(\mathcal{Q})$, it is the case that $\mathbf{db} \otimes \Sigma$ is an uncertain database over **qschema**$(\mathcal{Q} \otimes \Sigma)$. It is correct to conclude that for every uncertain database **db** over **qschema**$(\mathcal{Q})$, we have that ψ evaluates to true on $\mathbf{db} \otimes \Sigma$ if and only if every repair of $\mathbf{db} \otimes \Sigma$ satisfies $\mathcal{Q} \otimes \Sigma$.

By Lemma 1, $\mathbf{db} \otimes \Sigma$ is first-order computable from **db**. Consequently, there exists a first-order formula $\tilde{\psi}$ such that for every uncertain database **db** over **qschema**$(\mathcal{Q})$, we have that $\tilde{\psi}$ evaluates to true on **db** if and only if every repair of $\mathbf{db} \otimes \Sigma$ satisfies $\mathcal{Q} \otimes \Sigma$. In particular, for every uncertain database **db** that satisfies Σ, the following are equivalent:

1. $\tilde{\psi}$ evaluates to true on **db**.
2. Every repair of $\mathbf{db} \otimes \Sigma$ satisfies $\mathcal{Q} \otimes \Sigma$.

Then, by Lemma 2, for every uncertain database **db** that satisfies Σ, the following are equivalent:

1. $\tilde{\psi}$ evaluates to true on **db**.
2. Every repair of **db** satisfies $\mathcal{Q}$.

Consequently, **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible. $\square$

Importantly, it happens that the attack graph of $\mathcal{Q}$ is cyclic, and the attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic. Thus, by Theorem 3 and Theorem 4, there are cases where **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible, but **CERTAINTY**$(\mathcal{Q})$ is not. We illustrate this by two examples.

Example 21. *Figure 4.4 shows the cyclic attack graph of $\mathcal{Q}_2$. Figure 4.5 shows the acyclic attack graph of $\mathcal{Q}_2 \otimes \{\sigma_2\}$ where $\sigma_2 = \mathtt{R} : 5 \rightarrow 6$. By*

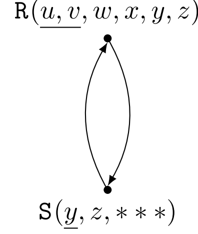

Figure 4.4: Attack graph of $\mathcal{Q}_2$.

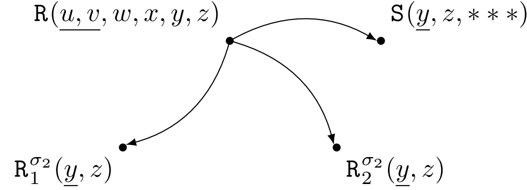

Figure 4.5: Attack graph $\mathcal{Q}_2 \otimes \{\sigma_2\}$ with $\sigma_2 = \mathtt{R} : 5 \rightarrow 6$.

Theorem 3 and Theorem 4, **CERTAINTY**$(\mathcal{Q}_2)$ *is not first-order expressible, but* **CERTAINTY**$(\mathcal{Q}_2, \{\sigma_2\})$ *is first-order expressible.*

Example 22. *Consider again the query* $\mathcal{Q}_1 = \{\mathtt{R}(\underline{x}, y, z), \mathtt{S}(\underline{y}, u, x, z)\}$ *introduced in Example 18. The attack graph of* $\mathcal{Q}_1$ *is cyclic (as shown in Figure 4.3 (left)). Figure 4.3 (right) shows the attack graph of* $\mathcal{Q}_1 \otimes \Sigma_1$ *with* $\Sigma_1 = \{\mathtt{R} : \bowtie [\{1,2\}, \{1,3\}], \mathtt{S} : 3, 4 \rightarrow 1\}$*. The latter attack graph is acyclic.*

Incidentally, one can easily verify that if we delete the KJD and/or the FD from Σ_1*, then the attack graph remains cyclic. That is, both the KJD and the FD are needed to attain an acyclic attack graph.*

The proof of Theorem 5 can now be given.

Theorem 5. *Given* $\mathcal{Q}$ *and* Σ*, it is decidable whether* **CERTAINTY**$(\mathcal{Q}, \Sigma)$ *is first-order expressible, where*

- $\mathcal{Q}$ *is a Boolean self-join-free conjunctive query, and*

- *Σ is a jd-singular set of KJDs and FDs.*

Moreover, if **CERTAINTY**$(\mathcal{Q}, \Sigma)$ *is first-order expressible, then a first-order definition of* **CERTAINTY**$(\mathcal{Q}, \Sigma)$ *can be effectively constructed.*

Proof. By Theorem 4, **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible if and only if the attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic. Acyclicity of attack graphs can be tested in quadratic time [KW17, Lemma 3.3]. Furthermore, the proof of Theorem 4 is constructive, meaning that it constructs a first-order definition of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ if it exists. □

4.4 Construction of Consistent First-Order Rewritings

Assume **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible. The proof of Theorem 4 implies that a first-order definition of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ can be constructed in two steps: start by constructing a first-order definition ψ of **CERTAINTY**$(\mathcal{Q} \otimes \Sigma)$, and then substitute away from ψ all atoms with spurious relation names. In this section, we develop a simpler approach in which there is no need for these spurious relation names.

4.4.1 Attack Graph of $(\mathcal{Q}, \Sigma)$

The following definition extends the notion of attack graph to deal with the presence of KJDs and FDs.

Definition 5. *Let $\mathcal{Q}$ be a Boolean query in* ***SJFCQ****. Let Σ be a jd-singular set of KJDs and FDs. The* attack graph of $(\mathcal{Q}, \Sigma)$ *is a directed graph whose vertices are the atoms of $\mathcal{Q}$. If the attack graph of $\mathcal{Q} \otimes \Sigma$ contains an attack $F \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} G$ where F is an* $\mathtt{R}$*-atom or an* $\mathtt{R}_i^{\bowtie}$*-atom, and G is an* $\mathtt{S}$*-atom or an* $\mathtt{S}_j^{\bowtie}$*-atom, then the attack graph of $(\mathcal{Q}, \Sigma)$ contains a directed edge from the* $\mathtt{R}$*-atom of $\mathcal{Q}$ to the* $\mathtt{S}$*-atom of $\mathcal{Q}$. Notice that spurious relation names* $\mathtt{R}_1^{\sigma}$, $\mathtt{R}_2^{\sigma}$, $\mathtt{S}_1^{\sigma}$, $\mathtt{S}_2^{\sigma}$*, originating from some FD σ, play no role in the construction of the attack graph of $(\mathcal{Q}, \Sigma)$.*

Example 23. *This example continues Example 22. From the attack graph of* $\mathcal{Q}_1 \otimes \Sigma_1$ *in Figure 4.3 (right), one finds that the attack graph of* $(\mathcal{Q}_1, \Sigma_1)$ *consists of a single directed edge from* $\mathtt{S}(\underline{y}, u, x, z)$ *to* $\mathtt{R}(\underline{x}, y, z)$.

Clearly, if $\Sigma = \emptyset$, then the attack graph of $(\mathcal{Q}, \Sigma)$ is the same graph as the attack graph of $\mathcal{Q}$. The following lemma states that the attack graphs of $\mathcal{Q} \otimes \Sigma$ and $(\mathcal{Q}, \Sigma)$ are either both cyclic or both acyclic.

Lemma 3. *Let* $\mathcal{Q}$ *be a Boolean query in* ***SJFCQ***. *Let* Σ *be a jd-singular set of KJDs and FDs. The following are equivalent:*

1. *the attack graph of* $(\mathcal{Q}, \Sigma)$ *is acyclic;*
2. *the attack graph of* $\mathcal{Q} \otimes \Sigma$ *is acyclic.*

Proof. $\boxed{1 \Longrightarrow 2}$ Proof by contraposition. Assume that the attack graph of $\mathcal{Q} \otimes \Sigma$ is cyclic. By [KW17, Lemma 3.6], the attack graph of $\mathcal{Q} \otimes \Sigma$ must contain a cycle of size 2. So we can assume two atoms F, G such that $F \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} F$. For every FD σ defined on $\mathtt{R}$, the corresponding atoms $\mathtt{R}_1^{\sigma}(\underline{\vec{x}}, \vec{y})$ and $\mathtt{R}_2^{\sigma}(\underline{\vec{x}}, \vec{y})$ contain no outgoing attacks. We can assume without loss of generality that for some $\mathtt{R} \in$ **qschema**$(\mathcal{Q})$, it is the case that F is an $\mathtt{R}$-atom or an $\mathtt{R}_i^{\bowtie}$-atom. Since no $\mathtt{R}_i^{\bowtie}$-atom attacks an $\mathtt{R}_j^{\bowtie}$-atom, it must be the case that G is an $\mathtt{S}$-atom or an $\mathtt{S}_k^{\bowtie}$-atom for some $\mathtt{S} \neq \mathtt{R}$ ($\mathtt{S} \in$ **qschema**$(\mathcal{Q})$). Then the attack graph of $(\mathcal{Q}, \Sigma)$ contains a cycle involving the $\mathtt{R}$-atom and $\mathtt{S}$-atom of $\mathcal{Q}$.

$\boxed{2 \Longrightarrow 1}$ Assume that the attack graph of $\mathcal{Q} \otimes \Sigma$ is acyclic. By [KW17, Lemma 3.5], the attack graph of $\mathcal{Q} \otimes \Sigma$ is transitive. Assume towards a contradiction that the attack graph of $(\mathcal{Q}, \Sigma)$ contains a directed cycle. Then it must be the case that the attack graph of $\mathcal{Q} \otimes \Sigma$ contains an elementary path from some $\mathtt{R}_i^{\bowtie}$-atom (call it $\tilde{F}$) to some $\mathtt{R}_j^{\bowtie}$-atom (call it $\tilde{G}$) where $\mathtt{R} \in$ **qschema**$(\mathcal{Q})$ and $i \neq j$. Since the attack graph of $\mathcal{Q} \otimes \Sigma$ is transitive, it follows $\tilde{F} \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} \tilde{G}$. This contradicts the obvious observation that the attack graph of $\mathcal{Q} \otimes \Sigma$ contains no directed edge from an $\mathtt{R}_i^{\bowtie}$-atom to some $\mathtt{R}_j^{\bowtie}$-atom. $\square$

The following lemma is technical. It states that if $\mathtt{R}(\underline{\vec{x}}, \vec{y})$ is unattacked in the attack graph of $(\mathcal{Q}, \Sigma)$ and if Σ contains a KJD $\mathtt{R} : \bowtie [K_1, K_2, \ldots, K_l]$,

then in the attack graph of $\mathcal{Q} \otimes \Sigma$,

1. each $\mathtt{R}_i^{\bowtie}$-atom is unattacked ($i \in \{1, 2, \ldots, l\}$);

2. the $\mathtt{R}_\circ^{\bowtie}$-atom can only be attacked by some $\mathtt{R}_i^{\bowtie}$-atom; and

3. if σ is an FD of Σ, then the $\mathtt{R}_1^{\sigma}$-atom and the $\mathtt{R}_2^{\sigma}$-atom can only be attacked by some $\mathtt{R}_i^{\bowtie}$-atom.

Lemma 4. *Let $\mathcal{Q}$ be a Boolean query in **SJFCQ**. Let Σ be a jd-singular set of KJDs and FDs. Assume that Σ contains a KJD $\mathtt{R} : \bowtie [K_1, K_2, \ldots, K_l]$. Let F be the $\mathtt{R}$-atom of $\mathcal{Q}$. Let $\tilde{F}$ be the $\mathtt{R}_\circ^{\bowtie}$-atom of $\mathcal{Q} \otimes \Sigma$, and for $i \in \{1, 2, \ldots, l\}$, let F_i be the $\mathtt{R}_i^{\bowtie}$-atom of $\mathcal{Q} \otimes \Sigma$. If F is unattacked in the attack graph of $(\mathcal{Q}, \Sigma)$, then*

1. *$\{F_1, F_2, \ldots, F_l\}$ are unattacked in the attack graph of $\mathcal{Q} \otimes \Sigma$;*

2. *for every atom $G \in \mathcal{Q} \otimes \Sigma$, if $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} \tilde{F}$, then $G \in \{F_1, F_2, \ldots, F_l\}$; and*

3. *if $\mathcal{Q} \otimes \Sigma$ contains an atom H with relation name $\mathtt{R}_i^{\sigma}$ for some FD $\sigma \in \Sigma$, then for each $G \in \mathcal{Q} \otimes \Sigma$, if $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} H$, then $G \in \{F_1, F_2, \ldots, F_l\}$.*

Proof. Assume F is unattacked in the attack graph of $(\mathcal{Q}, \Sigma)$.

$\boxed{1}$ Assume towards a contradiction that for some atom $G \in \mathcal{Q} \otimes \Sigma$, we have $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} F_i$ (for some $i \in \{1, 2, \ldots, l\}$). Then F is attacked in the attack graph of $(\mathcal{Q}, \Sigma)$, a contradiction.

$\boxed{2}$ Assume towards a contradiction that for some $G \in (\mathcal{Q} \otimes \Sigma) \setminus \{F_1, F_2, \ldots, F_l\}$, we have $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} \tilde{F}$. Since $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} \tilde{F}$, we can assume a variable $x \in \mathbf{vars}(\tilde{F})$ such that $x \notin G^{+,\mathcal{Q}\otimes\Sigma}$. Since there exists $i \in \{1, 2, \ldots, l\}$ such that $x \in \mathbf{vars}(\tilde{F}) \cap \mathbf{vars}(F_i)$, it follows $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} F_i$, contradicting property $\boxed{1}$ shown above.

$\boxed{3}$ Let H be an atom of $\mathcal{Q} \otimes \Sigma$ with relation name $\mathtt{R}_i^{\sigma}$. Assume $G \in \mathcal{Q} \otimes \Sigma$ such that $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} H$. Obviously, $G \neq \tilde{F}$. From $\mathbf{vars}(H) \subseteq \mathbf{vars}(\tilde{F})$, it follows $G \overset{\mathcal{Q}\otimes\Sigma}{\rightsquigarrow} \tilde{F}$. By property $\boxed{2}$ shown above, $G \in \{F_1, F_2, \ldots, F_l\}$. □

4.4.2 Free Variables

So far, we have assumed that all queries are Boolean. In the construction of first-order definitions, we will have to treat queries with free variables. In virtue of Theorem 2, it is possible to construct a consistent first-order rewriting of a non-Boolean self-join-free conjunctive query if we can build such a rewriting for the same query in which every free variable is replaced by some constant. However it is not immediate that this theorem also applies to the setting of this chapter.

In the following, the notation $\mathcal{Q}(u_1, u_2, \ldots, u_l)$, where $u_1, u_2, \ldots, u_l$ are distinct variables, indicates that the variables in $u_1, u_2, \ldots, u_l$ are free in the query $\mathcal{Q}$. The problem of consistent query answering in the presence of satisfied constraints naturally extends to queries with free variables. Definition 6 straightforwardly extends the definition of **CERTAINTY**$(\mathcal{Q}(\vec{x}))$ given in Section 3.1 by adding a set Σ of constraints.

Definition 6. *Let $\mathcal{Q}(u_1, u_2, \ldots, u_l)$ be a conjunctive query with free variables $u_1, u_2, \ldots, u_l$. Let Σ be a set of first-order constraints.*

We define **CERTAINTY**$(\mathcal{Q}(u_1, u_2, \ldots, u_l), \Sigma)$ *as the function problem that takes as input an uncertain database* **db** *and that returns as output all sequences $(\mathsf{a}_1, \ldots, \mathsf{a}_l)$ of constants, of length l, such that every repair of* **db** *satisfies $\mathcal{Q}(\mathsf{a}_1, \ldots, \mathsf{a}_l)$.*

A first-order definition *of* **CERTAINTY**$(\mathcal{Q}(u_1, u_2, \ldots, u_l), \Sigma)$ *is a first-order formula $\varphi(u_1, u_2, \ldots, u_l)$ such that for every sequence $\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_l$ of constants, for every uncertain database* **db** *that satisfies Σ, the following are equivalent:*

1. *every repair of* **db** *satisfies $\mathcal{Q}(\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_l)$;*

2. **db** $\models \varphi(\mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_l)$.

The addition of free variables is not fundamental. Assume we are asked to determine a first-order definition of **CERTAINTY**$(\mathcal{Q}(u_1, u_2, \ldots, u_l), \Sigma)$ where $\mathcal{Q}$ is a self-join-free conjunctive query. Let $\mathsf{c}_1, \mathsf{c}_2, \ldots, \mathsf{c}_l$ be distinct

constants not occurring in $\mathcal{Q}$. Let $\mathcal{Q}_{u_1,u_2,\ldots,u_l \to \mathsf{c}_1,\mathsf{c}_2,\ldots,\mathsf{c}_l}$ be the query obtained from $\mathcal{Q}$ by replacing each occurrence of u_i with c_i, for $i \in \{1, 2, \ldots, l\}$. Let φ be a first-order definition of **CERTAINTY**$(\mathcal{Q}_{u_1,u_2,\ldots,u_l \to \mathsf{c}_1,\mathsf{c}_2,\ldots,\mathsf{c}_l}, \Sigma)$. Clearly, the query $\mathcal{Q}_{u_1,u_2,\ldots,u_l \to \mathsf{c}_1,\mathsf{c}_2,\ldots,\mathsf{c}_l}$ is Boolean. Since first-order definitions treat all constants in a generic fashion, one can obtain a first-order definition of **CERTAINTY**$(\mathcal{Q}(u_1, u_2, \ldots, u_l), \Sigma)$ by replacing each c_i with u_i in φ. This is tantamount to saying that free variables are treated as constants. Likewise, in the computation of attack graphs, free variables are to be treated as constants.

4.4.3 The Function rewrite$^{\Sigma}(\mathcal{Q})$

In this section, we extend *Function Rewrite* introduced in Chapter 3 to take into consideration jd-singular sets of KJDs and FDs. The function **rewrite**$^{\Sigma}(\mathcal{Q})$ is almost identical to **rewrite**$(\mathcal{Q})$ introduced in Chapter 3. The only difference is that **rewrite**$^{\Sigma}(\mathcal{Q})$ rewrites atoms in a topological order of the attack graph of $(\mathcal{Q}, \Sigma)$. We nevertheless give a detailed definition of **rewrite**$^{\Sigma}(\mathcal{Q})$ so that we can rely upon it in the proof of Theorem 6.

Definition 7. *Let $\mathcal{Q}(\vec{u})$ be a query in **SJFCQ**. Let Σ be a jd-singular set of KJDs and FDs such that the attack graph of $(\mathcal{Q}(\vec{u}), \Sigma)$ is acyclic. We define* **rewrite**$^{\Sigma}(\mathcal{Q}(\vec{u}))$ *recursively as follows.*

Case $|\mathcal{Q}| = 0$. *Then* **rewrite**$^{\Sigma}(\mathcal{Q}(\vec{u})) =$ ***true****.*

Case $|\mathcal{Q}| > 0$. *Choose an atom $F = \mathtt{R}(\underline{\vec{x}}, y_1, \ldots, y_m)$ that is unattacked in the attack graph of $(\mathcal{Q}, \Sigma)$. Let $z_1, \ldots, z_m$ be distinct variables and let C be a conjunction of equalities constructed as follows. For $i \in \{1, \ldots, m\}$,*

- *if y_i is a variable that does not occur in $\vec{u}$ nor in $\langle \vec{x}, y_1, \ldots, y_{i-1} \rangle$, then z_i is the same variable as y_i;*
- *otherwise z_i is a new variable and C contains $z_i = y_i$. Notice that this case applies if y_i is a constant, if y_i is a (free) variable in $\vec{u}$, or if y_i is a variable occuring in $\langle \vec{x}, y_1, \ldots, y_{i-1} \rangle$.*

Let $\vec{v}$ be a sequence of variables that contains exactly once each variable that occurs in $\langle \vec{x}, y_1, \ldots, y_{i-1} \rangle$ and that does not occur in $\vec{u}$. Then

$$\begin{aligned}\mathbf{rewrite}^{\Sigma}(\mathcal{Q}(\vec{u})) \quad = \quad & \exists \vec{v}(\mathtt{R}(\underline{\vec{x}}, y_1, \ldots, y_n) \wedge \\ & \forall z_1 \cdots \forall z_m(\mathtt{R}(\underline{\vec{x}}, z_1, \ldots, z_m) \Rightarrow \\ & C \wedge \mathbf{rewrite}^{\Sigma}(\mathcal{Q}'(\vec{u}, \vec{v})))),\end{aligned}$$

where $\mathcal{Q}' = \mathcal{Q} \setminus \{\mathtt{R}(\underline{\vec{x}}, y_1, \ldots, y_m)\}$. Notice that the variables $\vec{u}$ that are free in $\mathcal{Q}(\vec{u})$ are also free in $\mathbf{rewrite}^{\Sigma}(\mathcal{Q}(\vec{u}))$.

Notice that by [KW17, Lemma 3.7] and Lemma 3, the attack graph of $(\mathcal{Q}'(\vec{u}, \vec{v}), \Sigma)$ is acyclic and hence the recursive call $\mathbf{rewrite}(\mathcal{Q}'(\vec{u}, \vec{v}), \Sigma)$ is well defined.

Theorem 6. *Let $\mathcal{Q}$ and Σ be as in Definition 7. If $\mathbf{CERTAINTY}(\mathcal{Q}, \Sigma)$ is first-order expressible, then a first-order definition of it is given by $\mathbf{rewrite}^{\Sigma}(\mathcal{Q})$.*

Proof. Assume $\mathbf{CERTAINTY}(\mathcal{Q}(\vec{u}), \Sigma)$ is first-order expressible. By Theorem 4, the attack graph of $\mathcal{Q}(\vec{u}) \otimes \Sigma$ is acyclic. By Lemma 3, the attack graph of $(\mathcal{Q}(\vec{u}), \Sigma)$ is acyclic. Consequently, the rewrite function of Definition 7 applies to $(\mathcal{Q}(\vec{u}), \Sigma)$ and to $(\mathcal{Q}(\vec{u}) \otimes \Sigma, \emptyset)$. Let

$$\varphi(\vec{u}) = \mathbf{rewrite}^{\Sigma}(\mathcal{Q}(\vec{u}));$$

$$\psi(\vec{u}) = \mathbf{rewrite}^{\emptyset}(\mathcal{Q}(\vec{u}) \otimes \Sigma).$$

It is known [KW17] that $\psi(\vec{u})$ is a consistent first-order rewriting for $\mathcal{Q}(\vec{u}) \otimes \Sigma$. By Lemma 2, it suffices to show that for every uncertain database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, for every sequence $\vec{a}$ of constants (where $\vec{a}$ has the same length as $\vec{u}$),

$$\mathbf{db} \models \varphi(\vec{a}) \Leftrightarrow \mathbf{db} \otimes \Sigma \models \psi(\vec{a}).$$

The proof runs by induction on the number $|\mathcal{Q}|$ of atoms in $\mathcal{Q}$. The result is obvious if $|\mathcal{Q}| = 0$. For the induction step, assume $|\mathcal{Q}| > 0$.

Assume that the attack graph of $(\mathcal{Q}(\vec{u}), \Sigma)$ contains an unattacked atom $\mathtt{R}(\underline{\vec{x}}, y_1, \ldots, y_m)$, as in Definition 7. In the proof, we will assume that Σ contains a KJD $\mathtt{R} : \bowtie [K_1, \ldots, K_l]$ (call it σ) for $\mathtt{R}$. This assumption is without loss of

generality, because if Σ contained no KJD for $\mathtt{R}$, we can always add a trivial KJD $\mathtt{R} : \bowtie [\{1, \dots, n\}]$ where n is the arity of $\mathtt{R}$. We have

$$\varphi(\vec{u}) \quad = \quad \begin{array}{l} \exists\vec{v}(\mathtt{R}(\underline{\vec{x}}, y_1, \dots, y_m) \wedge \\ \quad \forall z_1 \cdots \forall z_m(\mathtt{R}(\underline{\vec{x}}, z_1, \dots, z_m) \Rightarrow \\ \quad\quad (C \wedge \varphi'(\vec{u}, \vec{v})))), \end{array} \tag{4.2}$$

where

- $\vec{v}, z_1, \dots, z_m, C$ are as in Definition 7; and
- $\varphi'(\vec{u}, \vec{v}) = \mathbf{rewrite}^{\Sigma}(\mathcal{Q}'(\vec{u}, \vec{v}))$ with $\mathcal{Q}' = \mathcal{Q} \setminus \{\mathtt{R}(\underline{\vec{x}}, y_1, \dots, y_m)\}$.

By Lemma 4, no $\mathtt{R}_i^{\bowtie}$-atom ($1 \leq i \leq l$) is attacked in the attack graph of $\mathcal{Q} \otimes \Sigma$. Let

$$\tilde{\psi}(\vec{u}) \quad = \quad \begin{array}{l} \exists\vec{v}(\mathtt{R}_1^{\bowtie}(\underline{\vec{x}}, \vec{y}_1) \wedge \cdots \wedge \mathtt{R}_l^{\bowtie}(\underline{\vec{x}}, \vec{y}_l) \wedge \\ \quad \forall z_1 \cdots \forall z_m(\mathtt{R}_1^{\bowtie}(\underline{\vec{x}}, \vec{z}_1) \wedge \cdots \wedge \mathtt{R}_l^{\bowtie}(\underline{\vec{x}}, \vec{z}_l) \Rightarrow \\ \quad\quad (C \wedge \psi'(\vec{u}, \vec{v})))), \end{array} \tag{4.3}$$

where

- $\{\mathtt{R}_i^{\bowtie}(\underline{\vec{x}}, \vec{y}_i)\}_{i=1}^{l} = \{\mathtt{R}(\underline{\vec{x}}, y_1, \dots, y_m)\} \otimes \{\sigma\}$;
- $\{\mathtt{R}_i^{\bowtie}(\underline{\vec{x}}, \vec{z}_i)\}_{i=1}^{l} = \{\mathtt{R}(\underline{\vec{x}}, z_1, \dots, z_m)\} \otimes \{\sigma\}$; and
- $\psi'(\vec{u}, \vec{v}) = \mathbf{rewrite}^{\emptyset}(\mathcal{Q}'(\vec{u}, \vec{v}) \otimes \Sigma)$.

That is, $\tilde{\psi}(\vec{u})$ rewrites the $\mathtt{R}_i^{\bowtie}$-atoms, but completely ignores the $\mathtt{R}_\circ^{\bowtie}$-atom and the $\mathtt{R}_i^{\delta}$-atoms for any functional dependency $\delta \in \Sigma$. It is not hard to convince oneself (see also Example 26 and syntactic simplifications introduced in Chapter 6) that for every uncertain database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, for every sequence $\vec{a}$ of constants (where $\vec{a}$ has same length as $\vec{u}$),

$$\mathbf{db} \otimes \Sigma \models \tilde{\psi}(\vec{a}) \Leftrightarrow \mathbf{db} \otimes \Sigma \models \psi(\vec{a}).$$

In particular, we argue next why we can safely ignore in (4.3) conjuncts that result from "rewriting" $\mathtt{R}_\circ^{\bowtie}$-atoms and $\mathtt{R}_i^{\delta}$-atoms for any functional dependency $\delta \in \Sigma$.

Rationale for omitting $\mathtt{R}_{\circ}^{\bowtie}$-atoms. Assume $F = \mathtt{R}_{\circ}^{\bowtie}(\underline{\vec{x}, y_1, \dots, y_m})$ is an unattacked atom of $\mathcal{Q}'$ where all variables in $\vec{x}, y_1, \dots, y_m$ are free. We have $\mathbf{rewrite}^{\emptyset}(\mathcal{Q}') = \mathtt{R}_{\circ}^{\bowtie}(\underline{\vec{x}, y_1, \dots, y_m}) \wedge \mathbf{rewrite}^{\emptyset}(\mathcal{Q}' \setminus \{F\})$. By the construction in Definition 4, if $\mathbf{db} \otimes \Sigma$ contains facts $\mathtt{R}_{1}^{\bowtie}(\underline{\vec{x}}, \vec{y}_1), \dots, \mathtt{R}_{l}^{\bowtie}(\underline{\vec{x}}, \vec{y}_l)$, then it contains $\mathtt{R}_{\circ}^{\bowtie}(\underline{\vec{x}, y_1, \dots, y_m})$. So the conjunct $\mathtt{R}_{\circ}^{\bowtie}(\underline{\vec{x}, y_1, \dots, y_m})$ can be omitted in $\mathbf{rewrite}^{\emptyset}(\mathcal{Q}')$.

Rationale for omitting $\mathtt{R}_{i}^{\delta}$-atoms. Assume $F = \mathtt{R}_{i}^{\delta}(\underline{\vec{s}}, w)$ is an unattacked atom of $\mathcal{Q}'$ where all variables in $\vec{s}, w$ are free. We have

$$\mathbf{rewrite}^{\emptyset}(\mathcal{Q}') = \mathtt{R}_{i}^{\delta}(\underline{\vec{s}}, w) \wedge \forall z \left(\mathtt{R}_{i}^{\delta}(\underline{\vec{s}}, z) \Rightarrow \left(z = w \wedge \mathbf{rewrite}^{\emptyset}(\mathcal{Q}' \setminus \{F\})\right)\right).$$

For every uncertain database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, it will be the case that $\mathbf{db} \otimes \Sigma$ contains no two facts $\mathtt{R}_{i}^{\delta}(\underline{\vec{a}}, \mathtt{b})$, $\mathtt{R}_{i}^{\delta}(\underline{\vec{a}}, \mathtt{c})$ with $\mathtt{b} \neq \mathtt{c}$. The equality $z = w$ in the above rewriting will thus always evaluate to true. Consequently, there is no need to rewrite the atom $\mathtt{R}_{i}^{\delta}$, which can thus be omitted.

From [KW17, Lemma 3.7], it follows that the attack graph of $\mathcal{Q}'(\vec{u}, \vec{v}) \otimes \Sigma$ is acyclic. From Theorem 4, it follows that $\mathbf{CERTAINTY}(\mathcal{Q}'(\vec{u}, \vec{v}), \Sigma)$ is first-order expressible. By the induction hypothesis, for every uncertain database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, for all sequences $\vec{a}, \vec{b}$ of constants,

$$\mathbf{db} \models \varphi'(\vec{a}, \vec{b}) \Leftrightarrow \mathbf{db} \otimes \Sigma \models \psi'(\vec{a}, \vec{b}).$$

From the form (4.2) and (4.3), it is correct to conclude that for every uncertain database $\mathbf{db}$ such that $\mathbf{db} \models \Sigma$, for every sequence $\vec{a}$ of constants (where $\vec{a}$ has same length as $\vec{u}$),

$$\mathbf{db} \models \varphi(\vec{a}) \Leftrightarrow \mathbf{db} \otimes \Sigma \models \tilde{\psi}(\vec{a}).$$

This concludes the proof. □

Example 24. *We can now explain all technical details behind Example 15 introduced in Section 4.2. Figure 4.4 on page 68 shows the attack graph*

Figure 4.6: The attack graph of $(\mathcal{Q}_2, \{\mathtt{R} : 5 \rightarrow 6\})$.

R	**First**	**Last**	**Birth**	**Sal**	**City**	**Country**
	Ed	Smith	1960	50K	Mons	Belgium
	An	Allen	1970	40K	Mons	France

S	**City**	**Country**	**Stars**
	Mons	Belgium	$*\,*\,*$
	Mons	France	$*\,*\,*$

Figure 4.7: Uncertain database falsifying $\mathtt{R} : City \rightarrow Country$.

of $\mathcal{Q}_2$*. Since the attack graph is cyclic, we conclude (by Theorem 4) that* **CERTAINTY**$(\mathcal{Q}_2)$ *is not first-order expressible. Figure 4.5 on page 68 shows the attack graph of* $\mathcal{Q}_2 \otimes \{\mathtt{R} : 5 \rightarrow 6\}$*. Since the attack graph is acyclic, we conclude (by Theorem 4) that* **CERTAINTY**$(\mathcal{Q}_2, \{\mathtt{R} : 5 \rightarrow 6\})$ *is first-order expressible.*

The attack graph of $(\mathcal{Q}_2, \{\mathtt{R} : 5 \rightarrow 6\})$ *is shown in Figure 4.6. Based on this attack graph,* $\mathbf{rewrite}^{\{\mathtt{R}:5\rightarrow 6\}}(\mathcal{Q}_2)$ *yields the following first-order sentence.*

$$
\begin{aligned}
&\varphi_2 = \\
&\quad \exists u\, \exists v\, \exists w\, \exists x\, \exists y\, \exists z (\mathtt{R}(\underline{u, v}, w, x, y, z) \wedge \\
&\quad\quad \forall w\, \forall x\, \forall y\, \forall z (\mathtt{R}(\underline{u, v}, w, x, y, z) \Rightarrow \\
&\quad\quad\quad (\mathtt{S}(\underline{y}, z, *\,*\,*) \wedge \\
&\quad\quad\quad\quad \forall z'\, \forall s (\mathtt{S}(\underline{y}, z', s) \Rightarrow \\
&\quad\quad\quad\quad\quad z' = z \wedge s = *\,*\,*))))
\end{aligned}
$$

By Theorem 6, φ_2 *is a first-order definition of* **CERTAINTY**$(\mathcal{Q}_2, \{\mathtt{R} : 5 \rightarrow 6\})$*. Thus, for every uncertain database* **db** *that satisfies* $\mathtt{R} : 5 \rightarrow 6$*, it is the case that* φ_2 *evaluates to true on* **db** *if and only if* $\mathcal{Q}_2$ *evaluates to true on every repair of* **db**.

```
SELECT 'yes' FROM R AS R1
WHERE NOT EXISTS (
  SELECT * FROM R AS R2
  WHERE R2.FIRST = R1.FIRST
  AND   R2.LAST = R1.LAST
  AND   NOT EXISTS (
    SELECT * FROM S AS S1
    WHERE S1.CITY = R2.CITY
    AND   S1.COUNTRY = R2.COUNTRY
    AND   S1.STARS = '***'
    AND   NOT EXISTS (
      SELECT * FROM S AS S2
      WHERE S2.City = S1.CITY
      AND (
        S2.Country <> S1.Country
        OR S2.Stars <> '***'))));
```

Listing 4.1: Certain SQL rewriting.

Unsurprisingly, φ_2 does not provide consistent answers on uncertain databases that falsify $\mathtt{R} : 5 \rightarrow 6$. *For example, φ_2 evaluates to false on the uncertain database of Figure 4.7, even though all repairs of this database satisfy $\mathcal{Q}_2$.*

Translating a first-order definition of **CERTAINTY**$(\mathcal{Q}, \Sigma)$ into SQL is fairly straightforward. The performance of such SQL queries has been studied in [DPW12].

Example 25. *The first-order sentence φ_2 of Example 24 translates into the SQL query of Listing 4.1.*

Example 26. *Let $\mathcal{Q} = \{\mathtt{R}(\underline{x}, y, y)\}$ and let Σ be the singleton containing KJD*

$\bowtie [\{1,2\},\{1,3\}]$. *We have* $\mathcal{Q} \otimes \Sigma = \{\mathtt{R}^{\bowtie}_{\circ}(\underline{x,y,y}), \mathtt{R}^{\bowtie}_{1}(\underline{x},y), \mathtt{R}^{\bowtie}_{2}(\underline{x},y)\}$. *We have*

$$\begin{aligned}&\mathbf{rewrite}^{\Sigma}(\mathcal{Q}) =\\ &\quad \exists x\, \exists y(\mathtt{R}(\underline{x},y,y) \wedge\\ &\qquad \forall y\, \forall z(\mathtt{R}(\underline{x},y,z) \Rightarrow\\ &\qquad\quad y = z)).\end{aligned}$$

Let $\psi = \mathbf{rewrite}^{\emptyset}(\mathcal{Q} \otimes \Sigma)$, *i.e.,*

$$\begin{aligned}&\psi =\\ &\quad \exists x\, \exists y(\mathtt{R}^{\bowtie}_{1}(\underline{x},y) \wedge\\ &\qquad \forall y(\mathtt{R}^{\bowtie}_{1}(\underline{x},y) \Rightarrow\\ &\qquad\quad (\mathtt{R}^{\bowtie}_{2}(\underline{x},y) \wedge\\ &\qquad\qquad \forall z(\mathtt{R}^{\bowtie}_{2}(\underline{x},z) \Rightarrow\\ &\qquad\qquad\quad (z = y \wedge \mathtt{R}^{\bowtie}_{\circ}(\underline{x,y,y})))))).\end{aligned}$$

Let $\tilde{\psi}$ *be the following sentence:*

$$\begin{aligned}&\tilde{\psi} =\\ &\quad \exists x\, \exists y((\mathtt{R}^{\bowtie}_{1}(\underline{x},y) \wedge \mathtt{R}^{\bowtie}_{2}(\underline{x},y)) \wedge\\ &\qquad \forall y\, \forall z((\mathtt{R}^{\bowtie}_{1}(\underline{x},y) \wedge \mathtt{R}^{\bowtie}_{2}(\underline{x},z)) \Rightarrow\\ &\qquad\quad y = z)).\end{aligned}$$

ψ *and* $\tilde{\psi}$ *both express that there exist facts* $\mathtt{R}^{\bowtie}_{1}(\underline{\mathsf{a}},\mathsf{b}), \mathtt{R}^{\bowtie}_{2}(\underline{\mathsf{a}},\mathsf{b})$ *for which there exist no key-equal distinct facts.*

For every uncertain database $\mathbf{db}$ *that satisfies* Σ, *we have that the following are equivalent:*

1. $\mathbf{db} \models \mathbf{rewrite}^{\Sigma}(\mathcal{Q})$;
2. $\mathbf{db} \otimes \Sigma \models \psi$; *and*
3. $\mathbf{db} \otimes \Sigma \models \tilde{\psi}$.

This is no longer true for uncertain databases that violate Σ. *For example, let* $\mathbf{db}_0 = \{\mathtt{R}(\underline{\mathsf{a}},\mathsf{b},\mathsf{b}), \mathtt{R}(\underline{\mathsf{a}},\mathsf{c},\mathsf{c})\}$. *We have* $\mathbf{db}_0 \otimes \Sigma = \{\mathtt{R}^{\bowtie}_{\circ}(\underline{\mathsf{a},\mathsf{b},\mathsf{b}}), \mathtt{R}^{\bowtie}_{1}(\underline{\mathsf{a}},\mathsf{b}), \mathtt{R}^{\bowtie}_{2}(\underline{\mathsf{a}},\mathsf{b}), \mathtt{R}^{\bowtie}_{\circ}(\underline{\mathsf{a},\mathsf{c},\mathsf{c}}), \mathtt{R}^{\bowtie}_{1}(\underline{\mathsf{a}},\mathsf{c}), \mathtt{R}^{\bowtie}_{2}(\underline{\mathsf{a}},\mathsf{c})\}$. *Then,* $\mathbf{db}_0 \models \mathbf{rewrite}^{\Sigma}(\mathcal{Q})$, *but* $\mathbf{db}_0 \otimes \Sigma \not\models \psi$ *and* $\mathbf{db}_0 \otimes \Sigma \not\models \tilde{\psi}$.

4.5 Conclusion

The problem of consistent query answering under primary keys, also known as **CERTAINTY**$(\mathcal{Q})$, has attracted much research attention in recent years. This problem takes as its input an uncertain database **db** and asks whether the Boolean query $\mathcal{Q}$ evaluates to true on every repair of **db**. In practical situations, however, one may know that input databases satisfy some set Σ of constraints, i.e., that the input databases are partially consistent. The problem **CERTAINTY**$(\mathcal{Q}, \Sigma)$ takes as its input an uncertain database **db** *that satisfies* Σ and asks whether the query $\mathcal{Q}$ evaluates to true on every repair of **db**. The knowledge that some constraints be satisfied brings a new flavor of practical interest to consistent query answering.

We studied **CERTAINTY**$(\mathcal{Q}, \Sigma)$ in case $\mathcal{Q}$ is a Boolean query in **SJFCQ** and Σ is a set of FDs and KJDs, containing at most one KJD per relation name. The main result is that it is decidable whether **CERTAINTY**$(\mathcal{Q}, \Sigma)$ is first-order expressible. This allows to solve **CERTAINTY**$(\mathcal{Q}, \Sigma)$ by means of standard SQL database technology.

Chapter 5

Under-Approximations of Consistent Query Answers

Consistent Query Answering (CQA) is a principled approach for answering queries on inconsistent databases. The consistent answer to a query $\mathcal{Q}$ on an inconsistent database **db** is the intersection of the answers to $\mathcal{Q}$ on all repairs, where a repair is any consistent database that is maximally close to **db**. Recall from Chapter 1 that we write $\lfloor \mathcal{Q} \rfloor$ for the query that maps every database to the consistent answer to $\mathcal{Q}$. Unfortunately, there exist simple conjunctive queries $\mathcal{Q}$ and primary key constraints such that $\lfloor \mathcal{Q} \rfloor$ has exponential data complexity—which is completely impracticable—and cannot be expressed in some standard database language (like SQL or Datalog).

In this chapter, we propose a new framework for divulging an inconsistent database to end users, which adopts two postulates. The first postulate complies with CQA and states that inconsistencies should never be divulged to end users. Therefore, end users should only get consistent query answers. The second postulate states that only those queries whose consistent answers can be obtained with low data complexity (i.e., by a polynomial-time algorithm or even a first-order logic query) can be answered. User queries that exhibit a higher data complexity will be rejected.

A significant problem in this framework is as follows: given a rejected

M	<u>N</u>	A	C
	Ed	48	Mons
	Ed	48	Paris
	Dirk	29	Mons

F	<u>N</u>	A	C
	An	37	Mons
	Iris	37	Paris

Figure 5.1: Example database with primary key violations.

query, find other queries, called under-approximations, that are accepted and whose consistent answers are contained in those of the rejected query. We provide solutions to this problem for the special case where the constraints are primary keys and the queries are self-join-free conjunctive queries.

The results of this chapters have been published in [GPW15, GPW17].

5.1 Introduction

This chapter adopts the notion of uncertain database defined in Chapter 3. In particular, we assume that the only constraints are primary keys, one per relation. A repair of an inconsistent database **db** is a maximal subset of **db** that satisfies all primary key constraints. Our running example given in Figure 5.1 stores ages and cities of residence of male and female persons. For simplicity, assume that persons have unique names (attribute **N**). Every person has exactly one age (attribute **A**) and city (attribute **C**). In the database of Figure 5.1, there is uncertainty about the city of Ed (it can be Mons or Paris). The database can be repaired in two ways: delete either M(<u>Ed</u>, 48, Mons) or M(<u>Ed</u>, 48, Paris). A maximal set of tuples that agree on their primary key will be called a *block*; in Figure 5.1, blocks are separated by dashed lines.

A practical obstacle to CQA is that the shift to certainty semantics involves a significant increase in (data) complexity. It is known for long [Mar02] that there exist conjunctive queries $\mathcal{Q}$ that join two relations such that the data complexity of $\lfloor \mathcal{Q} \rfloor$ is already **coNP**-hard. If this happens, CQA is completely

impracticable.

This chapter investigates ways to circumvent the high data complexity of CQA in a realistic setting, which is based on the following assumptions:

- If a query returns an answer to a user, then every tuple in that answer should belong to the consistent answer. In Libkin's terminology [Lib15], query answers must not contain *false positives*, i.e., tuples that do not belong to the consistent answer.

- The only queries that can be executed in practice are those with data complexity in **FP** or, even better, in **FO**. Here, **FO** refers to the descriptive complexity class that captures all queries expressible in relational calculus [Imm99]. **FP** is the class of function problems solvable in polynomial time.

Therefore, if the data complexity of a query $\lfloor \mathcal{Q} \rfloor$ is not in **FP**, then the best we can go for is an approximation without false positives (also called under-approximation), computable in polynomial time. The term *strategy* will be used for queries that compute such approximations. Intuitively, a strategy can be regarded as a two-step process in which one starts by issuing a number of well-behaved queries $\lfloor \mathcal{Q}_i \rfloor$, for $i \in \{1, 2, \ldots, l\}$, which can then be subject to a post-processing step. In this chapter, well-behaved queries are those that are accepted by a query interface, e.g., self-join-free conjunctive queries $\mathcal{Q}_i$ such that $\lfloor \mathcal{Q}_i \rfloor$ is in **FO**, and post-processing is formalized as queries built-up from the $\lfloor \mathcal{Q}_i \rfloor$'s.

We next illustrate our setting by an example. Consider the following scenario with two persons, called Bob and Alice. The person called Bob owns a database that is publicly accessible only via a query interface which restricts the syntax of the queries that can be asked. Our main results concern the case where the interface is restricted to self-join-free conjunctive queries. The database schema including all primary key constraints is publicly available. However, Bob is aware that his database contains many mistakes which should not be divulged. Therefore, whenever some end user asks a query $\mathcal{Q}$,

Bob will actually execute the query $\lfloor \mathcal{Q} \rfloor$. That is, end users will get exclusively consistent answers. But, for feasibility reasons, Bob will reject any query $\mathcal{Q}$ for which the data complexity of $\lfloor \mathcal{Q} \rfloor$ is too high. In this chapter, we assume that Bob considers that data complexity is too high when it is not in **FO**. The person called Alice interrogates Bob's database, and she will be happy to get exclusively consistent answers. Unfortunately, her query $\mathcal{Q}$ will be rejected by Bob if the data complexity of $\lfloor \mathcal{Q} \rfloor$ is too high (i.e., not in **FO**). If this happens, Alice has to change strategy. Instead of asking $\mathcal{Q}$, she can ask a finite number of queries $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_l$ such that for every $i \in \{1, 2, \ldots, l\}$, the data complexity of $\lfloor \mathcal{Q}_i \rfloor$ is in **FO**, and hence the query $\mathcal{Q}_i$ will be accepted by Bob. No restriction is imposed on the number l of queries that can be asked. The best Alice can hope for is that she can compute herself the answer to $\lfloor \mathcal{Q} \rfloor$ (or even to $\mathcal{Q}$) from Bob's answers to $\lfloor \mathcal{Q}_1 \rfloor$, ..., $\lfloor \mathcal{Q}_l \rfloor$ by means of some post-processing. The question addressed in this chapter is: Given that Alice wants to answer $\mathcal{Q}$, what queries should she ask to Bob?

Here is a concrete example. Assume Bob owns the database of Figure 5.1. Interested in stable couples[1], Alice submits the query $\mathcal{Q}_1$ which asks "Get pairs of ages of men and women living in the same city":

$$\mathcal{Q}_1 = \left\{ (y, w) \mid \exists x \, \exists u \, \exists z \left(\mathtt{M}(\underline{x}, y, z) \wedge \mathtt{F}(\underline{u}, w, z) \right) \right\}.$$

The consistent answer is $\{(\mathsf{48}, \mathsf{37}), (\mathsf{29}, \mathsf{37})\}$. However, the query $\lfloor \mathcal{Q}_1 \rfloor$ that returns the consistent answer is known to have **coNP**-hard data complexity [KW17]. Therefore, Bob will reject $\mathcal{Q}_1$. Alice changes strategy and submits the query $\mathcal{Q}_2$ which asks "Get pairs of ages and city of men and women living in the same city":

$$\mathcal{Q}_2 = \left\{ (y, w, z) \mid \exists x \, \exists u \left(\mathtt{M}(\underline{x}, y, z) \wedge \mathtt{F}(\underline{u}, w, z) \right) \right\}. \tag{5.1}$$

Since the data complexity of $\lfloor \mathcal{Q}_2 \rfloor$ is known to be in **FO** [KW17], Bob will execute $\lfloor \mathcal{Q}_2 \rfloor$. The query $\mathcal{Q}_2$ returns $\{(\mathsf{29}, \mathsf{37}, \mathsf{Mons}), (\mathsf{48}, \mathsf{37}, \mathsf{Mons})\}$ on one repair, and $\{(\mathsf{29}, \mathsf{37}, \mathsf{Mons}), (\mathsf{48}, \mathsf{37}, \mathsf{Paris})\}$ on the other repair, so the consistent

[1] According to [CFG+10], marital stability is higher when the wife is 5+ years younger than her husband.

answer is $\{(\mathsf{29}, \mathsf{37}, \mathsf{Mons})\}$. This in turn allows Alice to derive a consistent answer to the original query: since $(\mathsf{29}, \mathsf{37}, \mathsf{Mons})$ belongs to the answer to $\lfloor \mathcal{Q}_2 \rfloor$, it is correct to conclude that $(\mathsf{29}, \mathsf{37})$ belongs to the answer to $\lfloor \mathcal{Q}_1 \rfloor$. An interesting question is whether Alice has a better strategy that divulges even more answers to $\lfloor \mathcal{Q}_1 \rfloor$.

Our work can also be regarded as querying "consistent views," in the sense that Bob returns exclusively consistent answers. It has been observed long ago [Wij04] that consistent views are not closed under relational calculus. In other words, the position of the $\lfloor \cdot \rfloor$ construct in a query does matter. For example, for the database of Figure 5.1, the query $\{x \mid \exists y \exists z(\lfloor \mathtt{M}(\underline{x}, y, z) \rfloor)\}$ returns only (Dirk), while $\lfloor \{x \mid \exists y \exists z(\mathtt{M}(\underline{x}, y, z))\} \rfloor$ returns both (Ed) and (Dirk). Bertossi and Li [BL13] have used views to protect the secrecy of data in a database. In our setting, the query answers that are to be hidden from end users are those that are not true in every repair.

The technical contributions of this chapter are as follows. We first show that the following problem is undecidable: Given a relational calculus query $\mathcal{Q}$, is $\lfloor \mathcal{Q} \rfloor$ in **FO**? In view of this undecidability result, we then limit our attention to strategies that are first-order combinations (using disjunction and existential quantification) of queries $\lfloor \mathcal{Q} \rfloor$ that are known to be in **FO**. We show how to build optimal strategies under such syntax restrictions.

This chapter is organized as follows. Section 5.2 provides some additional mathematical definitions. Section 5.3 introduces our new framework for studying consistent query answering under primary key constraints, and introduces the problem **OPTSTRATEGY**. Intuitively, **OPTSTRATEGY** asks, given a query $\mathcal{Q}$, to find a new query $\mathcal{Q}'$ that gets the largest subset of consistent answers while still obeying the restrictions imposed by our framework. Section 5.4 provides ways to solve **OPTSTRATEGY** in restricted settings. Section 5.5 studies a novel query containment problem that is intimately related to the simplification of strategies. Finally, Section 5.6 concludes the chapter.

5.2 Preliminaries

Given two m-ary queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$, we say that $\mathcal{Q}_1$ is *contained in* $\mathcal{Q}_2$, denoted by $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ if for every database **db**, $\mathcal{Q}_1(\mathbf{db}) \subseteq \mathcal{Q}_2(\mathbf{db})$. We write $\mathcal{Q}_1 \sqsubset \mathcal{Q}_2$, if $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ and $\mathcal{Q}_2 \not\sqsubseteq \mathcal{Q}_1$. We say that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are *equivalent*, denoted by $\mathcal{Q}_1 \equiv \mathcal{Q}_2$, if $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ and $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$.

A 0-ary query is called Boolean. If $\mathcal{Q}$ is a Boolean query, then $\mathcal{Q}$ maps any database to either $\{()\}$ or $\{\}$, corresponding to **true** and **false** respectively.

Recall that a *conjunctive query* is a relational calculus query of the form $\{\vec{z} \mid \exists \vec{y}\,(B)\}$ where B is a conjunction of atoms. By a slight abuse of notation, we denote by B also the set of conjuncts that occur in B. For example, if $B_1 = \mathtt{R}(\vec{x}) \land \mathtt{R}(\vec{y})$ and $B_2 = \mathtt{R}(\vec{x}) \land \mathtt{R}(\vec{y}) \land \mathtt{R}(\vec{z})$, then we may write $B_1 \subseteq B_2$. Finally, if $\mathcal{Q}$ is a self-join-free conjunctive query with an $\mathtt{R}$-atom, then, by an abuse of notation, we write $\mathtt{R}$ to mean the $\mathtt{R}$-atom of $\mathcal{Q}$.

5.3 A Framework for Divulging Inconsistent Databases

In this section, we formalize the setting that was described and illustrated in Section 5.1. The setting is captured by the language called **CQAFO**, which consists of first-order quantification and Boolean combinations of atomic formulas of the form $\lfloor \mathcal{Q} \rfloor$, where $\mathcal{Q}$ is any relational calculus query. The atomic formulas $\lfloor \mathcal{Q} \rfloor$ capture that the database owner Bob only returns consistent answers. Subsequently, the end user Alice, who interrogates Bob's database, can do some post-processing on Bob's outputs. In our setting, we assume that Alice uses first-order quantification and Boolean combinations of Bob's consistent answers to the atomic formulas $\lfloor \mathcal{Q} \rfloor$.

Example 27. *The scenario in Section 5.1 is captured by the* ***CQAFO*** *query*

$$\left\{(y, w) \mid \exists Z\left(\left\lfloor \exists x \exists u \left(\mathtt{M}(\underline{x}, y, Z) \land \mathtt{F}(\underline{u}, w, Z)\right)\right\rfloor\right)\right\}.$$

The formula within $\lfloor\cdot\rfloor$ is the query (5.1). The quantification $\exists Z$ corresponds to Alice *projecting away the cities column returned by* Bob*. For readability, we*

will often use upper case letters for variables that are quantified outside the range of $\lfloor\cdot\rfloor$.

Example 28. *The following query allows* Alice *to find the names of men with at least two cities in the database:*

$$\left\{x \mid \left\lfloor \exists y \, \exists z \left(\mathtt{M}(\underline{x}, y, z)\right)\right\rfloor \wedge \neg \exists Z \left(\left\lfloor \exists y \left(\mathtt{M}(\underline{x}, y, Z)\right)\right\rfloor\right)\right\}.$$

To understand this query, it may be helpful to notice that

$$\left\{(x, Z) \mid \left\lfloor \exists y \, \mathtt{M}(\underline{x}, y, Z)\right\rfloor\right\}$$

returns tuple (n, c) *whenever* c *is the only city of residence recorded for the person named* n*. Interestingly, even though* Alice *gets only consistent answers, she can still infer that the database contains inconsistencies. In particular, since the foregoing query returns* Ed *on the example database of Figure 5.1,* Alice *can infer that there is uncertainty about the city of* Ed*. This example shows, maybe somewhat unexpectedly, that end users who only get consistent query answers may still be able to infer that the database is inconsistent.*

5.3.1 The Language CQAFO

We next describe the syntax and semantics of the language **CQAFO** used for postprocessing.

Syntax The following are formulas in **CQAFO**:

- if $\mathcal{Q}$ is a relational calculus query, then $\lfloor\mathcal{Q}\rfloor$ is a **CQAFO** formula with the same free variables as $\mathcal{Q}$;
- if φ_1 and φ_2 are **CQAFO** formulas, then $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, and $\neg\varphi_1$ are **CQAFO** formulas;
- if φ is a **CQAFO** formula, then $\exists Y\,(\varphi)$ and $\forall Y\,(\varphi)$ are **CQAFO** formulas.

If φ is a **CQAFO** formula, then $\mathbf{free}(\varphi)$ denotes the set of free variables of φ (i.e., the variables not bound by a quantifier). If $\vec{x}$ is a tuple containing the free variables of φ, we write $\varphi(\vec{x})$.

A **CQAFO** query is an expression of the form $\{\vec{u} \mid \varphi\}$, where $\vec{u}$ is a sequence of symbols containing each variable of $\mathbf{free}(\varphi)$. If $\vec{n}$ contains no constants and no double occurrences of the same variable, then such query is also denoted $\varphi(\vec{t})$.

Semantics Let $\mathbf{db}$ be an uncertain database. Let $\varphi(\vec{x})$ be a **CQAFO** formula, and $\vec{a}$ be a sequence of constants (of the same length as $\vec{x}$). We inductively define $\mathbf{db} \models \varphi(\vec{a})$.

- If $\varphi(\vec{x}) = \lfloor \mathcal{Q}(\vec{x}) \rfloor$ for some relational calculus query $\mathcal{Q}(\vec{x})$, then $\mathbf{db} \models \varphi(\vec{a})$ if for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathbf{r} \models \mathcal{Q}(\vec{a})$; [2]
- $\mathbf{db} \models \neg\varphi(\vec{a})$ if $\mathbf{db} \not\models \varphi(\vec{a})$;
- $\mathbf{db} \models \varphi_1 \wedge \varphi_2$ if $\mathbf{db} \models \varphi_1$ and $\mathbf{db} \models \varphi_2$;
- $\mathbf{db} \models \varphi_1 \vee \varphi_2$ if $\mathbf{db} \models \varphi_1$ or $\mathbf{db} \models \varphi_2$;
- if $\psi(\vec{x}) = \exists Y\, (\varphi(Y, \vec{x}))$, then $\mathbf{db} \models \psi(\vec{a})$ if $\mathbf{db} \models \varphi(\mathsf{a}', \vec{a})$ for some a';
- if $\psi(\vec{x}) = \forall Y\, (\varphi(Y, \vec{x}))$, then $\mathbf{db} \models \psi(\vec{a})$ if $\mathbf{db} \models \varphi(\mathsf{a}', \vec{a})$ for all a'.

Let $q = \{\vec{n} \mid \varphi(\vec{x})\}$ be a **CQAFO** query. The answer $q(\mathbf{db})$ is the smallest set containing $\theta(\vec{n})$ for every valuation θ over $\mathbf{vars}(\vec{n})$ such that for some $\vec{a}$, $\theta(\vec{x}) = \vec{a}$ and $\mathbf{db} \models \varphi(\vec{a})$. By definition, we have $\mathbf{vars}(\vec{n}) = \mathbf{vars}(\vec{x})$, but $\vec{n}$, unlike $\vec{x}$, can contain constants and multiple occurrences of the same variable. If $\vec{t}$ contains no variables, then q is Boolean.

Domain independence is a desirable property of queries that emerges in **CQAFO** in the same way as in relational calculus [AHV95, p. 79]. For example, consider the **CQAFO** query

$$q_0 = \left\{x \mid \left\lfloor \exists y\, \exists z \left(\mathtt{M}(\underline{x}, y, z)\right)\right\rfloor \vee \left\lfloor \mathtt{F}(\underline{\mathsf{Iris}}, \mathsf{37}, \mathsf{Paris})\right\rfloor\right\}$$

[2] $\mathbf{r} \models \mathcal{Q}(\vec{a})$ is defined in the standard way.

on the example database of Figure 5.1. Since $\mathtt{F}(\underline{\mathsf{Iris}}, 37, \mathsf{Paris})$ holds true in every repair, the query is true for any value of x. The query q_0 is thus not domain independent. Nevertheless, domain independence will not be an issue in this chapter, because we will only deal with syntactic fragments of **CQAFO** that guarantee domain independence.

5.3.2 Restrictions on Data Complexity

The language **CQAFO** of Section 5.3.1 captures our first postulate which states that the database owner Bob returns exclusively consistent answers. But we do not prohibit that end user Alice does some post-processing on Bob's answers. In this section, we will add our second postulate which states that Bob rejects queries $\mathcal{Q}$ if the data complexity of $\lfloor \mathcal{Q} \rfloor$ is not in **FO**. Unfortunately, Bob has to face the following undecidability result.

Theorem 7. *The following problem is undecidable. Given a relational calculus query $\mathcal{Q}$, is $\lfloor \mathcal{Q} \rfloor$ in* **FO***?*

Proof. Let $\mathcal{Q}_1 = \{() \mid \exists x \, \exists y \, \exists z (\mathtt{R}(\underline{x}, z) \wedge \mathtt{S}(\underline{y}, z) \wedge \varphi)\}$ where φ is a closed relational calculus formula, i.e., a formula with no free variables, such that all relation names in φ are all-key. Observe that this implies that the relation names in φ are distinct from R and S. We show hereinafter that $\lfloor \mathcal{Q} \rfloor$ is in **FO** if and only if φ is unsatisfiable. The desired result then follows by [AHV95, Theorem 6.3.1], which states that (finite) satisfiability of relational calculus queries is undecidable.

Obviously, if φ is unsatisfiable, then $\lfloor \mathcal{Q}_1 \rfloor \equiv$ **false**, and hence $\lfloor \mathcal{Q}_1 \rfloor$ is in **FO**.

We show next that if φ is satisfiable, then $\lfloor \mathcal{Q}_1 \rfloor$ is not in **FO**. Assume that φ is satisfiable. Let $\mathcal{Q}_0 = \exists x \, \exists y \, \exists z (\mathtt{R}(\underline{x}, z) \wedge \mathtt{S}(\underline{y}, z))$ and consider the following two problems:

- **CERTAIN0**: Given a database **db**, determine whether every repair of **db** satisfies $\mathcal{Q}_0$.

- **CERTAIN1**: Given a database **db**, determine whether every repair of **db** satisfies $\mathcal{Q}_1$.

We now show a polynomial-time many-one reduction between **CERTAIN0** and **CERTAIN1**. Let $\mathbf{db}_0$ be a database that is input to **CERTAIN0**. Let **S** be the database schema that contains the relation names occurring in φ. An algorithm can consider systematically every finite database $\mathbf{db}'$ over **S** and test $\mathbf{db}' \models \varphi$, until a database $\mathbf{db}'$ is found such that $\mathbf{db}' \models \varphi$. The algorithm terminates because φ is satisfiable. Since the computation of $\mathbf{db}'$ does not depend on $\mathbf{db}_0$, it takes $\mathcal{O}(1)$ time. Since all relation names in $\mathbf{db}'$ are all-key, we have that $\mathbf{db}'$ is consistent. Clearly, $\mathcal{Q}_0$ is true in every repair of $\mathbf{db}_0$ if and only if $\mathcal{Q}_1$ is true in every repair of $\mathbf{db}_0 \cup \mathbf{db}'$. This follows from the fact that the relation names in φ are distinct from R and S. So we have established a polynomial-time many-one reduction from **CERTAIN0** to **CERTAIN1**. Since **CERTAIN0** is **coNP**-hard [KW17], it follows that **CERTAIN1** is **coNP**-hard. Since **FO** $\subsetneq$ **coNP** [Imm99], it follows that **CERTAIN1** is not in **FO**. □

By Theorem 7, there exists no algorithm that allows Bob to decide whether he has to accept or reject a relational calculus query. In general, little is known about the complexity of $\lfloor \mathcal{Q} \rfloor$ for relational calculus queries $\mathcal{Q}$. Of course, Theorem 3 implies the following result.

Theorem 8 ([KW17])**.** *The following problem is decidable in polynomial time. Given a self-join-free conjunctive query $\mathcal{Q}$, is $\lfloor \mathcal{Q} \rfloor$ in* **FO***? Moreover, if $\lfloor \mathcal{Q} \rfloor$ is in* **FO***, then a relational calculus query equivalent to $\lfloor \mathcal{Q} \rfloor$ can be effectively constructed.*

In view of Theorem 7 and of Theorem 8, the following scenario is the best we can go for with the current state of art.

1. The database owner Bob only accepts self-join-free conjunctive queries $\mathcal{Q}$ such that $\lfloor \mathcal{Q} \rfloor$ is in **FO**. Thus, Bob rejects every query that is not self-join-free conjunctive, and rejects a self-join-free conjunctive query $\mathcal{Q}$ if $\lfloor \mathcal{Q} \rfloor$ is not in **FO**.

2. As before, Alice can do some first-order post-processing on the answers obtained from Bob.

Under these restrictions, we focus on the following problem: given that Alice wants to answer a self-join-free conjunctive query $\mathcal{Q}$ on a database owned by Bob, develop a *strategy* for Alice to get a subset (the greater, the better) of the consistent answer to $\mathcal{Q}$. Our framework applies to Boolean queries by representing **true** and **false** by $\{()\}$ and $\{\}$ respectively. A formal definition follows.

5.3.3 Strategies

Strategies for a query $\mathcal{Q}$ are defined next as relational calculus queries that can be expressed in **CQAFO** and that are contained in $\lfloor\mathcal{Q}\rfloor$.

Definition 8. *Let $\mathcal{Q}$ be a self-join-free conjunctive query. A* strategy for $\mathcal{Q}$ *is a **CQAFO** query φ such that $\varphi \sqsubseteq \lfloor\mathcal{Q}\rfloor$ and for every atomic formula $\lfloor\mathcal{Q}'\rfloor$ in φ, we have that $\mathcal{Q}'$ is a self-join-free conjunctive query such that $\lfloor\mathcal{Q}'\rfloor$ is in* **FO**.

A strategy φ for $\mathcal{Q}$ is optimal *if for every strategy ψ for $\mathcal{Q}$, we have $\psi \sqsubseteq \varphi$. The problem* **OPTSTRATEGY** *takes in a self-join-free conjunctive query $\mathcal{Q}$ and asks to determine an optimal strategy for $\mathcal{Q}$.*

Some observations are in place.

- If the input to **OPTSTRATEGY** is a self-join-free conjunctive $\mathcal{Q}$ such that $\lfloor\mathcal{Q}\rfloor$ is in **FO**, then the **CQAFO** query $\lfloor\mathcal{Q}\rfloor$ is itself an optimal strategy.

- Every strategy φ is in **FO**, because all atomic formulas $\lfloor\mathcal{Q}'\rfloor$ are required to be in **FO**. Therefore, if Alice wants to answer a query $\mathcal{Q}$ such that $\lfloor\mathcal{Q}\rfloor$ is not in **FO**, then there is no strategy φ such that $\varphi \equiv \lfloor\mathcal{Q}\rfloor$.

- We require that the input query to **OPTSTRATEGY** belongs to the class of self-join-free conjunctive queries. The reason for this requirement

is that this is the largest class of queries for which the existence (or not) of consistent first-order rewritings is known to be decidable.

We will not investigate the problem **OPTSTRATEGY** in its most general form in the remainder of this chapter. Instead, we will confine our investigation to strategies that can be expressed and effectively constructed in a syntactic fragment of **CQAFO**. We will explain how such strategies can be constructed, but leave open the computational complexity of the construction.

5.4 How to Construct Good Strategies?

Let $\mathcal{Q}$ be a self-join-free conjunctive query. In this section, we investigate ways for constructing good (if not optimal) strategies for $\mathcal{Q}$ of a particular syntax. In Section 5.4.1, we take the most simple approach: take the union of queries $\lfloor \mathcal{Q}_i \rfloor$ contained in $\lfloor \mathcal{Q} \rfloor$, where $\mathcal{Q}_i$ is self-join-free conjunctive and $\lfloor \mathcal{Q}_i \rfloor$ is in **FO**. We then show that the strategies obtained in this way cannot be optimal. Therefore, an enhanced approach is developed in Section 5.4.2.

5.4.1 Post-Processing by Unions Only

Assume that the input to **OPTSTRATEGY** is a self-join-free conjunctive query $\mathcal{Q}(\vec{z})$. In this section, we look at strategies of the form

$$\bigcup_{i=1}^{l} \lfloor \mathcal{Q}_i \rfloor , \tag{5.2}$$

where each $\mathcal{Q}_i$ is of the form $\{\vec{z_i} \mid \exists \vec{y_i}\, (B_i)\}$ in which $\vec{z_i}$ has the same length as $\vec{z}$ and B_i is a self-join-free conjunction of atoms. Speaking strictly syntactically, $\lfloor \{\vec{z_i} \mid \exists \vec{y_i}\, (B_i)\} \rfloor$ is not a **CQAFO** query, as it is not of the form $\{\vec{u} \mid \varphi\}$ for some **CQAFO** formula φ as defined in Section 5.3.1. However, it can be easily verified that $\lfloor \{\vec{z_i} \mid \exists \vec{y_i}\, (B_i)\} \rfloor \equiv \{\vec{z_i} \mid \lfloor \exists \vec{y_i}\, (B_i) \rfloor\}$, and the latter query is a **CQAFO** query.

We use union (with its standard semantics) instead of disjunction to avoid

notational difficulties. For example, the union

$$\left\{(x, \mathtt{a}) \mid \left\lfloor \mathtt{R}(\underline{x}, \mathtt{a})\right\rfloor\right\} \cup \left\{(x, y) \mid \left\lfloor \mathtt{S}(\underline{x}, y)\right\rfloor\right\},$$

where $\mathtt{a}$ is a constant, is semantically clear, and is equivalent to

$$\left\{(x, y) \mid \left\lfloor \mathtt{R}(\underline{x}, y) \wedge y = \mathtt{a}\right\rfloor \vee \left\lfloor \mathtt{S}(\underline{x}, y)\right\rfloor\right\},$$

in which equality is used. It would be wrong to write $\{(x, y) \mid \lfloor \mathtt{R}(\underline{x}, \mathtt{a})\rfloor \vee \lfloor \mathtt{S}(\underline{x}, y)\rfloor\}$, an expression that is not domain independent [AHV95, p. 79], because if some fact $\mathtt{R}(\underline{\mathtt{c}}, \mathtt{a})$ holds true in every repair, then $\lfloor \mathtt{R}(\underline{x}, \mathtt{a})\rfloor \vee \lfloor \mathtt{S}(\underline{x}, y)\rfloor$ is true when $\mathtt{c}$ is assigned to x, no matter what value is assigned to y. On the other hand, a **CQAFO** formula of the form (5.2) is domain independent if each $\lfloor \mathcal{Q}_i \rfloor$ is domain independent.

Furthermore, a formula of the form (5.2) is a strategy if for every $i \in \{1, 2, \ldots, l\}$, $\lfloor \mathcal{Q}_i \rfloor$ is in **FO** and $\lfloor \mathcal{Q}_i \rfloor \sqsubseteq \lfloor \mathcal{Q} \rfloor$. The latter condition is equivalent to $\mathcal{Q}_i \sqsubseteq \mathcal{Q}$ as is shown next.

Lemma 5. *Let $\mathcal{Q}$ and $\mathcal{Q}'$ be self-join-free m-ary conjunctive queries. Then, $\mathcal{Q} \sqsubseteq \mathcal{Q}'$ if and only if $\lfloor \mathcal{Q} \rfloor \sqsubseteq \lfloor \mathcal{Q}' \rfloor$.*

Proof. Let $\mathcal{Q} = \{\vec{z} \mid \exists \vec{y}\,(B)\}$ and $\mathcal{Q}' = \{\vec{z}_0 \mid \exists \vec{y}_0\,(B')\}$, where $\vec{z}$ and $\vec{z}_0$ both have the same length m.

$\boxed{\Longrightarrow}$ Straightforward. $\boxed{\Longleftarrow}$ Assume $\lfloor \mathcal{Q} \rfloor \sqsubseteq \lfloor \mathcal{Q}' \rfloor$. Let μ be an injective mapping with domain $\mathbf{vars}(B)$ that maps each variable to a fresh constant not occurring elsewhere. Since μ is injective, its inverse μ^{-1} is well defined. Let $\mathbf{db} = \mu(B)$. Clearly, $\mathbf{db}$ is consistent and $\mathcal{Q}(\mathbf{db}) = \{\mu(\vec{z})\} = \lfloor \mathcal{Q} \rfloor(\mathbf{db})$. From $\lfloor \mathcal{Q} \rfloor \sqsubseteq \lfloor \mathcal{Q}' \rfloor$, it follows $\mu(\vec{z}) \in \lfloor \mathcal{Q}' \rfloor(\mathbf{db}) = \mathcal{Q}'(\mathbf{db})$. Then, there exists a valuation θ over $\mathbf{vars}(B')$ such that $\theta(B') \subseteq \mathbf{db}$ and $\theta(\vec{z}_0) = \mu(\vec{z})$. Then $\mu^{-1} \circ \theta(B') \subseteq B$ and $\mu^{-1} \circ \theta(\vec{z}_0) = \vec{z}$. Since $\mu^{-1} \circ \theta$ is a homomorphism from $\mathcal{Q}'$ to $\mathcal{Q}$, it follows $\mathcal{Q} \sqsubseteq \mathcal{Q}'$ by the Homomorphism Theorem [AHV95, Theorem 6.2.3]. □

Lemma 5 does not hold for conjunctive queries with self-joins, as shown next.

Example 29. *Let* $\mathcal{Q} = \{() \mid \mathtt{R}(\underline{\mathtt{a}}, \mathtt{b}) \wedge \mathtt{R}(\underline{\mathtt{a}}, \mathtt{c})\}$*. For every uncertain database* $\mathbf{db}$*, we have* $\lfloor \mathcal{Q} \rfloor (\mathbf{db}) = \{\}$*. Let* $\mathcal{Q}'$ *be a query such that* $\mathcal{Q} \not\sqsubseteq \mathcal{Q}'$ *(such query obviously exists). Then,* $\lfloor \mathcal{Q} \rfloor \sqsubseteq \lfloor \mathcal{Q}' \rfloor$ *and* $\mathcal{Q} \not\sqsubseteq \mathcal{Q}'$*.*

Lemma 5 allows us to construct strategies of the form (5.2), as follows. Assume that the input to **OPTSTRATEGY** is a self-join-free conjunctive query $\mathcal{Q}(\vec{z})$. For some positive integer l, generate self-join-free conjunctive queries $\mathcal{Q}_1, \ldots, \mathcal{Q}_l$ such that for each $i \in \{1, 2, \ldots, l\}$, $\mathcal{Q}_i \sqsubseteq \mathcal{Q}$ and $\lfloor \mathcal{Q}_i \rfloor$ is in **FO**. The condition $\mathcal{Q}_i \sqsubseteq \mathcal{Q}$ is decidable by [AHV95, Theorem 6.2.3]; the condition that $\lfloor \mathcal{Q}_i \rfloor$ is in **FO** is decidable by Theorem 8. Then by Lemma 5, $\bigcup_{i=1}^{l} \lfloor \mathcal{Q}_i \rfloor$ is a strategy for $\mathcal{Q}$.

Unfortunately, Theorem 9 given hereinafter states that there are cases where no strategy of the form (5.2) is optimal. We first generalize Lemma 5 to unions.

Lemma 6. *Let* $\mathcal{Q}_0$*,* $\mathcal{Q}_1$*,* $\ldots \mathcal{Q}_l$ *be self-join-free* m*-ary conjunctive queries. Then,* $\lfloor \mathcal{Q}_0 \rfloor \sqsubseteq \bigcup_{i=1}^{l} \lfloor \mathcal{Q}_i \rfloor$ *if and only if for some* $i \in \{1, 2, \ldots, l\}$*,* $\mathcal{Q}_0 \sqsubseteq \mathcal{Q}_i$*.*

Proof. $\boxed{\Longleftarrow}$ Straightforward. $\boxed{\Longrightarrow}$ Assume $\lfloor \mathcal{Q}_0 \rfloor \sqsubseteq \bigcup_{i=1}^{l} \lfloor \mathcal{Q}_i \rfloor$. Let $\mathcal{Q}_0 = \{\vec{z}_0 \mid \exists \vec{y}_0 \, (B_0)\}$, where B_0 is self-join-free. Let μ be an injective mapping with domain $\mathbf{vars}(B_0)$ that maps each variable to a fresh constant not occurring elsewhere. Since μ is injective, its inverse μ^{-1} is well defined. Let $\mathbf{db} = \mu(B_0)$. Clearly, $\mathbf{db}$ is consistent and $\mathcal{Q}_0(\mathbf{db}) = \{\mu(\vec{z}_0)\} = \lfloor \mathcal{Q}_0 \rfloor (\mathbf{db})$. From $\lfloor \mathcal{Q}_0 \rfloor \sqsubseteq \bigcup_{i=1}^{l} \lfloor \mathcal{Q}_i \rfloor$, it follows that we can assume $i \in \{1, 2, \ldots, l\}$ such that $\mu(\vec{z}_0) \in \lfloor \mathcal{Q}_i \rfloor (\mathbf{db}) = \mathcal{Q}_i(\mathbf{db})$. Let $\mathcal{Q}_i = \{\vec{z}_i \mid \exists \vec{y}_i \, (B_i)\}$. Then, there exists a valuation θ over $\mathbf{vars}(B_i)$ such that $\theta(B_i) \subseteq \mathbf{db}$ and $\theta(\vec{z}_i) = \mu(\vec{z}_0)$. Then $\mu^{-1} \circ \theta(B_i) \subseteq B_0$ and $\mu^{-1} \circ \theta(\vec{z}_i) = \vec{z}_0$. Since $\mu^{-1} \circ \theta$ is a homomorphism from $\mathcal{Q}_i$ to $\mathcal{Q}_0$, it follows $\mathcal{Q}_0 \sqsubseteq \mathcal{Q}_i$. $\square$

Theorem 9. *There exists a self-join-free conjunctive query* $\mathcal{Q}$ *such that for every strategy* φ *of the form (5.2) for* $\mathcal{Q}$*, there exists another strategy* ψ *of the form (5.2) for* $\mathcal{Q}$ *such that* $\varphi \sqsubset \psi$*.*

Proof. Let $\mathcal{Q} = \{() \mid \exists x \, \exists y \, \exists z (\mathtt{R}(\underline{x}, z) \wedge \mathtt{S}(\underline{y}, z))\}$. Then $\lfloor \mathcal{Q} \rfloor$ is not in **FO** by Theorem 3 (which is subsumed by Theorem 3.2 in [KW17]). For every

constant c, let $\mathcal{Q}_{\mathsf{c}}$ be the query defined by

$$\mathcal{Q}_{\mathsf{c}} := \left\{ () \mid \exists y \, \exists z \left(\mathtt{R}(\underline{\mathsf{c}}, z) \wedge \mathtt{S}(\underline{y}, z) \right) \right\}.$$

For every constant c, we have that $\lfloor \mathcal{Q}_{\mathsf{c}} \rfloor \sqsubseteq \lfloor \mathcal{Q} \rfloor$ by Lemma 5, and again by Theorem 3, $\lfloor \mathcal{Q}_{\mathsf{c}} \rfloor$ is in **FO**.

Let φ be a strategy for $\mathcal{Q}$ of the form (5.2). Let A be the greatest set of constants such that for all $\mathsf{c} \in A$, there exists some $i \in \{1, 2, \ldots, l\}$ such that $\mathcal{Q}_i \equiv \mathcal{Q}_{\mathsf{c}}$. Let b be a constant such that $\mathsf{b} \notin A$. Clearly $\varphi \sqsubseteq \varphi \cup \lfloor \mathcal{Q}_{\mathsf{b}} \rfloor \sqsubseteq \lfloor \mathcal{Q} \rfloor$. It suffices to show that $\varphi \sqsubset \varphi \cup \lfloor \mathcal{Q}_{\mathsf{b}} \rfloor$, meaning that φ is not optimal.

Assume towards a contradiction that $\lfloor \mathcal{Q}_{\mathsf{b}} \rfloor \sqsubseteq \varphi$. By Lemma 6, there exists $i \in \{1, 2, \ldots, l\}$ such that $\mathcal{Q}_{\mathsf{b}} \sqsubseteq \mathcal{Q}_i \sqsubseteq \mathcal{Q}$. We can assume (not necessarily distinct) variables s, t, u, v such that $\mathcal{Q}_i$ is the existential closure of $\mathtt{R}(\underline{s}, t) \wedge \mathtt{S}(\underline{u}, v)$. From $\mathcal{Q}_i \sqsubseteq \mathcal{Q}$, it follows that $t = v$. From $\mathcal{Q}_{\mathsf{b}} \sqsubseteq \mathcal{Q}_i$ and $\mathsf{b} \notin A$, it follows that s, t, u are pairwise distinct variables. But then $\mathcal{Q}_i \equiv \mathcal{Q}$, contradicting that $\lfloor \mathcal{Q}_i \rfloor$ is in **FO**. We conclude by contradiction that $\varphi \sqsubset \varphi \cup \lfloor \mathcal{Q}_{\mathsf{b}} \rfloor$. □

5.4.2 Post-Processing by Unions and Quantification

The proof of Theorem 9 indicates that strategies of the form (5.2) lack expressiveness because the number of constants in such strategies is bounded. An obvious extension is to look for strategies that replace constants with existentially quantified variables. The following example shows how such extension solves the lack of expressiveness that underlies the proof of Theorem 9.

Example 30. *Let* $\mathcal{Q} = \{() \mid \exists x \, \exists y \, \exists z (\mathtt{R}(\underline{x}, z) \wedge \mathtt{S}(\underline{y}, z))\}$ *and consider the* ***CQAFO*** *formula* φ *defined by* $\varphi := \exists X (\lfloor \exists y \, \exists z (\mathtt{R}(\underline{X}, z) \wedge \mathtt{S}(\underline{y}, z)) \rfloor)$*. From Lemma 7 and Theorem 3, it follows that* φ *is a strategy for* $\mathcal{Q}$*, i.e.,* $\varphi \sqsubseteq \lfloor \mathcal{Q} \rfloor$ *and* $\lfloor \exists y \, \exists z (\mathtt{R}(\underline{X}, z) \wedge \mathtt{S}(\underline{y}, z)) \rfloor$ *is in* **FO**. *Recall from Example 27 that the use of upper case* X *is for readability.*

Assume that the input to **OPTSTRATEGY** is a self-join-free conjunctive query $\mathcal{Q}(\vec{z})$. We next investigate strategies of the form

$$\bigcup_{i=1}^{l} q_i, \tag{5.3}$$

where for each $i \in \{1, 2, \ldots, l\}$, q_i is a **CQAFO** query of the form

$$\left\{\vec{z}_i \mid \exists \vec{X}_i \left(\lfloor \exists \vec{y}_i \left(B_i\right)\rfloor\right)\right\}, \tag{5.4}$$

in which $\vec{z}_i$ has the same length as $\vec{z}$, and B_i is a self-join-free conjunction of atoms. It is understood that $\vec{z}_i$, $\vec{X}_i$, and $\vec{y}_i$ have, pairwise, no variables in common, and that $\mathbf{vars}(\vec{z}_i) \cup \mathbf{vars}(\vec{X}_i) \cup \mathbf{vars}(\vec{y}_i) = \mathbf{vars}(B_i)$. For readability, we will use lower case q to refer to **CQAFO** queries of the form (5.4). The main tools for constructing strategies of the form (5.3) are provided by Theorem 10 and Theorem 11.

Theorem 10. *The following problem is decidable in polynomial time. Given a **CQAFO** query q of the form (5.4), is q in* **FO**? *Moreover, if q is in* **FO**, *then a relational calculus query equivalent to q can be effectively constructed.*

Proof. Let B be a self-join-free conjunction of atoms, and let

$$\begin{aligned} q &= \left\{\vec{z} \mid \exists \vec{X} \left(\lfloor \exists \vec{y} \left(B\right)\rfloor\right)\right\}; \\ q' &= \left\{\vec{z}\vec{X} \mid \lfloor \exists \vec{y} \left(B\right)\rfloor\right\}. \end{aligned}$$

Obviously, if q' is in **FO**, then so is q. We show next that if q' is not in **FO**, then q is not in **FO**.

For every variable x, we assume an infinite set of constants, denoted $\mathbf{type}(x)$, such that $x \neq y$ implies $\mathbf{type}(x) \cap \mathbf{type}(y) = \emptyset$. Let **db** be an uncertain database. We say that **db** is *typed relative to B* if for every atom $\mathtt{R}(x_1, \ldots, x_n)$ in B, for every $i \in \{1, 2, \ldots, n\}$, if x_i is a variable, then for every fact $\mathtt{R}(\mathtt{a}_1, \ldots, \mathtt{a}_n)$ in **db**, $\mathtt{a}_i \in \mathbf{type}(x_i)$ and the constant $\mathtt{a}_i$ does not occur in B. Significantly, since B is self-join-free, we can assume without loss of generality that q and q' are executed on databases that are typed relative to B.

From the complexity proofs in [KW17], it follows that if q' is not in **FO**, then q' is not in **FO** even if for every variable $v \in \mathbf{vars}(\vec{z}) \cup \mathbf{vars}(\vec{X})$ (i.e., for every free variable v of q'), $\mathbf{type}(v)$ is a singleton. This means that if q' is not in **FO**, it is not in **FO** even on uncertain databases **db** such that for every

atom $\mathtt{R}(x_1, \ldots, x_n)$ in B and $i \in \{1, 2, \ldots, n\}$, if $x_i \in \mathbf{vars}(\vec{z}) \cup \mathbf{vars}(\vec{X})$, then all R-facts of **db** agree on position i. It is then obvious that if q' is not in **FO**, it must be the case that q is not in **FO** (because there is only one valuation for $\mathbf{vars}(\vec{z}) \cup \mathbf{vars}(\vec{X})$ that can make $\lfloor \exists \vec{y} \, (B) \rfloor$ true).

By Theorem 8, it can be decided whether q' is in **FO**. A relational calculus query equivalent to q can be straightforwardly obtained from a relational calculus query equivalent to q'. □

We will be concerned with testing containment between **CQAFO** queries of the form (5.4). The following lemma generalizes Lemma 5 by allowing (restricted forms of) existential quantification outside $\lfloor \cdot \rfloor$.

Lemma 7. *Let B_1 and B_2 be self-join-free conjunctions of atoms in the following **CQAFO** queries:*

$$q_1 = \left\{ \vec{z}_1 \mid \exists \vec{X}_1 \left(\lfloor \exists \vec{y}_1 \, (B_1) \rfloor \right) \right\};$$
$$q_2 = \left\{ \vec{z}_2 \mid \exists \vec{X}_2 \left(\lfloor \exists \vec{y}_2 \, (B_2) \rfloor \right) \right\}.$$

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be the queries obtained from respectively q_1 and q_2 by omitting $\lfloor \cdot \rfloor$, that is,

$$\mathcal{Q}_1 = \left\{ \vec{z}_1 \mid \exists \vec{X}_1 \left(\exists \vec{y}_1 \, (B_1) \right) \right\};$$
$$\mathcal{Q}_2 = \left\{ \vec{z}_2 \mid \exists \vec{X}_2 \left(\exists \vec{y}_2 \, (B_2) \right) \right\}.$$

1. *If $q_2 \sqsubseteq q_1$, then $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$.*

2. *If X_1 is empty and $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$, then $q_2 \sqsubseteq q_1$.*

Proof. The proof of 1 is analogous to the proof of the if-direction of Lemma 5.

For 2, assume X_1 is empty and $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$. By the Homomorphism Theorem [AHV95, Theorem 6.2.3], there exists a valuation θ over $\mathbf{vars}(B_1)$ such that $\theta(\vec{z}_1) = \vec{z}_2$ and $\theta(B_1) \subseteq B_2$. Let **db** be a database and $\vec{a}$ a sequence of constants such that $\vec{a} \in q_2(\mathbf{db})$. Then, there exists a valuation γ over $\mathbf{vars}(\vec{z}_2) \cup \mathbf{vars}(\vec{X}_2)$ with $\gamma(\vec{z}_2) = \vec{a}$ such that for every repair $\mathbf{r}$ of **db**, γ can be extended into a valuation $\Gamma_{\mathbf{r}}$ over $\mathbf{vars}(B_2)$ such that $\Gamma_{\mathbf{r}}(B_2) \subseteq \mathbf{r}$. Let $\mathbf{r}_0$

be an arbitrary repair of **db**. The result $\vec{a} \in \mathcal{Q}_1(\mathbf{r}_0)$ follows because $\Gamma_{\mathbf{r}_0} \circ \theta$ is a valuation over $\mathbf{vars}(B_1)$ such that $\Gamma_{\mathbf{r}_0} \circ \theta(B_1) \subseteq \mathbf{r}_0$ and $\Gamma_{\mathbf{r}_0} \circ \theta(\vec{z}_1) = \vec{a}$. Since $\mathbf{r}_0$ be an arbitrary repair, from $\vec{a} \in \mathcal{Q}_1(\mathbf{r}_0)$ and X_1 empty, it follows $\vec{a} \in q_1(\mathbf{db})$. □

Theorem 11. *Given a self-join-free conjunctive query $\mathcal{Q}_1$ and a **CQAFO** query q_2 of the form (5.4), it can be decided whether $q_2 \sqsubseteq \lfloor \mathcal{Q}_1 \rfloor$.*

Proof. Immediate from Lemma 7 and the decidability of containment for conjunctive queries. □

We point out that Theorem 11 is interesting in its own right. It is well known [AHV95, Corollary 6.3.2] that containment of relational calculus queries is undecidable. A large fragment for which containment is decidable is the class of unions of conjunctive queries. Notice, however, that the queries in the statement of Theorem 11 need not be monotonic (and even not first-order), and that decidability of containment for such queries is not obvious. We next provide an example of such a non-monotonic query.

Example 31. *Let $q = \{x \mid \exists Y(\lfloor \mathtt{R}(\underline{x}, Y) \rfloor)\}$. Let $\mathbf{db} = \{\mathtt{R}(\underline{\mathtt{a}}, 1)\}$ and $\mathbf{db}' = \{\mathtt{R}(\underline{\mathtt{a}}, 1), \mathtt{R}(\underline{\mathtt{a}}, 2)\}$. Then $\mathbf{db} \subseteq \mathbf{db}'$, but $q(\mathbf{db}) = \{(\mathtt{a})\}$ is not contained in $q(\mathbf{db}') = \{\}$. Hence q is not monotonic. As a note aside, we observe that q is equivalent to the following relational calculus query:*

$$\left\{x \mid \exists y \left(\mathtt{R}(\underline{x}, y) \wedge \forall y' \left(\mathtt{R}(\underline{x}, y') \Rightarrow y = y'\right)\right)\right\}.$$

Assume that the input to **OPTSTRATEGY** is a self-join-free conjunctive query $\mathcal{Q}(\vec{z})$. Theorem 11 allows us to build a strategy φ of the form (5.3) for $\mathcal{Q}$ as follows. Let A be the set of constants that occur in $\mathcal{Q}$. Let φ be the disjunction of all (up to variable renaming) **CQAFO** formulas q_i of the form (5.4) that use exclusively constants from A such that $q_i \sqsubseteq \lfloor \mathcal{Q} \rfloor$ and q_i is in **FO**. Clearly, there are at most finitely many such formulas (up to variable renaming). Containment of q_i in $\lfloor \mathcal{Q} \rfloor$ is decidable by Theorem 11. Finally, the condition that q_i is in **FO** is decidable by Theorem 10. The following theorem remedies the negative result of Theorem 9.

Theorem 12. *For every self-join-free conjunctive query $\mathcal{Q}$, there exists a computable strategy φ of the form (5.3) for $\mathcal{Q}$, such that for every strategy ψ of the form (5.3) for $\mathcal{Q}$, $\psi \sqsubseteq \varphi$.*

Proof. Assume that the input to **OPTSTRATEGY** is a self-join-free conjunctive query $\mathcal{Q}(\vec{z})$. Let φ be the strategy defined in the paragraph preceding this theorem. Let $q = \{\vec{z}_0 \mid \exists\vec{X}\,(\lfloor\exists\vec{y}\,(B)\rfloor)\}$ be a query of the form (5.4) where B is a self-join-free conjunction of atoms such that q is in **FO** and $q \sqsubseteq \lfloor\mathcal{Q}\rfloor$. If all constants that occur in B also occur in $\mathcal{Q}$, then q is already contained in some disjunct of φ (by construction of φ). Assume next that B contains some constants that do not occur in $\mathcal{Q}$, and let these constants be $\mathsf{a}_1, \ldots, \mathsf{a}_m$. For $i \in \{1, 2, \ldots, m\}$, let X_i be a new fresh variable. Let B' be the conjunction obtained from B by replacing each occurrence of each a_i with X_i. Let $q' = \{\vec{z}_0 \mid \exists\vec{X}\,(\exists X_i\,\exists X_i\,\ldots\,\exists X_i\,(\lfloor\exists\vec{y}\,(B')\rfloor))\}$.

From $q \sqsubseteq \lfloor\mathcal{Q}\rfloor$ and Lemma 7, it follows that $\{\vec{z}_0 \mid \exists\vec{X}\,\exists\vec{y}\,(B)\} \sqsubseteq \mathcal{Q}$. By the Homomorphism Theorem [AHV95, Theorem 6.2.3], we can assume a homomorphism θ from $\mathcal{Q}$ to $\{\vec{z}_0 \mid \exists\vec{X}\,\exists\vec{y}\,(B)\}$. Notice that if $\theta(t) = \mathsf{a}_i$ for some symbol t that occurs in $\mathcal{Q}$ and $i \in \{1, 2, \ldots, m\}$, then it must be the case that t is a variable (because a_i does not occur in $\mathcal{Q}$). Let θ' be the substitution obtained from θ such that for every variable v in $\mathcal{Q}$ and $i \in \{1, 2, \ldots, m\}$,

$$\theta'(v) = \begin{cases} X_i & \text{if } \theta(v) = \mathsf{a}_i \\ \theta(v) & \text{otherwise.} \end{cases}$$

Obviously θ' is a homomorphism from $\mathcal{Q}$ to $\{\vec{z}_0 \mid \exists\vec{X}\,(\exists X_1\,\exists X_2\,\ldots\,\exists X_m\,(B'))\}$. From the Homomorphism Theorem and Lemma 7, it follows $q' \sqsubseteq \lfloor\mathcal{Q}\rfloor$. It can be easily seen that $q \sqsubseteq q'$. Furthermore, q' is in **FO** because q is in **FO** and it can be easily argued that membership in **FO** is preserved if constants are replaced with free variables. Notice here that each variable X_i is free in $\lfloor\exists\vec{y}\,(B')\rfloor$. Since all constants that occur in B' also occur in $\mathcal{Q}$, we have that q' is already contained in some disjunct of φ (by construction of φ).

To conclude, whenever $q = \{\vec{z}_0 \mid \exists\vec{X}\,(\lfloor\exists\vec{y}\,(B)\rfloor)\}$ is a query of the form (5.4) where B is a self-join-free conjunction of atoms such that q is in **FO** and

$q \sqsubseteq \lfloor \mathcal{Q} \rfloor$, we have that $\varphi \cup q \sqsubseteq \varphi$. □

So far, we have imposed no restrictions on the size of the computable strategy φ in the statement of Theorem 12. From a practical point of view, it is interesting to construct, among all optimal strategies φ of the form (5.3), the one with the smallest number l of disjuncts. This problem will be addressed in the next section.

5.5 Simplifying Strategies

In Section 5.4.2, we considered strategies that are unions of **CQAFO** queries of the form (5.4). A natural question is whether such strategies can be simplified. One obvious simplification is to remove any component of the union that is contained in another component, which requires an effective procedure for deciding containment between queries of the from (5.4). Developing such a procedure turns out to be a challenging problem. In Section 5.5.1, we illustrate this problem and introduce some simplifying assumptions. We will tackle this problem by using an existing tool, called attack graph, which was defined in Section 3.2 and which we generalize to account for the two queries involved in a containment test (Section 5.5.3). In Section 5.5.4, we provide an algorithm `ContainedIn` that decides containment of **CQAFO** queries of the form (5.4) under some additional restrictions.

5.5.1 Problem Statement and Motivation

We consider strategies $q_1 \cup q_2 \cup \cdots \cup q_l$ consisting of **CQAFO** queries q_i of the form

$$\left\{ \vec{z}_i \mid \exists \vec{X}_i \left(\lfloor \exists \vec{y}_i \left(B_i \right) \rfloor \right) \right\}.$$

Clearly, if some q_i is contained in another q_j (i.e., $q_i \sqsubseteq q_j$ with $i \neq j$), then the presence of q_i in the strategy is vacuous and q_i is redundant. That is, an equivalent shorter strategy is obtained by removing q_i from the union. This raises an important and interesting research question:

Given two **CQAFO** queries q_1 and q_2 of the form (5.4), decide whether $q_1 \sqsubseteq q_2$.

Theorem 11 settles containment of $q_2 \sqsubseteq \lfloor \mathcal{Q}_1 \rfloor$. In this containment, the right-hand side $\lfloor \mathcal{Q}_1 \rfloor$ is restricted to have no quantifier outside the scope of $\lfloor \cdot \rfloor$. The opposite containment $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq q_2$ turns out to be more difficult to handle, as illustrated next.

Example 32. *Consider the following two Boolean queries:*

$$\begin{aligned} \mathcal{Q}_2 &= \exists u \exists v \exists w \left(\mathtt{R}(\underline{u}, w) \wedge \mathtt{S}(\underline{v}, w) \right); \\ q_2 &= \exists U \left(\left\lfloor \exists v \exists w \left(\mathtt{R}(\underline{U}, w) \wedge \mathtt{S}(\underline{v}, w) \right) \right\rfloor \right), \end{aligned}$$

and consider a database (call it **db***) with the following tables, where for readability, columns are named by variables, and blocks are separated by dashed lines.*

R	$\underline{u}$	w
	a	1
	b	2

S	$\underline{v}$	w
	a	1
	b	2

The database **db** *has two repairs, each satisfying* $\mathcal{Q}_2$*, hence* $\mathbf{db} \models \lfloor \mathcal{Q}_2 \rfloor$*. However,* $\mathbf{db} \not\models q_2$*, because the two repairs of* **db** *use different values for* u *(*a *and* b*) to make the query true. So it is correct to conclude* $\lfloor \mathcal{Q}_2 \rfloor \not\sqsubseteq q_2$*.*

Consider furthermore the following query $\mathcal{Q}_1$*:*

$$\mathcal{Q}_1 = \exists x \exists y \left(\mathtt{R}(\underline{x}, y) \wedge \mathtt{S}(\underline{x}, y) \right).$$

By means of the Homomorphism Theorem [AHV95, p. 117], it can be verified that $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$*, hence* $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$ *by Lemma 5. It takes some effort to see that if a database satisfies* $\lfloor \mathcal{Q}_1 \rfloor$*, then it must contain two singleton blocks of the form* $\{\mathtt{R}(\underline{\mathsf{d}}, \mathsf{e})\}$ *and* $\{\mathtt{S}(\underline{\mathsf{d}}, \mathsf{e})\}$*, as follows.*

R	$\underline{x}$	y
	d	e
	⋮	

S	$\underline{x}$	y
	d	e
	⋮	

Such database will necessarily satisfy q_2*, hence* $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq q_2$.

It turns out that the containment problem for queries of the form (5.4) is quite challenging. To ease the technical treatment, we make the following simplifications:

- We will only deal with Boolean conjunctive queries (i.e., henceforth, all variables are assumed to be quantified). By Proposition 1, the restriction to Boolean queries does not compromise generality. At some places, it will be convenient (and unambiguous) to denote a Boolean conjunctive query by its set of atoms. For example, $\mathcal{Q}_1 = \{() \mid \exists x \, \exists y \, \exists z (\mathtt{R}(\underline{x}, z) \wedge \mathtt{S}(\underline{y}, z))\}$ can be denoted by the set $\{\mathtt{R}(\underline{x}, z), \mathtt{S}(\underline{y}, z)\}$.

- Let $\mathcal{Q}$ be a self-join-free conjunction of atoms. Let $\vec{X}$ be a sequence of distinct variables such that $\mathbf{vars}(\vec{X}) \subseteq \mathbf{vars}(\mathcal{Q})$. We write $\exists \vec{X} \, (\lfloor \mathcal{Q} \rfloor)$ for the query
$$\exists \vec{X} \, (\lfloor \exists \vec{u} \, (\mathcal{Q}) \rfloor) \, ,$$
where $\mathbf{vars}(\vec{u}) = \mathbf{vars}(\mathcal{Q}) \setminus \mathbf{vars}(\vec{X})$. That is, we only show the quantifiers that are outside the scope of $\lfloor \cdot \rfloor$.

- Our results concerning containment $\exists \vec{X}_1 \, (\lfloor \mathcal{Q}_1 \rfloor) \sqsubseteq \exists \vec{X}_2 \, (\lfloor \mathcal{Q}_2 \rfloor)$ will often require a homomorphism from $\mathcal{Q}_2$ to $\mathcal{Q}_1$ (which is tantamount to requiring $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, by the Homomorphism Theorem [AHV95, p. 117]). This requirement is reasonable, because if no such homomorphism exists, then $\exists \vec{X}_1 \, (\lfloor \mathcal{Q}_1 \rfloor) \not\sqsubseteq \exists \vec{X}_2 \, (\lfloor \mathcal{Q}_2 \rfloor)$ by Lemma 7. For completeness, we recall here that a *homomorphism* from $\mathcal{Q}_2$ to $\mathcal{Q}_1$ is a mapping h with domain $\mathbf{vars}(\mathcal{Q}_2)$ such that for every atom $\mathtt{R}(s_1, s_2, \ldots, s_l)$ in $\mathcal{Q}_2$, we have that $\mathtt{R}(h(s_1), h(s_2), \ldots, h(s_l))$ belongs to $\mathcal{Q}_1$.

Proposition 1. *Let q_2 and q_1 be two **CQAFO** queries of the form (5.4). One can compute in polynomial time two Boolean **CQAFO** queries q_2' and q_1', both of the form (5.4), such that $q_1 \sqsubseteq q_2$ if and only if $q_1' \sqsubseteq q_2'$.*

Proof. We can assume self-join-free conjunctions of atoms, B_1 and B_2, such that:

$$\begin{aligned} q_1 &= \left\{\vec{z}_1 \mid \exists \vec{X}_1 \left(\lfloor \exists \vec{y}_1 \left(B_1\right) \rfloor\right)\right\}; \\ q_2 &= \left\{\vec{z}_2 \mid \exists \vec{X}_2 \left(\lfloor \exists \vec{y}_2 \left(B_2\right) \rfloor\right)\right\}. \end{aligned}$$

Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be the queries obtained from respectively q_1 and q_2 by omitting $\lfloor \cdot \rfloor$, that is,

$$\begin{aligned} \mathcal{Q}_1 &= \left\{\vec{z}_1 \mid \exists \vec{X}_1 \left(\exists \vec{y}_1 \left(B_1\right)\right)\right\}; \\ \mathcal{Q}_2 &= \left\{\vec{z}_2 \mid \exists \vec{X}_2 \left(\exists \vec{y}_2 \left(B_2\right)\right)\right\}. \end{aligned}$$

If $\mathcal{Q}_1 \not\sqsubseteq \mathcal{Q}_2$, then $q_1 \not\sqsubseteq q_2$ by Lemma 7. In this case, pick two distinct key-equal facts A and B and let $q_1' = A$ and $q_2' = B$. Clearly, $q_1' \not\sqsubseteq q_2'$. Notice that the test $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ can be performed in polynomial time in the absence of self-joins.

Assume next $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$. By the Homomorphism Theorem [AHV95, Theorem 6.2.3], we can assume a valuation θ over $\mathbf{vars}(B_2)$ such that $\theta(B_2) \subseteq B_1$ and $\theta(\vec{z}_2) = \vec{z}_1$. Let μ be a valuation over $\mathbf{vars}(\vec{z}_1)$ that maps distinct variables to distinct fresh constants. Let $q_1' := \{\mu(\vec{z}_1) \mid \exists \vec{X}_1 \left(\lfloor \exists \vec{y}_1 \left(\mu(B_1)\right)\rfloor\right)\}$, the query obtained from q_1 by replacing each occurrence of each variable $z_1 \in \mathbf{vars}(\vec{z}_1)$ with $\mu(z_1)$. Intuitively, q_1' is the Boolean query obtained from q_1 by treating free variables as constants. Since B_1 is self-join-free, it can be seen that $q_1 \sqsubseteq q_2$ if and only if $q_1' \sqsubseteq q_2$.

For example, for $q_1 = \{z \mid \exists X(\lfloor \exists y(\mathtt{R}(\underline{X}, y, \mathtt{b}) \wedge \mathtt{S}(\underline{X}, y, z))\rfloor)\}$ with $\mathtt{b}$ a constant and distinct relation names $\mathtt{R}$ and $\mathtt{S}$, we would have that $q_1' = \{(\mathtt{c}) \mid \exists X(\lfloor \exists y(\mathtt{R}(\underline{X}, y, \mathtt{b}) \wedge \mathtt{S}(\underline{X}, y, \mathtt{c}))\rfloor)\}$, where $\mathtt{c}$ is a fresh constant. Notice that the above construction would make no sense in the presence of self-joins. In particular, if $\mathtt{R} = \mathtt{S}$, then any answer to q_1' would be empty (because $\mathtt{b} \neq \mathtt{c}$).

Since the answer to q_1' is either empty or the singleton $\{\mu(\vec{z}_1)\}$, the containment $q_1' \sqsubseteq q_2$ holds if q_2 returns $\{\mu(\vec{z}_1)\}$ whenever q_1' does. Let q_2' be the query

obtained from q_2 by replacing each occurrence of each variable $z_2 \in \mathbf{vars}(\vec{z}_2)$ with $\mu \circ \theta(z_2)$. That is, the free tuple in q_2' is equal to $\mu(\vec{z}_1)$. It is now obvious that $q_1' \sqsubseteq q_2$ if and only if $q_1' \sqsubseteq q_2'$. This concludes the proof. $\square$

To sum up, we start with two Boolean conjunctive queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ such that $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ (and hence $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$ by Lemma 5), and we want to know which existential quantification can be moved "outside the scope of $\lfloor \cdot \rfloor$" while preserving the containment $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$. For the left-hand side (i.e., $\lfloor \mathcal{Q}_1 \rfloor$), this is easy because, by Lemma 7, $\exists \vec{X}_1 \left(\lfloor \mathcal{Q}_1 \rfloor \right) \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$ if and only if $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$. For the right-hand side (i.e., $\lfloor \mathcal{Q}_2 \rfloor$), our major result will be an algorithm for deciding the containment $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \exists \vec{X}_2 \left(\lfloor \mathcal{Q}_2 \rfloor \right)$ (Theorem 14), albeit by imposing some further restrictions on $\mathcal{Q}_1$. We leave the design of a general containment test for future work.

To explain how the containment test works, we rely on the notion of attack graph which is defined relative to a single query (Section 5.5.2) and then introduce a new notion of attack that takes into account two queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$ related by a homomorphism (Section 5.5.3).

5.5.2 Attacks from Atoms to Variables

Attack graphs were defined in Section 3.2. The attacks defined there are from an atom to another atom. Attacks from an atom to a variable are defined as follows:

$$F \overset{\mathcal{Q}}{\rightsquigarrow} x \text{ if } F \overset{\mathcal{Q} \cup \{\mathtt{N}(\underline{x})\}}{\rightsquigarrow} \mathtt{N}(\underline{x}),$$

where $\mathtt{N}$ is a new relation name with signature $[1, 1]$. That is, $F \overset{\mathcal{Q}}{\rightsquigarrow} x$ if there is an attack from F to the "dummy" atom $\mathtt{N}(\underline{x})$ in the attack graph of $\mathcal{Q} \cup \{\mathtt{N}(\underline{x})\}$. The following lemma gives an important semantic property of unattacked variables.

Lemma 8. *Let $\mathcal{Q}$ be a self-join-free Boolean conjunctive query. Let $x \in \mathbf{vars}(\mathcal{Q})$ such that for every atom F of $\mathcal{Q}$, $F \overset{\mathcal{Q}}{\not\rightsquigarrow} x$. Then for every database $\mathbf{db}$ such that $\mathbf{db} \models \lfloor \mathcal{Q} \rfloor$, there exists a constant c such that $\mathbf{db} \models \lfloor \mathcal{Q}_{x \to \mathsf{c}} \rfloor$.*

Proof. Let $\mathcal{Q}' = \mathcal{Q} \cup \{\mathtt{N}(\underline{x})\}$ where $\mathtt{N}$ is a fresh relation name. The attack graph of $\mathcal{Q}'$ can be obtained from the attack graph of $\mathcal{Q}$ by adding the isolated vertex $\mathtt{N}(\underline{x})$. The desired result then follows from [KW17, Lemma 4.4]. □

The proof of the following lemma is analogous to the proof of Lemma C.1 in [Wij12]. Intuitively, it states that no new attacks emerge if we replace a variable with a constant in a Boolean self-join-free conjunctive query.

Lemma 9. *Let $\mathcal{Q}$ be a self-join-free Boolean conjunctive query. Let $\mathtt{c}$ be a constant and let $\mathcal{Q}' = \mathcal{Q}_{x \to \mathtt{c}}$. For every $F \in \mathcal{Q}$, let F' be the atom in $\mathcal{Q}'$ with the same relation name as F. For all $F, G \in \mathcal{Q}$, if $F' \overset{\mathcal{Q}'}{\rightsquigarrow} G'$, then $F \overset{\mathcal{Q}}{\rightsquigarrow} G$.*

Let $\mathcal{Q}$ be a self-join-free Boolean conjunctive query such that the attack graph of $\mathcal{Q}$ is acyclic. To avoid non-determinism in some definitions and results to follow, assume a lexicographic order on the atoms of $\mathcal{Q}$. We write $\mathbf{head}(\mathcal{Q})$ to denote the first (in lexicographic order) atom of $\mathcal{Q}$ that has no incoming attacks in the attack graph of $\mathcal{Q}$.

5.5.3 A New Attack Notion

We now define a generalized attack notion, which refers to two Boolean conjunctive queries, $\mathcal{Q}_1$ and $\mathcal{Q}_2$, such that there exists a homomorphism from $\mathcal{Q}_2$ to $\mathcal{Q}_1$. This new attack notion, denoted by the symbol $\cdot \overset{\mathcal{Q}_2 \mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} \cdot$, turns out to be a useful tool in the study of the containment problem for queries of the form (5.4).

Definition 9. *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be self-join-free Boolean conjunctive queries such that there exists a homomorphism (call it h) from $\mathcal{Q}_2$ to $\mathcal{Q}_1$. Notice that such a homomorphism, if it exists, is unique (because the queries are self-join-free). Let G and H be distinct atoms of $\mathcal{Q}_2$. We write*

$$G \overset{\mathcal{Q}_2 \mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} H$$

if there exists a sequence

$$G_0 \overset{u_1}{\frown} G_1 \overset{u_2}{\frown} G_2 \ldots \overset{u_l}{\frown} G_l \tag{5.5}$$

such that

1. $G_0, G_1, \ldots, G_l$ *are atoms of* $\mathcal{Q}_2$ *such that* $G_0 = G$ *and* $G_l = H$;

2. *for all* $i \in \{1, 2, \ldots, l\}$, $u_i \in \mathbf{vars}(G_{i-1}) \cap \mathbf{vars}(G_i)$;

3. *for all* $i \in \{1, 2, \ldots, l\}$, $h(u_i)$ *is a variable that does not belong to* $(h(G_0))^{+,\mathcal{Q}_1}$.

Let $u \in \mathbf{vars}(\mathcal{Q}_2)$. *We write*

$$G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} u$$

if

$$G \overset{\mathcal{Q}_2'\mathcal{Q}_1'}{\rightsquigarrow\rightsquigarrow} \mathtt{N}(\underline{u})$$

where

1. $\mathtt{N}$ *is a new relation name with signature* $[1, 1]$;

2. $\mathcal{Q}_2' = \mathcal{Q}_2 \cup \{\mathtt{N}(\underline{u})\}$; *and*

3. $\mathcal{Q}_1' = \mathcal{Q}_1 \cup \{\mathtt{N}(\underline{h(u)})\}$.

Notice that if $h(u)$ *is a constant, then* $G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow\rightsquigarrow} u$. *Note also that for every atom* F *of* $\mathcal{Q}_1$, $F^{+,\mathcal{Q}_1} = F^{+,\mathcal{Q}_1'}$.

Intuitively, $G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} H$ holds true if there exists a sequence of the form (5.5) whose image under the homomorphism h is a witness for $h(G) \overset{\mathcal{Q}_1}{\rightsquigarrow} h(H)$. The notion $\overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow}$ is a proper generalization of $\overset{\mathcal{Q}_1}{\rightsquigarrow}$, because for $\mathcal{Q}_1 = \mathcal{Q}_2$, the relationship $\overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow}$ is the same as $\overset{\mathcal{Q}_1}{\rightsquigarrow}$. That is, $F \overset{\mathcal{Q}_1\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} F'$ if and only if $F \overset{\mathcal{Q}_1}{\rightsquigarrow} F'$.

Example 33. *Let* $\mathcal{Q}_2 = \{\mathtt{R}(\underline{u}, x)\}$ *and* $\mathcal{Q}_1 = \{\mathtt{R}(\underline{y}, z), \mathtt{S}(\underline{z})\}$. *Then,* $\mathtt{R}(\underline{u}, x) \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\rightsquigarrow} x$ *because* $\mathtt{R}(\underline{u}, x) \overset{\mathcal{Q}_2'\mathcal{Q}_1'}{\rightsquigarrow\rightsquigarrow} \mathtt{N}(\underline{x})$, *where* $\mathcal{Q}_2' = \{\mathtt{R}(\underline{u}, x), \mathtt{N}(\underline{x})\}$ *and* $\mathcal{Q}_1' = \{\mathtt{R}(\underline{y}, z), \mathtt{S}(\underline{z}), \mathtt{N}(\underline{z})\}$. *Indeed, note that* $\mathtt{R}(\underline{u}, x)$ *and* $\mathtt{N}(\underline{x})$ *share the variable* x, *and* $h(x) = z$ *does not belong to* $\mathtt{R}^{+,\mathcal{Q}_1'} = \{y\}$.

Example 34. *Consider the following two queries:*

$$\begin{aligned} \mathcal{Q}_2 &= \left\{\mathtt{R}(\underline{\mathtt{a}}, u), \mathtt{S}(\underline{u}, x), \mathtt{T}(\underline{x'})\right\}; \\ \mathcal{Q}_1 &= \left\{\mathtt{R}(\underline{\mathtt{a}}, y), \mathtt{S}(\underline{y}, z), \mathtt{T}(\underline{z})\right\}, \end{aligned}$$

and let h be the (unique) homomorphism from $\mathcal{Q}_2$ to $\mathcal{Q}_1$. Notice that $h(x) = h(x') = z$. Since $\mathbf{keyvars}(\mathtt{R}) = \emptyset$ *in $\mathcal{Q}_1$, but* $\mathbf{keyvars}(\mathtt{S}) \neq \emptyset \neq \mathbf{keyvars}(\mathtt{T})$*, we have $\mathtt{R}^{+,\mathcal{Q}_1} = \emptyset$. Hence, $\mathtt{R}(\underline{\mathtt{a}}, u) \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} x$ and $\mathtt{R}(\underline{\mathtt{a}}, u) \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$ trivially hold. Note, however, that $\mathtt{R}(\underline{\mathtt{a}}, u) \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} x'$. This is because the atom $\mathtt{T}(\underline{x'})$ shares no variable with any other atom of $\mathcal{Q}_2$.*

5.5.4 Testing containment

The following theorem expresses a significant relationship between $\overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow}$ and query containment for queries of the form (5.4). Paraphrasing somewhat, if $\lfloor\mathcal{Q}_1\rfloor \sqsubseteq \lfloor\mathcal{Q}_2\rfloor$ and $u \in \mathbf{vars}(\mathcal{Q}_2)$ such that $G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$ for some $G \in \mathcal{Q}_2$, then query containment is lost if the quantification of the variable u is moved outside the scope of $\lfloor\cdot\rfloor$. It is an open question whether the inverse of Theorem 13 also holds.

Theorem 13. *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be self-join-free Boolean conjunctive queries such that there exists a homomorphism (call it h) from $\mathcal{Q}_2$ to $\mathcal{Q}_1$. Let $u \in$* $\mathbf{vars}(\mathcal{Q}_2)$*. If $G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$ for some $G \in \mathcal{Q}_2$, then $\lfloor\mathcal{Q}_1\rfloor \not\sqsubseteq \exists u\,(\lfloor\mathcal{Q}_2\rfloor)$.*

Proof. We first fix some notations. Let $G_0 \in \mathcal{Q}_2$ such that $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$. Let $h(G_0) = F_0$ and $h(u) = w$. Assume that $\mathtt{R}_0$ is the relation name of G_0 (which is necessarily equal to the relation name of F_0). We show that $\lfloor\mathcal{Q}_1\rfloor \not\sqsubseteq \exists u\,(\lfloor\mathcal{Q}_2\rfloor)$ by constructing a database instance **db** such that $\mathbf{db} \models \lfloor\mathcal{Q}_1\rfloor$ but $\mathbf{db} \not\models \exists u\,(\lfloor\mathcal{Q}_2\rfloor)$.

To define **db**, let θ, μ be two valuations over $\mathbf{vars}(\mathcal{Q}_1)$ such that for every $x \in \mathbf{vars}(\mathcal{Q}_1)$, $\theta(x) = \mu(x)$ if and only if $x \in F_0^{+,\mathcal{Q}_1}$. Assume that $\mathcal{Q}_1 = \{() \mid \exists\vec{y}\,(B_1)\}$. Let $\mathbf{db} = \theta(B_1) \cup \mu(B_1)$. We next show that **db** has only two

repairs, denoted by $\mathbf{r}$ and $\mathbf{s}$, where

$$\begin{aligned}\mathbf{r} &= \mathbf{db} \setminus \{\mu(F_0)\}\,;\\ \mathbf{s} &= \mathbf{db} \setminus \{\theta(F_0)\}\,.\end{aligned}$$

To see that these are repairs, we first show that for every $F \in \mathcal{Q}_1 \setminus \{F_0\}$, the facts $\theta(F)$ and $\mu(F)$ are either equal or not key-equal, i.e., they never constitute two distinct facts of a same block. Indeed, for every $F \in \mathcal{Q}_1 \setminus \{F_0\}$, two cases are possible:

Case $\mathbf{keyvars}(F) \subseteq F_0^{+,\mathcal{Q}_1}$. Then, $\mathbf{vars}(F) \subseteq F_0^{+,\mathcal{Q}_1}$, and thus θ and μ agree on all variables of $\mathbf{vars}(F)$. That is, $\theta(F) = \mu(F)$.

Case $\mathbf{keyvars}(F) \nsubseteq F_0^{+,\mathcal{Q}_1}$. Then, by the definition of θ and μ, for some variable $x \in \mathbf{keyvars}(F)$, $\theta(x) \neq \mu(x)$, hence $\theta(F)$ and $\mu(F)$ are not key-equal.

Furthermore, when considering F_0, $\theta(F_0)$ and $\mu(F_0)$ are distinct and key-equal (hence, $\mathbf{r}$ contains $\theta(F_0)$ and $\mathbf{s}$ contains $\mu(F_0)$). The facts $\theta(F_0)$ and $\mu(F_0)$ are key-equal because $\mathbf{keyvars}(F_0) \subseteq F_0^{+,\mathcal{Q}_1}$ is obvious. Further, from $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\!\!\rightsquigarrow} u$, we can assume some variable $y \in \mathbf{vars}(F_0)$ such that $F_0 \overset{\mathcal{Q}_1}{\rightsquigarrow} y$, hence $y \notin F_0^{+,\mathcal{Q}_1}$. Since θ and μ disagree on y, we have $\theta(F_0) \neq \mu(F_0)$. Clearly, $\mathbf{r}$ and $\mathbf{s}$ are the only repairs of $\mathbf{db}$, since $\{\theta(F_0), \mu(F_0)\}$ is the only block of $\mathbf{db}$ with more than one fact.

It is obvious that $\mathbf{r} \models \mathcal{Q}_1$ and $\mathbf{s} \models \mathcal{Q}_1$, hence $\mathbf{db} \models \lfloor \mathcal{Q}_1 \rfloor$ since $\mathbf{r}$ and $\mathbf{s}$ are the only repairs of $\mathbf{db}$. We now show that $\mathbf{db} \not\models \exists u\,(\lfloor \mathcal{Q}_2 \rfloor)$, or in other words, that there is no constant $\mathtt{c}$ such that both $\mathbf{r} \models \mathcal{Q}_{2u\to\mathtt{c}}$ and $\mathbf{s} \models \mathcal{Q}_{2u\to\mathtt{c}}$. First, we show that if $\mathbf{r} \models \mathcal{Q}_{2u\to\mathtt{c}}$ and $\mathbf{s} \models \mathcal{Q}_{2u\to\mathtt{c}}$ for some constant $\mathtt{c}$, then it must be the case that either $\mathtt{c} = \mu(w)$ or $\mathtt{c} = \theta(w)$. Indeed, for every valuation α over $\mathbf{vars}(\mathcal{Q}_2)$ such that $\alpha(\mathcal{Q}_2) \subseteq \mathbf{r}$, we have $\alpha(u) \in \{\mu(w), \theta(w)\}$. Likewise, for every valuation β over $\mathbf{vars}(\mathcal{Q}_2)$ such that $\beta(\mathcal{Q}_2) \subseteq \mathbf{s}$, we have $\beta(u) \in \{\mu(w), \theta(w)\}$. Second, we show that $\mu(w) \neq \theta(w)$. Indeed, from $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\!\!\rightsquigarrow} u$, it is correct to conclude $w \notin F_0^{+,\mathcal{Q}_1}$. To see this, consider a sequence $G_0 \overset{u_1}{\frown} G_1 \overset{u_2}{\frown} G_2 \ldots \overset{u}{\frown} \mathtt{N}(\underline{u})$ witnessing that $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow\!\!\rightsquigarrow} u$. Then,

$h(u) = w \notin (h(G_0))^{+,\mathcal{Q}_1'} = F_0^{+,\mathcal{Q}_1}$. From the definition of μ and θ, it is correct to conclude that $\mu(w) \neq \theta(w)$. Finally, we show that $\mathbf{r} \not\models \mathcal{Q}_{2u\to\mu(w)}$ and $\mathbf{s} \not\models \mathcal{Q}_{2u\to\theta(w)}$. This suffices to show that $\mathbf{db} \not\models \exists u\,(\lfloor\mathcal{Q}_2\rfloor)$.

We show $\mathbf{r} \not\models \mathcal{Q}_{2u\to\mu(w)}$ (the proof of $\mathbf{s} \not\models \mathcal{Q}_{2u\to\theta(w)}$ is symmetrical). More specifically, we show that any valuation α over $\mathbf{vars}(\mathcal{Q}_2)$ such that $\alpha(\mathcal{Q}_2) \subseteq \mathbf{r}$ satisfies $\alpha(u) = \theta(w)$. Hence, $\alpha(u) \neq \mu(w)$ for any such valuation α and it is correct to infer that $\mathbf{r} \not\models \mathcal{Q}_{2u\to\mu(w)}$.

It is easily verified that from $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$, it follows that for some $l \geq 0$, there exists a sequence

$$G_0 \overset{u_1}{\frown} G_1 \overset{u_2}{\frown} G_2 \ldots \overset{u_l}{\frown} G_l \tag{5.6}$$

such that

1. $G_0, G_1, \ldots, G_l$ are atoms of $\mathcal{Q}_2$;
2. $u \in \mathbf{vars}(G_l)$;
3. for all $i \in \{1, 2, \ldots, l\}$, $u_i \in \mathbf{vars}(G_{i-1}) \cap \mathbf{vars}(G_i)$; and
4. for all $i \in \{1, 2, \ldots, l\}$, $h(u_i)$ is a variable such that $\mu(h(u_i)) \neq \theta(h(u_i))$.

Observe that the latter condition is equivalent to $h(u_i) \notin h(G_0)^{+,\mathcal{Q}_1} = F_0^{+,\mathcal{Q}_1}$ (for all $i \in \{1, 2, \ldots, l\}$). For every $i \in \{1, 2, \ldots, l\}$, define $w_i := h(u_i)$. Let α be a valuation over $\mathbf{vars}(\mathcal{Q}_2)$ such that $\alpha(\mathcal{Q}_2) \subseteq \mathbf{r}$. Based on the sequence (5.6), we show by induction on increasing i that for $i \in \{0, 1, \ldots, l\}$, $\alpha(G_i) = \theta(F_i)$. This suffices since if this holds, then $\alpha(G_l) = \theta(F_l)$ and since $u \in \mathbf{vars}(G_l)$, $\alpha(u) = \theta(w)$.

The induction hypothesis trivially holds for $i = 0$. Indeed, as argued above, $\theta(F_0)$ is the only $\mathtt{R}_0$-fact of $\mathbf{r}$.

For the induction step, $i \mapsto i + 1$, the induction hypothesis is that for all $j \in \{0, 1, \ldots, i\}$, $\alpha(G_j) = \theta(F_j)$. Clearly, since $u_{i+1} \in \mathbf{vars}(G_i)$, we have that $\alpha(u_{i+1}) = \theta(u_{i+1})$. Then, since $u_{i+1} \in \mathbf{vars}(G_{i+1})$ and $\theta(w_{i+1}) \neq \mu(w_{i+1})$, it must be the case that $\alpha(G_{i+1}) = \theta(F_{i+1})$.

So we obtain $\alpha(G_l) = \theta(F_l)$, hence $\alpha(u) = \theta(w)$. This concludes the proof. □

As already mentioned, it is an open question whether the inverse of Theorem 13 also holds:

> From $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \lfloor \mathcal{Q}_2 \rfloor$, $u \in \textbf{vars}(\mathcal{Q}_2)$, and $G \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} u$ for all $G \in \mathcal{Q}_2$,
> is it correct to conclude $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \exists u\, (\lfloor \mathcal{Q}_2 \rfloor)$?

Theorem 14 provides a positive answer to this question under restrictions on $\mathcal{Q}_1$. The theorem is stated in the form of the function `ContainedIn`, which recursively checks whether the variable u has an incoming $\overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow}$-attack. The function will be called once for every atom of $\mathcal{Q}_2$. We briefly discuss the restrictions imposed on $\mathcal{Q}_1$ by Theorem 14.

- The restriction that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ have the same cardinality can be easily met, because we can always add "dummy" atoms to a conjunctive query without affecting query containment. For example, if $\mathcal{Q}_1$ contains an R-atom with signature $[n, k]$, but $\mathcal{Q}_2$ contains no R-atom, then we can add to $\mathcal{Q}_2$ the dummy atom $\texttt{R}(\underline{u_1, u_2, \ldots, u_k}, u_{k+1}, u_{k+2}, \ldots, u_n)$, where each u_i is a fresh variable not occurring elsewhere.

- The restriction that $\lfloor \mathcal{Q}_1 \rfloor$ is in **FO** is not problematic for the application we have in mind, which, as explained in Section 5.5.1, is the simplification of strategies, which are unions of queries of the form (5.4) that are in **FO**. Notice that no such restriction is imposed on $\lfloor \mathcal{Q}_2 \rfloor$, which can thus be a query not in **FO**.

- The more technical restriction is $F^{+,\mathcal{Q}_1} \subseteq \textbf{vars}(F)$. This restriction is met, for example, by the queries $\mathcal{Q}_{11} = \exists x\, \exists y (\texttt{R}(\underline{x}, y) \wedge \texttt{S}(\underline{x}, y))$ and $\mathcal{Q}_{12} = \exists x\, \exists y\, \exists z (\texttt{R}(\underline{x}, y) \wedge \texttt{S}(\underline{y}, z))$, but not by the query $\mathcal{Q}_{13} = \exists x\, \exists y (\texttt{R}(\underline{x}, y) \wedge \texttt{S}(\underline{x}, z))$ (because $\texttt{R}^{+,\mathcal{Q}_{13}} = \{x, z\}$ and $z \notin \textbf{vars}(\texttt{R})$). This restriction excludes some queries, but is not overly prohibitive. It is an open question whether Theorem 14 can be proved without relying on this restriction.

Theorem 14. *Let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be self-join-free Boolean conjunctive queries, of the same cardinality, such that there exists a homomorphism (call it h) from*

```
Function ContainedIn(Q1,Q2,u) is
  Data: self-join-free Boolean conjunctive queries Q2 and Q1 such
        that |Q2| = |Q1|, there exists a homomorphism from Q2 to
        Q1, and ⌊Q1⌋ is in FO; a variable u
  Result: Is ⌊Q1⌋ ⊑ ∃u (⌊Q2⌋)?
  if u ∉ vars(Q2) then
    return true
  else
    let h be the (unique) homomorphism from Q2 to Q1
    let F0 := head(Q1)
    let G0 be the (unique) atom of Q2 such that h(G0) = F0
    if G0 ⇝^{Q2Q1} u then
      return false
    else
      let Q̂1 := Q1 \ {F0}
      let Q̂2 := Q2 \ {G0}
      let α be an arbitrary valuation over vars(F0)
      return ContainedIn(α(Q̂1),Q̂2,u)
```

Function ContainedIn

$\mathcal{Q}_2$ to $\mathcal{Q}_1$. Assume that $\lfloor \mathcal{Q}_1 \rfloor$ is in **FO** *and that for every $F \in \mathcal{Q}_1$, it is the case that $F^{+,\mathcal{Q}_1} \subseteq$* **vars**$(F)$*. Then the following are equivalent for any variable u:*

1. `ContainedIn(`$\mathcal{Q}_1$,$\mathcal{Q}_2$,u`)` *returns true; and*
2. $\lfloor \mathcal{Q}_1 \rfloor \sqsubseteq \exists u \, (\lfloor \mathcal{Q}_2 \rfloor)$.

Proof. $\boxed{\boxed{2} \Longrightarrow \boxed{1}}$ Proof by contraposition. Assume that *false* is returned by `ContainedIn(`$\mathcal{Q}_1$,$\mathcal{Q}_2$,u`)`. Then, at some point in its execution, the test "**if** $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} u$" becomes true. Let $F_0, F_1, \dots, F_n$ be a topological sort of the attack graph of $\mathcal{Q}_1$ where ties are broken lexicographically. For every $i \in \{0, 1, \dots, n\}$, let G_i be the atom of $\mathcal{Q}_2$ with the same relation name as F_i (i.e.,

$h(G_i) = F_i$). Then, there exists $l \in \{0, 1, \ldots, n\}$ such that $G_l \overset{\mathcal{Q}_2' \alpha(\mathcal{Q}_1')}{\rightsquigarrow} u$ where

- $\mathcal{Q}_2' = \{G_l, G_{l+1}, \ldots, G_n\}$,
- $\mathcal{Q}_1' = \{F_l, F_{l+1}, \ldots, F_n\}$, and
- α is a valuation over $\mathbf{vars}(F_0) \cup \mathbf{vars}(F_1) \cup \cdots \cup \mathbf{vars}(F_{l-1})$.

We have $\alpha(F_l) \overset{\alpha(\mathcal{Q}_1')}{\rightsquigarrow} h(u)$. From Lemma 9, it follows $F_l \overset{\mathcal{Q}_1}{\rightsquigarrow} h(u)$. It is now easy to see $G_l \overset{\mathcal{Q}_2 \mathcal{Q}_1}{\rightsquigarrow} u$. By Theorem 13, $\lfloor \mathcal{Q}_1 \rfloor \not\sqsubseteq \exists u \left(\lfloor \mathcal{Q}_2 \rfloor \right)$.

$\boxed{\boxed{1} \Longrightarrow \boxed{2}}$ We use the following notations:

$$
\begin{aligned}
h &:= \text{(unique) homomorphism from } \mathcal{Q}_2 \text{ to } \mathcal{Q}_1; \\
F_0 &:= \mathbf{head}(\mathcal{Q}_1); \\
G_0 &:= \text{the (unique) atom in } \mathcal{Q}_2 \text{ such that } h(G_0) = F_0; \\
\hat{\mathcal{Q}}_1 &:= \mathcal{Q}_1 \setminus \{F_0\}; \\
\hat{\mathcal{Q}}_2 &:= \mathcal{Q}_2 \setminus \{G_0\}.
\end{aligned}
$$

The initial assumptions are the following:

1. `ContainedIn(`$\mathcal{Q}_1$`,`$\mathcal{Q}_2$`,`u`)` returns true;
2. **db** is a database such that every repair of **db** satisfies $\mathcal{Q}_1$.

The proof runs by structural induction. For the base case (i.e., $u \notin \mathbf{vars}(\mathcal{Q}_2)$), it is obvious that $\exists u \left(\lfloor \mathcal{Q}_2 \rfloor \right) \equiv \lfloor \mathcal{Q}_2 \rfloor$ and the desired result holds because there exists a homomorphism from $\mathcal{Q}_2$ to $\mathcal{Q}_1$. Assume hereinafter that $u \in \mathbf{vars}(\mathcal{Q}_2)$.

Since $\lfloor \mathcal{Q}_1 \rfloor$ is in **FO**, the attack graph of $\mathcal{Q}_1$ is acyclic. Let $\mathtt{R}_0, \mathtt{R}_1, \ldots, \mathtt{R}_n$ be a topological ordering of the attack graph of $\mathcal{Q}_1$, where ties are broken lexicographically. [3] Since $F_0 = \mathbf{head}(\mathcal{Q}_1)$, the relation name of F_0 is $\mathtt{R}_0$.

We need to show show that $\mathbf{db} \models \exists u \left(\lfloor \mathcal{Q}_2 \rfloor \right)$. Clearly, since $\mathbf{db} \models \lfloor \mathcal{Q}_1 \rfloor$, there must exist a (not necessarily unique) subset $\mathbf{db}_0$ of **db** such that

1. $\mathbf{db}_0 \models \lfloor \mathcal{Q}_1 \rfloor$;

[3] By an abuse of notation, we blur the distinction between atoms and their relation names.

2. for every block $\mathbf{b}$ of $\mathbf{db}$, either $\mathbf{b} \subseteq \mathbf{db}_0$ or $\mathbf{b} \cap \mathbf{db}_0 = \emptyset$.

3. *Minimality:* for every block $\mathbf{b}$ of $\mathbf{db}_0$, we have $\mathbf{db}_0 \setminus \mathbf{b} \not\models \lfloor \mathcal{Q}_1 \rfloor$.

In practice, $\mathbf{db}_0$ can be obtained from $\mathbf{db}$ by repeatedly removing blocks until the further removal of any more block would lead to a database that falsifies $\lfloor \mathcal{Q}_1 \rfloor$. We will show that $\mathbf{db}_0 \models \exists u\,(\lfloor \mathcal{Q}_2 \rfloor)$, which obviously implies $\mathbf{db} \models \exists u\,(\lfloor \mathcal{Q}_2 \rfloor)$ (because every repair of $\mathbf{db}$ contains a repair of $\mathbf{db}_0$).

Let the set of $\mathtt{R}_0$-facts in $\mathbf{db}_0$ be $\{A_1, A_2, \ldots, A_m\}$. For $1 \leq i \leq m$, denote by θ_i the (unique) valuation over $\mathbf{vars}(F_0)$ such that $\theta_i(F_0) = A_i$. We show the following:

> *Agreement Property:* For every $v \in \mathbf{vars}(F_0) \cap F_0^{+,\mathcal{Q}_1}$, for all $i, j \in \{1, 2, \ldots, m\}$, $\theta_i(v) = \theta_j(v)$.

To this extent, let $v \in \mathbf{vars}(F_0) \cap F_0^{+,\mathcal{Q}_1}$. Then, $F_0 \overset{\mathcal{Q}_1}{\not\rightsquigarrow} v$. Moreover, since F_0 has no incoming attacks in the attack graph of $\mathcal{Q}_1$, we have that for all $F \in \mathcal{Q}_1$, $F \overset{\mathcal{Q}_1}{\not\rightsquigarrow} v$. From Lemma 8, it follows that for all $i, j \in \{1, 2, \ldots, m\}$, $\theta_i(v) = \theta_j(v)$, which concludes the proof of the *Agreement Property*. Notice that from $\mathbf{keyvars}(F_0) \subseteq F_0^{+,\mathcal{Q}_1}$ and the *Agreement Property*, it follows that the set $\{A_1, A_2, \ldots, A_m\}$ is the unique $\mathtt{R}_0$-block of $\mathbf{db}_0$.

It suffices now to show that there exists a constant $\mathtt{b}$ (which depends on $\mathbf{db}_0$) such that every repair of $\mathbf{db}_0$ satisfies $\mathcal{Q}_{2u \to \mathtt{b}}$. We distinguish two cases, the first case being the easier one.

Case $u \in \mathbf{vars}(G_0)$

In this case, it can be shown that all $\mathtt{R}_0$-facts agree on the position at which u occurs in G_0. Indeed, from $G_0 \overset{\mathcal{Q}_2 \mathcal{Q}_1}{\rightsquigarrow\not\rightsquigarrow} u$ (since `ContainedIn(`$\mathcal{Q}_1$,$\mathcal{Q}_2$,u`)` returns true), it follows $h(u) \in F_0^{+,\mathcal{Q}_1}$. From $h(u) \in \mathbf{vars}(F_0)$ and the *Agreement Property*, it follows that for all $i, j \in \{1, 2, \ldots, m\}$, $\theta_i(h(u)) = \theta_j(h(u))$. In this case, the desired result holds for $\mathtt{b} = \theta_1(h(u))$.

Case $u \notin \mathbf{vars}(G_0)$

Let $\hat{\mathbf{db}}_0 := \mathbf{db}_0 \setminus \{A_1, A_2, \ldots, A_m\}$. For $i \in \{1, 2, \ldots, m\}$, denote by $\hat{\mathbf{db}}_0^i$ a minimal subset of $\hat{\mathbf{db}}_0$ such that $\hat{\mathbf{db}}_0^i \models \lfloor \theta_i(\hat{\mathcal{Q}}_1) \rfloor$ and every block of $\hat{\mathbf{db}}_0$ is either contained in $\hat{\mathbf{db}}_0^i$ or disjoint with $\hat{\mathbf{db}}_0^i$. That is, $\hat{\mathbf{db}}_0^i$ is obtained from $\hat{\mathbf{db}}_0$ relative to $\theta_i(\hat{\mathcal{Q}}_1)$ in exactly the same way as $\mathbf{db}_0$ was obtained from $\mathbf{db}$ relative to $\mathcal{Q}_1$. In the same way as $\{A_1, A_2, \ldots, A_m\}$ was shown to be the only $\mathtt{R}_0$-block of $\mathbf{db}_0$, it can be shown that for each $i \in \{1, 2, \ldots, m\}$, $\hat{\mathbf{db}}_0^i$ contains only one $\mathtt{R}_1$-block.

It follows from Lemma 9 that $\mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_n$ will be a topological sort of the attack graph of $\theta_i(\hat{\mathcal{Q}}_1)$ (for all $1 \leq i \leq m$). The following hold for any $i \in \{1, 2, \ldots, m\}$:

- from our initial hypothesis that $\texttt{ContainedIn(}\mathcal{Q}_1\texttt{,}\mathcal{Q}_2\texttt{,}u\texttt{)}$ returns true, it follows that $\texttt{ContainedIn(}\theta_i(\hat{\mathcal{Q}}_1)\texttt{,}\hat{\mathcal{Q}}_2\texttt{,}u\texttt{)}$ returns true; and
- by the induction hypothesis, there exists a constant b_i such that every repair $\hat{\mathbf{r}}$ of $\hat{\mathbf{db}}_0^i$ satisfies $\hat{\mathcal{Q}}_{2u\to\mathsf{b}_i}$.

We show that $\mathbf{db}_0 \models \lfloor \mathcal{Q}_{2u\to\mathsf{b}_1} \rfloor$ (i.e., we fix $i = 1$). By symmetry, it will actually follow that for every $i \in \{1, 2, \ldots, m\}$, $\mathbf{db}_0 \models \lfloor \mathcal{Q}_{2u\to\mathsf{b}_i} \rfloor$.

Let $\mathbf{r}$ be an arbitrary repair of $\mathbf{db}_0$. We need to show $\mathbf{r} \models \mathcal{Q}_{2u\to\mathsf{b}_1}$.

We can assume $l \in \{1, 2, \ldots, m\}$ such that $A_l \in \mathbf{r}$. Since $\mathbf{r} \models \mathcal{Q}_1$, there exists a valuation δ over $\mathbf{vars}(\mathcal{Q}_1)$ such that $\delta(\mathcal{Q}_1) \subseteq \mathbf{r}$ and $\delta(F_0) = A_l$. The latter follows because A_l is the only $\mathtt{R}_0$-fact in $\mathbf{r}$. Let α be the valuation over $\mathbf{vars}(\mathcal{Q}_2)$ such that for every $x \in \mathbf{vars}(\mathcal{Q}_2)$, $\alpha(x) = \delta(h(x))$. Obviously, $\alpha(\mathcal{Q}_2) = \delta(\mathcal{Q}_1) \subseteq \mathbf{r}$ and $\alpha(G_0) = A_l$.

Clearly, $\mathbf{r} \cap \hat{\mathbf{db}}_0^1$ is a repair of $\hat{\mathbf{db}}_0^1$. By the induction hypothesis, we can assume a valuation β over $\mathbf{vars}(\mathcal{Q}_2)$ such that

1. $\beta(\hat{\mathcal{Q}}_2) \subseteq \mathbf{r} \cap \hat{\mathbf{db}}_0^1$;
2. $\beta(u) = \mathsf{b}_1$; and
3. $\beta(G_0) = A_1$.

Notice that the induction hypothesis gives us the first two items. The last item follows from the construction of $\hat{\mathbf{db}}_0^1$.

Let γ be the valuation over $\mathbf{vars}(\mathcal{Q}_2)$ such that for every $x \in \mathbf{vars}(\mathcal{Q}_2)$,

$$\gamma(x) = \begin{cases} \alpha(x) & \text{if } G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} x \\ \beta(x) & \text{otherwise.} \end{cases} \tag{5.7}$$

From the construction of γ and $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} u$, it follows $\gamma(u) = \mathsf{b}_1$. It remains to be shown that $\gamma(\mathcal{Q}_2) \subseteq \mathbf{r}$. To this extent, let G be an arbitrary atom of $\mathcal{Q}_2$. It remains to be shown that $\gamma(G) \in \mathbf{r}$. We distinguish two cases.

Case $G = G_0$. Recall that $\alpha(G_0) = A_l$, $\beta(G_0) = A_1$, and $A_l \in \mathbf{r}$. We show that $\gamma(G_0) = \alpha(G_0) = A_l$. To this extent, let w be an arbitrary variable in $\mathbf{vars}(G_0)$. If $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} w$, then $\gamma(w) = \alpha(w)$ by the construction of γ in (5.7). Consider next $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} w$. Then it must be the case that $h(w) \in F_0^{+,\mathcal{Q}_1}$ and, by the *Agreement Property*, A_1 and A_l agree on the position at which w occurs in G_0. Then, $\alpha(w) = \beta(w)$.

Case $G \neq G_0$. Assume towards a contradiction $G \in \mathcal{Q}_2$ such that $\gamma(G) \notin \mathbf{r}$. Then, it must be the case that $\alpha(G) \neq \gamma(G) \neq \beta(G)$, because $\alpha(G)$ and $\beta(G)$ belong to $\mathbf{r}$. Then we can assume $y_1, y_2 \in \mathbf{vars}(G)$ such that $\gamma(y_1) = \alpha(y_1) \neq \beta(y_1)$ and $\gamma(y_2) = \beta(y_2) \neq \alpha(y_2)$. We next show a contradiction by proving $\alpha(y_2) = \beta(y_2)$.

Observe that by the construction of γ in (5.7), from $\gamma(y_1) = \alpha(y_1) \neq \beta(y_1)$, it follows $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} y_1$. Likewise, from $\gamma(y_2) = \beta(y_2) \neq \alpha(y_2)$, it follows $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} y_2$. We show next $h(y_2) \in F_0^{+,\mathcal{Q}_1}$.

From $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} y_1$ and $y_1 \in \mathbf{vars}(G)$, it follows $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} G$, which implies the existence of a sequence of the form (5.5) with $G_l = G$. Then for every variable $v \in \mathbf{vars}(G)$, either $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\rightsquigarrow} v$ or $h(v) \in F_0^{+,\mathcal{Q}_1}$. Since $y_2 \in \mathbf{vars}(G)$ and $G_0 \overset{\mathcal{Q}_2\mathcal{Q}_1}{\not\rightsquigarrow} y_2$, it must be the case $h(y_2) \in F_0^{+,\mathcal{Q}_1}$.

The statement of Theorem 14 makes the hypothesis that $F_0^{+,\mathcal{Q}_1} \subseteq \mathbf{vars}(F_0)$, hence $h(y_2) \in \mathbf{vars}(F_0)$. Then, by the *Agreement Property*, it is correct to conclude that for all $i, j \in \{1, 2, \ldots, m\}$, $\theta_i(h(y_2)) = \theta_j(h(y_2))$. In the

remainder of the proof, we denote by $\mathtt{d}$ the constant such that for all $i \in \{1, 2, \dots, m\}$, $\theta_i(h(y_2)) = \mathtt{d}$. Intuitively, this means that all $\mathtt{R}_0$-facts of $\mathbf{db}_0$ contain the constant $\mathtt{d}$ at the position at which $h(y_2)$ occurs in F_0. Note incidentally that this does not mean that y_2 occurs in G_0, because the homomorphism h can map distinct variables of $\mathcal{Q}_2$ to the same variable in $\mathcal{Q}_1$ (i.e., h needs not to be injective).

Let F be the atom such that $h(G) = F$, and let the relation name of F be $\mathtt{R}$. From $G \neq G_0$, it follows $F \neq F_0$ (and $\mathtt{R} \neq \mathtt{R}_0$). Since y_2 occurs in G, $h(y_2)$ occurs in F. So $h(y_2)$ occurs in both F_0 and F. Let o be the arity of $\mathtt{R}$ and let $p \in \{1, 2, \dots, o\}$ such that y_2 occurs at position p in G (and hence $h(y_2)$ occurs at position p in F). The construction of $\mathbf{db}_0$ ensures that all $\mathtt{R}$-facts of $\mathbf{db}_0$ will contain the same constant $\mathtt{d}$ at position p. Indeed, if an $\mathtt{R}$-fact A of $\mathbf{db}$ contains a distinct constant at position p, then the block containing A will be excluded from $\mathbf{db}_0$ (because of the *Minimality* condition). It follows $\alpha(y_2) = \mathtt{d} = \beta(y_2)$, a contradiction. We conclude by contradiction that $\gamma(G) \in \mathbf{r}$.

This concludes the proof. □

5.6 Conclusion

We have studied a realistic setting for divulging an inconsistent database to end users. In this setting, users access the database exclusively via syntactically restricted queries, and get exclusively consistent answers computable in **FO** data complexity. If the data complexity is higher, then the query will be rejected, in which case users have to fall back on strategies that obtain a large (the larger, the better) subset of the consistent answer. Such strategies combine answers obtained from several “easier” queries.

Although our setting applies to arbitrary queries and constraints, we only searched for strategies when constraints are primary keys, and the database is accessible only via self-join-free conjunctive queries for which consistent query answering is in **FO**. Under these access restrictions, we showed how to con-

struct strategies that combine answers by means of union and quantification. It turns out that the simplification of such strategies raises a novel and challenging query containment problem. By means of a new tool (a generalization of attack graphs), we were able to solve this containment problem under some syntactic restrictions, leaving a general solution for future work. Another interesting open question is whether our strategies can still be improved, e.g., by using negation.

Of practical interest is the development of an academic prototype that allows investigating the real-life applicability and efficiency of the proposed strategies.

Chapter 6

On the Syntax of Consistent First-Order Rewritings

If a conjunctive query $\mathcal{Q}$ has a consistent first-order rewriting, then there exist several procedures for effectively constructing a consistent first-order rewriting for $\mathcal{Q}$. Although different procedures construct semantically equivalent first-order queries, their outcomes may be syntactically very different. Some past experiments [DPW12, Dec13] have revealed that these syntactic differences can result in different execution times on a real SQL RDBMS, suggesting that some procedures yield "more optimal" rewritings than others.

In this chapter, we investigate two syntactic properties of consistent first-order rewritings, namely the quantifier rank and the number of existential and universal quantifier blocks. We elaborate rewriting procedures that aim at a reduction in either measure. This chapter generalizes the theoretical treatment of [DPW12,Dec13] which assumed that the conjunctive queries that are input to the rewriting procedures are acyclic (as defined in [BFMY83]). In this chapter, the input conjunctive queries can be cyclic. New experiments involving cyclic queries are left for future work. Past experiments are already presented in [Dec13] and will not be repeated here.

R	Conf	Year	Town
	SUM	2012	Marburg
	SUM	2016	Mons
	SUM	2016	Gent
	SUM	2017	Rome
	SUM	2017	Paris

S	Town	Attractiveness
	Charleroi	C
	Marburg	A
	Mons	A
	Mons	B
	Paris	A
	Rome	A

Figure 6.1: Uncertain database $\mathbf{db}_0$.

6.1 Introduction

In the following running example database, there are still two candidate cities for organizing SUM 2016 and SUM 2017. The table S shows controversy about the attractiveness of Mons, while information about the attractiveness of Gent is missing.

Database $\mathbf{db}_0$ has 8 repairs, because there are two choices for SUM 2016, two choices for SUM 2017, and two choices for Mons' attractiveness. The following conjunctive query asks in which years SUM took place (or will take place) in a city with A attractiveness:

$$\mathcal{Q}_0 = \left\{ y \mid \exists z \left(\mathtt{R}(\underline{\mathsf{SUM}, y}, z) \land \mathtt{S}(\underline{z}, \mathsf{A}) \right) \right\}.$$

The consistent answer of $\mathcal{Q}_0$ on $\mathbf{db}_0$ is $\{2012, 2017\}$. Notice incidentally that $\mathcal{Q}_0(\mathbf{db}_0)$ also contains 2016, but that answer is not certain, because in some repairs, the organizing city of SUM 2016 does not have A attractiveness.

For every database, the consistent answer to $\mathcal{Q}_0$ is obtained by the following first-order query:

$$\varphi_0(y) = \begin{array}{l} \exists z (\mathtt{R}(\underline{\mathsf{SUM}, y}, z) \land \\ \quad \forall z (\mathtt{R}(\underline{\mathsf{SUM}, y}, z) \Rightarrow \\ \qquad \mathtt{S}(\underline{z}, \mathsf{A}) \land \\ \qquad\quad \forall v (\mathtt{S}(\underline{z}, v) \Rightarrow v = \mathsf{A}))). \end{array}$$

Consistent first-order rewritings are of practical importance, because they can be encoded in SQL, which allows to obtain consistent answers using standard database technology. We know by Theorem 3 that such rewritings can be computed for self-join-free conjunctive queries with an acyclic attack graph, which we focus on in this chapter.

We provide two theorems indicating that rewritings produced by Function Rewrite can generally be "simplified" by (i) reducing the number of (alternations of) quantifier blocks and/or by (ii) reducing the quantifier nesting depth.

This chapter is organized as follows. Section 6.2 provides missing notations and definitions. In particular, we provide measures for describing the syntactic complexity of a first-order formula. Section 6.3 provides an example justifying why the algorithm developed in Chapter 3 could (and therefore should) be improved. Section 6.4 and Section 6.5 show how rewritings can be simplified with respect to the complexity measures of Section 6.2. Section 6.6 concludes the chapter.

6.2 Notations and Terminology

Quantifier rank and quantifier alternation depth The *quantifier rank* of a first-order formula φ, denoted by $\mathbf{qr}(\varphi)$, is the depth of the quantifier nesting in φ and is defined as usual (see, for example, [Lib04, page 32]):

- if φ is quantifier-free, then $\mathbf{qr}(\varphi) = 0$;
- $\mathbf{qr}(\varphi_1 \wedge \varphi_2) = \mathbf{qr}(\varphi_1 \vee \varphi_2) = \max(\mathbf{qr}(\varphi_1), \mathbf{qr}(\varphi_2))$;
- $\mathbf{qr}(\neg\varphi) = \mathbf{qr}(\varphi)$;
- $\mathbf{qr}(\exists x\,(\varphi)) = \mathbf{qr}(\forall x\,(\varphi)) = 1 + \mathbf{qr}(\varphi)$.

A first-order formula φ is said to be in *prenex normal form* if it has the form $q_1x_1q_2x_2\dots q_nx_n\psi$, where q_i's are either $\exists$ or $\forall$ and ψ is quantifier-free. We say that φ has *quantifier alternation depth m* if $q_1x_1q_2x_2\dots q_nx_n$ can be

divided into m blocks such that all quantifiers in a block are of the same type and quantifiers in two consecutive blocks are different.

For formulas not in prenex normal form, the number of quantifier blocks is counted as follows. A universally quantified formula is a formula whose main connective is $\forall$. An existentially quantified formula is a formula whose main connective is $\exists$. The number of quantifier blocks in a first-order formula φ, denoted $\mathbf{qbn}(\varphi)$, is defined as follows:

- if φ is quantifier-free, then $\mathbf{qbn}(\varphi) = 0$;
- $\mathbf{qbn}(\varphi_1 \wedge \varphi_2) = \mathbf{qbn}(\varphi_1 \vee \varphi_2) = \mathbf{qbn}(\varphi_1) + \mathbf{qbn}(\varphi_2)$;
- $\mathbf{qbn}(\neg\varphi) = \mathbf{qbn}(\varphi)$;
- if φ is not universally quantified and $n \geq 1$, then $\mathbf{qbn}(\forall x_1 \forall x_2 \ldots \forall x_n (\varphi)) = 1 + \mathbf{qbn}(\varphi)$; and
- if φ is not existentially quantified and $n \geq 1$, then $\mathbf{qbn}(\exists x_1 \exists x_2 \ldots \exists x_n (\varphi)) = 1 + \mathbf{qbn}(\varphi)$.

For example, if φ is $\exists x \exists y (\exists u (\varphi_1) \wedge \exists v (\varphi_2))$ and φ_1, φ_2 are both quantifier-free, then $\mathbf{qbn}(\varphi) = 3$. Notice that φ has a prenex normal form with quantifier alternation depth equal to 1. Clearly, if φ is in prenex normal form, then the quantifier alternation depth of φ is equal to $\mathbf{qbn}(\varphi)$.

Proposition 2. *Every first-order formula φ has an equivalent one in prenex normal form with quantifier alternation depth less than or equal to $\mathbf{qbn}(\varphi)$.*

Proof. The proof runs by induction on the structure of φ. The result is obvious if φ is quantifier free. For the induction step, we distinguish the following cases.

Case $\varphi = \varphi_1 \wedge \varphi_2$. Assume $\mathbf{qbn}(\varphi_1) = n_1$ and $\mathbf{qbn}(\varphi_2) = n_2$. By the induction hypothesis, we can assume integers $m_1 \leq n_1$ and $m_2 \leq n_2$ such that φ_1 has an equivalent formula φ_1' in prenex normal form with quantifier alternation depth m_1, and φ_2 has an an equivalent formula φ_2' in prenex normal form with quantifier alternation depth m_2. A standard

translation [Pap94, page 99] of $\varphi_1' \wedge \varphi_2'$ in prenex normal form yields a formula with quantifier alternation depth $\leq m_1 + m_2$. Finally, notice $m_1 + m_2 \leq n_1 + n_2 = \mathbf{qbn}(\varphi)$.

Case $\varphi = \varphi_1 \vee \varphi_2$. Similar to the previous case.

Case $\varphi = \neg\varphi_1$. Easy.

Case $\varphi = \forall x_1 \dots \forall x_n \varphi_1$ when $n \geq 1$ and φ_1 not universally quantified. Assume $\mathbf{qbn}(\varphi_1) = n_1$. By the induction hypothesis, we can assume integer $m_1 \leq n_1$ such that φ_1 has an equivalent formula φ_1' in prenex normal form with quantifier alternation depth m_1. Obviously, $\forall x_1 \dots \forall x_n \varphi_1'$ is equivalent to φ, is in prenex normal form, and has quantifier alternation depth $\leq 1 + m_1$. Finally, notice $1 + m_1 \leq 1 + n_1 = \mathbf{qbn}(\varphi)$.

Case $\varphi = \exists x_1 \dots \exists x_n \varphi_1$ when $n \geq 1$ and φ_1 not existentially quantified. Analogous to the previous case.

□

6.3 Naive Algorithm

The algorithm Function Rewrite developped in Chapter 3 is called “naive” because it does not attempt to minimize the alternations or nesting depth of quantifiers. This may be problematic when the rewritings are translated into SQL for execution, as illustrated by the following example.

Example 35. *For each $m \geq 1$, assume relation name $\mathtt{R}_i$ with signature $[2, 1]$, and let $\mathbf{disjoint}(m) = \{\mathtt{R_1}(\underline{x_1}, \mathtt{b}), \dots, \mathtt{R_m}(\underline{x_m}, \mathtt{b})\}$, where $\mathtt{b}$ is a constant. Notice that $\mathbf{disjoint}(m)$ is a Boolean query whose attack graph has no edges. Formulas φ_1, φ_2, and φ_3 are three possible consistent first-order rewritings for $\mathbf{disjoint}(m)$. The formula φ_1 is returned by Function Rewrite, while φ_2 and φ_3 result from some syntactic simplification techniques described in Section 6.4 and Section 6.5. In particular, φ_2 minimizes the number of quantifier blocks, and φ_3 minimizes the nesting depth of quantifiers.*

$$
\begin{aligned}
\varphi_1 = \quad & \exists x_1(\mathtt{R_1}(\underline{x_1}, \mathsf{b}) \wedge \\
& \quad \forall z_1(\mathtt{R_1}(\underline{x_1}, z_1) \Rightarrow z_1 = \mathsf{b} \wedge \\
& \quad\quad \exists x_2(\mathtt{R_2}(\underline{x_2}, \mathsf{b}) \wedge \\
& \quad\quad\quad \forall z_2(\mathtt{R_2}(\underline{x_2}, z_2) \Rightarrow z_2 = \mathsf{b} \wedge \\
& \quad\quad\quad\quad \ldots \\
& \quad\quad\quad\quad\quad \exists x_m(\mathtt{R_m}(\underline{x_m}, \mathsf{b}) \wedge \\
& \quad\quad\quad\quad\quad\quad \forall z_m(\mathtt{R_m}(\underline{x_m}, z_m) \Rightarrow z_m = \mathsf{b})) \ldots))))
\end{aligned}
$$

$$
\begin{aligned}
\varphi_2 = \quad & \exists x_1 \exists x_2 \ldots \exists x_m (\textstyle\bigwedge_{i=1}^{m} \mathtt{R_i}(\underline{x_i}, \mathsf{b}) \wedge \\
& \quad \forall z_1 \forall z_2 \ldots \forall z_m (\textstyle\bigwedge_{i=1}^{m} \mathtt{R_i}(\underline{x_i}, z_i) \Rightarrow \\
& \quad\quad z_1 = \mathsf{b} \wedge z_2 = \mathsf{b} \wedge \ldots \wedge z_m = \mathsf{b}))
\end{aligned}
$$

$$
\begin{aligned}
\varphi_3 = \quad & \exists x_1(\mathtt{R_1}(\underline{x_1}, \mathsf{b}) \wedge \forall z_1(\mathtt{R_1}(\underline{x_1}, z_1) \Rightarrow z_1 = \mathsf{b})) \wedge \\
& \exists x_2(\mathtt{R_2}(\underline{x_2}, \mathsf{b}) \wedge \forall z_2(\mathtt{R_2}(\underline{x_2}, z_2) \Rightarrow z_2 = \mathsf{b})) \wedge \\
& \quad \vdots \\
& \exists x_m(\mathtt{R_m}(\underline{x_m}, \mathsf{b}) \wedge \forall z_m(\mathtt{R_m}(\underline{x_m}, z_m) \Rightarrow z_m = \mathsf{b}))
\end{aligned}
$$

Notice that φ_1, φ_2, and φ_3 each contain m existential and m universal quantifiers. The following table gives the quantifier rank and the number of quantifier blocks for these formulas; recall that these measures were defined in Section 6.2.

φ	$\mathbf{qr}(\varphi)$	$\mathbf{qbn}(\varphi)$
φ_1	$2m$	$2m$
φ_2	$2m$	2
φ_3	2	$2m$

*The differences in syntactic complexity persist in SQL. Assume that for each $i \in \{1, 2, \ldots, m\}$, the first and the second attribute of each $\mathtt{R}_i$ are named **A** and **B** respectively. Thus, **A** is the primary key attribute. For $m = 2$, the queries of Listing 6.1, Listing 6.2 and Listing 6.3 are direct translations into*

```sql
SELECT 'true' FROM R1 AS r11
WHERE NOT EXISTS (
  SELECT * FROM R1 AS r12
  WHERE r12.A = r11.A AND (r12.B <> 'b' OR NOT EXISTS (
    SELECT * FROM R2 AS r21
    WHERE NOT EXISTS (
      SELECT * FROM R2 AS r22
      WHERE r22.A = r21.A AND r22.B <> 'b'))));
```

Listing 6.1: Consistent SQL rewriting obtained from φ_1.

```sql
SELECT 'true' FROM R1 AS r11, R2 AS r21
WHERE NOT EXISTS (
  SELECT * FROM R1 AS r12, R2 AS r22
  WHERE r12.A = r11.A AND r22.A = r21.A
  AND   (r12.B <> 'b' OR r22.B <> 'b'));
```

Listing 6.2: Consistent SQL rewriting obtained from φ_2.

SQL of φ_1, φ_2, and φ_3. [1] *The fact that φ_2 only has one $\forall$ quantifier block results in an SQL query for φ_2 with one* `NOT EXISTS`*. Notice further that the SQL query for φ_2 requires m tables in each* `FROM` *clause, whereas the SQL query for φ_3 takes the intersection of m SQL queries, each with a single table in the* `FROM` *clause.*

The foregoing example shows that formulas returned by Function Rewrite can be "optimized" so as to have lower quantifier rank and/or less (alternations of) quantifiers blocks. The theoretical details will be given in the next two sections.

[1] In practice, we construct rewritings in tuple relational calculus (TRC), and then translate TRC into SQL. Such translations are well known (see, e.g., Chapter 3 of [Ull88]).

```
SELECT 'true' FROM R1 AS r11
WHERE  NOT EXISTS (
  SELECT * FROM R1 AS r12
  WHERE  r12.A = r11.A AND r12.B <> 'b')
INTERSECT
SELECT 'true' FROM R2 AS r21
WHERE NOT EXISTS (
  SELECT * FROM R2 AS r22
  WHERE  r22.A = r21.A AND r22.B <> 'b');
```

Listing 6.3: Consistent SQL rewritting obtained from φ_3.

Importantly, these simplifications do not decrease (nor increase) the number of $\exists$ or $\forall$ quantifiers in a formula; they merely group quantifiers of the same type in blocks and/or decrease the nesting depth of quantifiers.

6.4 Reducing the Number of Quantifier Blocks

Function Rewrite constructs a consistent first-order rewriting by treating one unattacked atom at a time. The next theorem implies that multiple unattacked atoms can be "rewritten" together, which generally results in less (alternations of) quantifier blocks, as expressed by Corollary 1.

Theorem 15. *Let $\mathcal{Q}(\vec{v})$ be a query in* ***SJFCQ****. Let $S \subseteq \mathcal{Q}$ be a set of unattacked atoms in the attack graph of $\mathcal{Q}$. Let $\vec{x}$ be a vector of variables such that $\mathbf{vars}(\vec{x}) = \bigcup_{F \in S} \mathbf{keyvars}(F) \setminus \mathbf{vars}(\vec{v})$. Let $\mathcal{Q}'(\vec{v}, \vec{x})$ be the query with the same atoms as $\mathcal{Q}(\vec{v})$ but in which the variables of $\vec{x}$ are free. If $\varphi(\vec{v}, \vec{x})$ is a consistent first-order rewriting for $\mathcal{Q}'(\vec{v}, \vec{x})$, then $\exists \vec{x} \varphi(\vec{v}, \vec{x})$ is a consistent first-order rewriting for $\mathcal{Q}(\vec{v})$.*

Proof. We fix $\vec{v} = v_1, \ldots, v_m$ and $\vec{x} = x_1, \ldots, x_l$. Assume φ is a consistent first-order rewriting for $\mathcal{Q}'$. Then, for every database $\mathbf{db}$, for all $\vec{a} \in \mathbf{dom}^m$,

$\vec{b} \in \mathbf{dom}^l$,

$$\mathbf{db} \models \left\lfloor \mathcal{Q}'(\vec{a}, \vec{b}) \right\rfloor \Leftrightarrow \mathbf{db} \models \varphi(\vec{a}, \vec{b}). \tag{6.1}$$

Note that $\mathbf{db} \models \left\lfloor \mathcal{Q}'(\vec{a}, \vec{b}) \right\rfloor$ is tantamount to saying that every repair of $\mathbf{db}$ satisfies the Boolean query $\mathcal{Q}'(\vec{a}, \vec{b})$. We need to show that for every database $\mathbf{db}$, for every $\vec{a} \in \mathbf{dom}^m$,

$$\mathbf{db} \models \lfloor \mathcal{Q}(\vec{a}) \rfloor \Leftrightarrow \mathbf{db} \models \exists \vec{x} \varphi(\vec{a}, \vec{x}). \tag{6.2}$$

$\boxed{\Leftarrow}$ Easy. $\boxed{\Rightarrow}$ Assume $S = \{F_1, \ldots, F_u\}$. Let $\vec{x}_1 = x_1, \ldots, x_k$ be a vector of variables such that $\mathbf{vars}(\vec{x}_1) = \mathbf{keyvars}(F_1) \setminus \mathbf{keyvars}(\vec{v})$. Let $\mathcal{Q}^1 = \mathcal{Q}(\vec{v}, \vec{x}_1)$, i.e., $\mathcal{Q}^1$ is obtained from $\mathcal{Q}$ by making the variables in $\vec{x}_1$ free. By Lemma 3.7 in [KW17], for every database $\mathbf{db}$, for every $\vec{a} \in \mathbf{dom}^m$, there exists $\vec{b} \in \mathbf{dom}^k$ such that:

$$\mathbf{db} \models \lfloor \mathcal{Q}(\vec{a}) \rfloor \Rightarrow \mathbf{db} \models \left\lfloor \mathcal{Q}^1(\vec{a}, \vec{b}) \right\rfloor.$$

It can be easily shown that none of the atoms of S is attacked in the attack graph of $\mathcal{Q}^1$ (still by Lemma 3.7 in [KW17]). Then, by repeated application of the same arguments, for every database $\mathbf{db}$, for every $\vec{a} \in \mathbf{dom}^m$, there exists $\vec{b} \in \mathbf{dom}^l$ such that:

$$\mathbf{db} \models \lfloor \mathcal{Q}(\vec{a}) \rfloor \Rightarrow \mathbf{db} \models \left\lfloor \mathcal{Q}'(\vec{a}, \vec{b}) \right\rfloor. \tag{6.3}$$

From (6.1) and (6.3), for every database $\mathbf{db}$, for every $\vec{a} \in \mathbf{dom}^m$, there exists $\vec{b} \in \mathbf{dom}^l$ such that $\mathbf{db} \models \lfloor \mathcal{Q}(\vec{a}) \rfloor$ implies $\mathbf{db} \models \varphi(\vec{a}, \vec{b})$. Consequently, for every database $\mathbf{db}$, for every $\vec{a} \in \mathbf{dom}^m$, $\mathbf{db} \models \lfloor \mathcal{Q}(\vec{a}) \rfloor$ implies $\mathbf{db} \models \exists \vec{x} \varphi(\vec{a}, \vec{x})$. □

Corollary 1. *Let $\mathcal{Q}(\vec{v})$ be a query in **SJFCQ** whose attack graph is acyclic. Let p be the number of atoms on the longest directed path in the attack graph of $\mathcal{Q}$. There exists a consistent first-order rewriting φ for $\mathcal{Q}$ such that $\mathbf{qbn}(\varphi) \leq 2p$.*

Proof. For every atom $F \in \mathcal{Q}$, we define the *stratum* of F as the number of atoms on the longest directed path (in the attack graph of $\mathcal{Q}$) that starts from

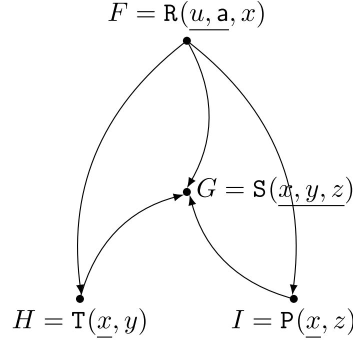

Figure 6.2: Attack graph for Boolean query $\mathcal{Q} = \{\mathtt{R}(\underline{u, \mathtt{a}}, x), \mathtt{S}(\underline{x, y, z}), \mathtt{T}(\underline{x}, y), \mathtt{P}(\underline{x}, z)\}$.

some unattacked atom and ends at F. In particular, an atom has stratum 1 if and only if it is unattacked. The greatest stratum is p.

From Theorem 15, it follows that Function Rewrite can be modified so as to proceed stratum-by-stratum (rather than atom-by-atom), starting with all atoms at stratum 1. In particular, let $\{\mathtt{R_i}(\underline{\vec{x_i}}, \vec{y_i})\}_{i=1}^{k}$ be all atoms at stratum 1. Then $\mathcal{Q}$ has a consistent first-order rewriting of the form:

$$\exists \vec{x}_1 \cdots \exists \vec{x}_k \exists \vec{y}_1 \cdots \exists \vec{y}_k \left(\begin{array}{l} \left(\bigwedge_{i=1}^{k} \mathtt{R_i}(\underline{\vec{x_i}}, \vec{y_i})\right) \wedge \\ \forall \vec{y}_1 \cdots \forall \vec{y}_k \left(\left(\bigwedge_{i=1}^{k} \mathtt{R_i}(\underline{\vec{x_i}}, \vec{y_i})\right) \Rightarrow C \wedge \psi\right) \end{array} \right)$$

where C is a conjunction of equalities and ψ is a consistent first-order rewriting of the query obtained from $\mathcal{Q}$ by removing all atoms with stratum 1. If ψ is obtained by recursive application of the same method (on a smaller query), we obtain a consistent first-order rewriting for $\mathcal{Q}$ that has a prenex normal form with $2p$ quantifier blocks. □

Example 36. *The longest path in the attack graph of Figure 6.2 contains* 3 *atoms. From Corollary 1 and Property 2, it follows that the query of Figure 6.2 has a consistent first-order rewriting with quantifier alternation depth less than or equal to* 6*.*

6.5 Reducing the Quantifier Rank

Consider query **disjoint**(m) with $m \geq 1$ in Example 35. Function Rewrite will "rewrite" the atoms $\mathtt{R_i}(\underline{x_i}, \mathtt{b})$ sequentially $(1 \leq i \leq m)$. However, since these atoms have no bound variables in common, it is correct to rewrite them "in parallel" and then join the resulting formulas. This idea is generalized in the following theorem.

Definition 10. *Let $\mathcal{Q}(\vec{v})$ be a query in **SJFCQ** with $|\mathcal{Q}| \geq 1$. An* independent partition *of $\mathcal{Q}$ is a (complete disjoint) partition $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_k\}$ of $\mathcal{Q}$ such that for $1 \leq i < j \leq k$, $\mathbf{vars}(\mathcal{Q}_i) \cap \mathbf{vars}(\mathcal{Q}_j) \subseteq \mathbf{vars}(\vec{v})$.*

Theorem 16. *Let $\mathcal{Q}(\vec{v})$ be a query in **SJFCQ**. Let $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_k\}$ be an independent partition of $\mathcal{Q}$. For each $1 \leq i \leq k$, let φ_i be a consistent first-order rewriting for $\mathcal{Q}_i(\vec{v_i})$, where $\vec{v_i}$ is $\vec{v}$ restricted to $\mathbf{vars}(\mathcal{Q}_i)$. Then, $\bigwedge_{i=1}^{k} \varphi_i$ is a consistent first-order rewriting for $\mathcal{Q}$.*

Proof. Let θ be an arbitrary valuation over $\mathbf{vars}(\vec{v})$. For $1 \leq i \leq k$, let θ_i be the restriction of θ on $\mathbf{vars}(\vec{v_i})$. Define $\varphi = \bigwedge_{i=1}^{k} \varphi_i$.

Let $\mathbf{db}$ be a database such that $\mathbf{db} \models \varphi(\theta(\vec{v}))$. Then, for $1 \leq i \leq k$, $\mathbf{db} \models \varphi_i(\theta_i(\vec{v_i}))$. Let $\mathbf{r}$ be an arbitrary repair of $\mathbf{db}$. Since each φ_i is a consistent first-order rewriting for $\mathcal{Q}_i$ and since $\mathbf{db} \models \varphi(\theta_i(\vec{v_i}))$, we have $\theta_i(\vec{v_i}) \in \mathcal{Q}_i(\mathbf{r})$ (for $1 \leq i \leq k$). Hence, for every $1 \leq i \leq k$, we can extend θ_i to a valuation Θ_i over $\mathbf{vars}(\mathcal{Q}_i)$ such that $\Theta_i(\mathcal{Q}_i) \subseteq \mathbf{r}$. Let $\Theta = \bigcup_{i=1}^{k} \Theta_i$. We show that Θ is a function. Assume that the same variable x occurs in $\mathbf{vars}(\mathcal{Q}_i)$ and $\mathbf{vars}(\mathcal{Q}_j)$ with $i \neq j$. Since $\{\mathcal{Q}_1, \ldots, \mathcal{Q}_k\}$ is an independent partition of $\mathcal{Q}$, we have $x \in \mathbf{vars}(\vec{v})$, hence $x \in \mathbf{vars}(\vec{v_i})$ and $x \in \mathbf{vars}(\vec{v_j})$. It follows that $\Theta_i(x) = \theta(x)$ and $\Theta_j(x) = \theta(x)$, hence $\Theta_i(x) = \Theta_j(x)$. Since $\Theta(\mathcal{Q}) \subseteq \mathbf{r}$ is now obvious, we have $\Theta(\vec{v}) = \theta(\vec{v}) \in \mathcal{Q}(\mathbf{r})$. Since $\mathbf{r}$ is an arbitrary repair of $\mathbf{db}$, it follows that every repair of $\mathbf{db}$ satisfies $\mathcal{Q}(\theta(\vec{v}))$.

Conversely, assume that every repair of $\mathbf{db}$ satisfies $\mathcal{Q}(\theta(\vec{v}))$. Let $\mathbf{r}$ be a repair of $\mathbf{db}$. We have $\theta(\vec{v}) \in \mathcal{Q}(\mathbf{r})$. Hence, we can extend θ to a valuation Θ over $\mathbf{vars}(\mathcal{Q})$ such that $\Theta(\mathcal{Q}) \subseteq \mathbf{r}$. For $1 \leq i \leq k$, let Θ_i be the restriction of Θ

on $\mathbf{vars}(\mathcal{Q}_i)$. Then for $1 \leq i \leq k$, $\Theta_i(\mathcal{Q}_i) \subseteq \mathbf{r}$. It follows $\Theta_i(\vec{v}_i) \in \mathcal{Q}_i(\mathbf{r})$. Since $\Theta(\vec{v}_i) = \theta_i(\vec{v}_i)$ is obvious and since $\mathbf{r}$ is an arbitrary repair of $\mathbf{db}$, it follows that for $1 \leq i \leq k$, every repair of $\mathbf{db}$ satisfies $\mathcal{Q}_i(\theta_i(\vec{v}_i))$. Since each φ_i is a consistent first-order rewriting for $\mathcal{Q}_i$, we have that for $1 \leq i \leq k$, $\mathbf{db} \models \varphi_i(\theta_i(\vec{v}_i))$. Consequently, $\mathbf{db} \models \bigwedge_{i=1}^{k} \varphi_i(\theta_i(\vec{v}_i))$. It follows $\mathbf{db} \models \varphi(\theta(\vec{v}))$. □

This theorem allows us to build queries with a lower quantifier rank than the naive algorithm, but we would like to predict the final quantifier rank we can achieve using it. To this extent, we provide Function QR which computes an upper bound on the smallest quantifier rank one can obtain by improving the naive algorithm using Theorem 16. That is, if Function QR returns the integer n on input $\mathcal{Q}(\vec{v})$, then $\mathcal{Q}(\vec{v})$ has a consistent first-order rewriting φ with $\mathbf{qr}(\varphi) \leq n$.

The expression $|\mathbf{keyvars}(F_i) \setminus \mathbf{vars}(\vec{v})| + (n-k)$ in Function QRaux is the increase in quantifier rank that results from rewriting an atom F_i. For example, if we rewrite an atom $\mathtt{R}(\underline{x_1,\ldots,x_k}, x_{k+1},\ldots,x_n)$ and assume no free variables (i.e., $\vec{v}$ is empty), the increase in quantifier rank is $k + (n-k) = n$, which can be verified on the following rewriting:

$$\exists x_1 \cdots \exists x_k \left(\begin{array}{l} \exists x_{k+1} \cdots \exists x_n \mathtt{R}(\underline{x_1,\ldots,x_k}, x_{k+1},\ldots,x_n) \wedge \\ \forall x_{k+1} \cdots \forall x_n \left(\mathtt{R}(\underline{x_1,\ldots,x_k}, x_{k+1},\ldots,x_n) \Rightarrow \varphi \right) \end{array} \right).$$

In the above rewriting, the formula φ, which rewrites the remaining atoms, occurs within n nested quantifiers. Significantly, many queries have a consistent first-order rewriting whose quantifier rank is strictly smaller than the upper bound given by Function QR. For example, in the following "smart" rewriting of the atom $\mathtt{R}(\underline{x}, y, y, y)$, the formula φ occurs within only two nested quantifiers, while $n = 4$:

$$\exists x \left(\begin{array}{ll} & \exists y \mathtt{R}(\underline{x}, y, y, y) \wedge \\ \wedge & \forall y \forall y' \forall y'' \left(\mathtt{R}(\underline{x}, y, y', y'') \Rightarrow y = y' = y'' \right) \\ \wedge & \forall y \left(\mathtt{R}(\underline{x}, y, y, y) \Rightarrow \varphi \right) \end{array} \right).$$

```
Input: Q(v⃗) is a query in SJFCQ whose attack graph is acyclic.
Result: An upper bound on the smallest quantifier rank of the
        consistent first-order rewriting constructed according to
        Theorem 16.
if Q = ∅ then
  | return 0;
else
  | let Q_1(v⃗_1), ..., Q_n(v⃗_n) be the independent subqueries of Q(v⃗);
  | return max_{1≤i≤n} QRaux(Q_i(v⃗_i));
```

Function QR computes the best quantifier rank.

```
let F_1, ..., F_n be the unattacked atoms of Q(v⃗);
foreach i ∈ {1, 2, ..., n} do
  | let v⃗_i be a vector of the variables in vars(v⃗) ∪ vars(F_i);
  | Q_i ← Q \ {F_i};
  | let [n, k] be the signature of F_i;
  | N_i ← |keyvars(F_i) \ vars(v⃗)| + (n − k);
return min_{1≤i≤n}(QR(Q_i(v⃗_i)) + N_i);
```

Function QRaux is an auxiliary function for $QR(\mathcal{Q}(\vec{v}))$.

Note that in Function QRaux, taking the minimum of the given measures is useful, as shown in the next two examples.

Example 37. *Consider the Boolean query* $\mathcal{Q} = \{\mathtt{R}(\underline{x,y}), \mathtt{S}(\underline{x},\mathtt{a}), \mathtt{T}(\underline{y},\mathtt{a})\}$. *All atoms are unattacked. If we start rewriting the* R*-atom, then there remains no variable in the subquery, and the* S*- and* T*-atoms can be handled in a parallel fashion. But if we end with the* R*-atom, then all the atoms have to be handled sequentially, leading to a difference in the quantifier rank resulting from the two strategies.*

Starting with R

$$\begin{aligned}
&\exists x\,\exists y(\mathtt{R}(\underline{x,y})\wedge\\
&\quad(\mathtt{S}(\underline{x},\mathtt{a})\wedge\\
&\qquad\forall z(\mathtt{S}(\underline{x},z)\Rightarrow z=\mathtt{a}))\wedge\\
&\quad(\mathtt{T}(\underline{y},\mathtt{a})\wedge\\
&\qquad\forall z(\mathtt{T}(\underline{y},z)\Rightarrow z=\mathtt{a})))
\end{aligned}$$

Quantifier rank is 3.

Ending with R (starting with the S-atom)

$$\begin{aligned}
&\exists x(\mathtt{S}(\underline{x},\mathtt{a})\wedge\\
&\quad\forall v(\mathtt{S}(\underline{x},v)\Rightarrow v=\mathtt{a}\wedge\\
&\qquad\exists y(\mathtt{T}(\underline{y},\mathtt{a})\wedge\\
&\qquad\quad\forall w(\mathtt{T}(\underline{y},w)\Rightarrow w=\mathtt{a}\wedge\\
&\qquad\qquad\mathtt{R}(\underline{x,y})))))
\end{aligned}$$

Quantifier rank is 4.

Example 38. *We can extend the previous example in the following way. Let* $\mathcal{Q}$ *be the following query in* ***SJFCQ***:

$$\mathbf{sm}(n) = \left\{\mathtt{R}(\underline{x_1,x_2,\ldots,x_n})\right\} \cup \left\{\mathtt{R_i}(\underline{x_i},\mathtt{a}_1,\mathtt{a}_2,\ldots,\mathtt{a}_{n-1}) \mid i\in\{1,2,\ldots,n\}\right\}$$

where n *is any non-negative integer. Note that* $\mathbf{sm}(2)$ *is the query from the previous example modulo relation names and constant* a *which becomes* $\mathtt{a}_1$.

Again we consider what happens when we start rewriting with the $\mathtt{R}$*-atom and when we end rewriting with this atom. In the second case, without loss of generality, we handle the atoms* $\mathtt{R}_1, \mathtt{R}_2, \ldots, \mathtt{R}_n$ *in natural order.*

Starting with $\mathtt{R}$

$$
\begin{array}{l}
\exists x_1\, \exists x_2\, \ldots\, \exists x_n (\mathtt{R}(\underline{x_1, x_2, \ldots, x_n}) \wedge \\
\quad (\mathtt{R_1}(\underline{x_1}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\qquad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_1}(\underline{x_1}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad\quad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1}))) \wedge \\
\quad (\mathtt{R_2}(\underline{x_2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\qquad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_2}(\underline{x_2}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad\quad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1}))) \wedge \\
\qquad \vdots \\
\quad (\mathtt{R_n}(\underline{x_n}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\qquad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_n}(\underline{x_n}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad\quad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1}))))
\end{array}
$$

Quantifier rank is $n+(n-1) = 2n-1$ *where* n *is the number of existential quantifiers and* $n-1$ *the number of universal quantifiers in each "branch."*

Ending with $\mathtt{R}$

$$
\begin{array}{l}
\exists x_1 (\mathtt{R_1}(\underline{x_1}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\quad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_1}(\underline{x_1}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1} \wedge \\
\qquad\quad \exists x_2 (\mathtt{R_2}(\underline{x_2}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\qquad\qquad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_2}(\underline{x_2}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad\qquad\quad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1} \wedge \\
\qquad\qquad\qquad \ddots \\
\qquad\qquad\qquad\quad \exists x_n (\mathtt{R_n}(\underline{x_n}, \mathsf{a}_1, \mathsf{a}_2, \ldots, \mathsf{a}_{n-1}) \wedge \\
\qquad\qquad\qquad\qquad \forall y_1\, \forall y_2\, \ldots\, \forall y_{n-1} (\mathtt{R_n}(\underline{x_n}, y_1, y_2, \ldots, y_{n-1}) \Rightarrow \\
\qquad\qquad\qquad\qquad\quad (y_1 = \mathsf{a}_1 \wedge y_2 = \mathsf{a}_2 \wedge \ldots \wedge y_{n-1} = \mathsf{a}_{n-1} \wedge \\
\qquad\qquad\qquad\qquad\qquad \mathtt{R}(\underline{x_1, x_2, \ldots, x_n})))) \ldots))))))
\end{array}
$$

> *Quantifier rank in* $n + (n * (n - 1)) = n^2$ *where* n *is the number of existential quantifiers and* $n - 1$ *the number of universal quantifiers in each "step" of the query (and there are* n *such steps).*

This example shows that choosing the rewrite order carefully can lead to significant reductions in the quantifier rank of the generated rewriting. Function QR shows us how to choose the best possible strategy.

6.6 Conclusion

Corollary 1 and Function QR show upper bounds on the number of quantifier blocks or the quantifier rank of consistent first-order rewritings. Function Rewrite can be easily modified so as to diminish either of those measures. It is not generally possible to minimize both measures simultaneously. For example, there seems to be no consistent first-order rewriting φ for **disjoint**(m) such that $\mathbf{qbn}(\varphi) = \mathbf{qr}(\varphi) = 2$ (cf. the table in Section 6.3).

We focused on the queries in **SJFCQ** that have a consistent first-order rewriting. We first implemented this earlier theory in a simple Function Rewrite for constructing consistent first-order rewritings. We then proposed two syntactic simplifications for such rewritings, which consist in reducing the number of quantifier blocks and reducing the quantifier rank.

Chapter 7

Tools

A number of software tools have been developed that implement the theory developed in the preceding chapters of this thesis. These tools offer a variety of functionality such as computing and visualizing attack graphs, computing consistent rewritings in first-order logic and SQL, in different syntactic forms (see Chapter 6), computing first-order under-approximations (see Chapter 5)... These tools have already proved their usefulness in research. For example, some conjectured properties of attack graphs can now be empirically tested on large numbers of queries, a process that is time-consuming and error-prone when executed manually. The tools are implemented using the OCaml language, which is ideally fitted for scientific and mathematical software due to its cleanliness and proximity to the mathematical language.

7.1 Canswer

Canswer (C stands for Consistent) is an OCaml library handling the constructs we use in our research, such as first-order formulas and attack graphs. It also allows exporting these structures into practical languages (SQL for first-order queries, dot and tikz for attack graphs). The main functionality of Canswer is to compute the attack graph of a self-join-free conjunctive query, to produce a consistent first-order rewriting if it exists, and to translate such rewriting in

SQL.

Canswer also includes the optimizations discussed in Chapter 6; it can deal with the presence of satisfied constraints as discussed in Chapter 4, and can produce the under-approximations established in Chapter 5.

Canswer is a library. As such it is meant to be used by other programs developed in OCaml. This section presents the interface and the capabilities of Canswer. The next sections present two interfaces that can be used by (human) end-users. Both of these interfaces (one is console-oriented, the other web-oriented) are built upon Canswer, featuring a subset of its functionalities.

For a full understanding of Canswer, the reader is advised to read the mli source files of Canswer, which contain the interface and the documentation of Canswer. However, reading this section should help to understand how the library is built and where to start in order to get a broad idea of what Canswer is able to do.

The Canswer library is distributed under LGPL license. This essentially means that one can copy it, alter it, distribute it, and even sell it, provided that one includes the original work, license and copyright, and that one discloses the source and states any changes made to Canswer. Applications using Canswer do not have to be licensed under LGPL.

7.1.1 Core

This part of Canswer defines the basic structures used in the whole library as well as some basic operations. We list the modules and give some important pieces of the interface with some explanations when appropriate.

Lset This module handles sets under the form of a list. It defines the `'a Lset.t` type (that is, a set containing elements of any type `'a`, e.g., symbols), and functions for set creation, set inspection, set operations, as well as the usual functionals.

Symbol A symbol is a sum type defined as follows.

```
type constant = string
type variable = string
type t =
  | Constant of constant
  | Variable of variable
```

A function filtering variables from a `Symbol.t Lset.t` is also given.

RelationName

```
type name = string
type attribute = string
type t = private {
  name : name;
  attributes : attribute Lset.t;
  key : attribute -> bool;
}
```

The type `RelationName.t` is private to ensure that the set of attributes uses a correct equality function. The key part of a relation name is a Boolean function telling whether an attribute is part of the key or not. The function `RelationName.make` allows one to create relation names. Other functions help the user to create standard relation names.

Substitution This module captures the notion of substitution. Creating new substitutions has to be done using the `+` and `*` operators; more information can be found in the documentation. One can check whether a substitution is a valuation by means of `Substitution.is_valuation`. Most of the other modules contain a function `substitute` that applies a substitution to its main data structure (e.g., to an atom or a query).

Atom An atom is a relation name along with a function mapping attribute names (contained in the relation name) to symbols.

```
type t = {
  relation : RelationName.t;
  f : RelationName.attribute -> Symbol.t;
}
```

Conjunctive A conjunctive query is a set of atoms along with a set of free variables.

```
type t = private {
  atoms : Atom.t Lset.t;
  free : Symbol.t Lset.t;
}
```

The type is private to ensure that the sets used use a correct equality function and that the free variables actually occur in the atoms. An easy syntax allows the user to create a conjunctive query using the empty query and the symbols `+` and `@`. The function `Conjunctive.self_join_free` tests whether a conjunctive query is self-join-free.

FromString This module parses strings into the various data structures of the `Core` package. It uses the parser `FromStringParser` and the lexer `FromStringLexer`. A symbol is considered a constant if it starts with character a, b, c, d, e, f, g or h. Otherwise it is considered a variable.

7.1.2 Attack Graphs

This package is in charge of constructing attack graphs for a given conjunctive query. It is composed of two modules, one handling functional dependencies, the other generating attack graphs.

FunDep

```
type t
val set_from_query : Conjunctive.t -> Lset.t
val closure :
  t Lset.t -> Symbol.variable Lset.t -> Symbol.variable Lset.t
```

Note that the module is specialized for functional dependencies on variables. The first function computes the set of functional dependencies from a given conjunctive query (possibly with self-joins). More formally, given a query $\mathcal{Q}$, it computes $\mathcal{K}(\mathcal{Q})$. The second function computes the closure of a set of variables with respect to a set of functional dependencies on variables.

AttackGraph This module computes, for every atom F of a query, the atoms and the variables attacked by F.

```
type node = {
  q : Conjunctive.t;
  atom : Atom.t;
  keycl : Symbol.variable Lset.t;
  keycl' : Symbol.variable Lset.t;
  attvars : Symbol.variable Lset.t;
  attatoms : Atom.t Lset.t;
  mutable atoms_after : Atom.t Lset.t option;
}
```

The type node represents a node of the attack graph. It contains a reference to the complete query, the atom F that is attached to this node of the attack graph, the key closure of **keyvars**(F) with respect to $\mathcal{K}(\mathcal{Q} \setminus \{F\})$, the key closure of **keyvars**(F) with respect to $\mathcal{K}(\mathcal{Q})$, the variables attacked by F, and the atoms attacked by F. It also contains an optional set of atoms reachable by F in the attack graph (if the attack graph is acyclic, this coincides with the set of attacked atoms).

```
type t = node Lset.t

val make : Conjunctive.t -> t

val has_cycle : t -> bool
val unattacked : t -> Atom.t Lset.t

val strong : t -> node -> Atom.t -> bool
val transitive : t -> node -> Atom.t -> bool
val cyclic : t -> node -> Atom.t -> bool
```

An attack graph is just a set of nodes, as the attacks are contained in the node structure. The `make` function builds up an attack graph from a conjunctive query. The function `has_cycle` checks if the original query can be rewritten, the function `unattacked` gives the atom that can be rewritten at this stage. The functions `strong`, `transitive` and `cyclic` apply to attacks. I.e., they take as parameter a node (with atom F) and an atom (G) and checks whether the attack is, respectively, strong ($\mathcal{K}(\mathcal{Q}) \not\models \mathbf{keyvars}(F) \rightarrow \mathbf{keyvars}(G)$), transitive (the attack graph is acyclic and there exists an atom H such that F attacks H and H attacks G), and cyclic (the attack belongs to a cycle).

7.1.3 Rewrite

The rewrite package produces a consistent first-order rewriting for a given self-join-free conjunctive query using the algorithms developed in this thesis. It is able to produce a DRC, a TRC or an SQL query. Furthermore it can apply the grouping and/or the splitting strategies explained in Chapter 6.

This package makes use of OCaml functors, i.e., parameterised modules. The reason is that the DRC and TRC languages are very similar; hence most of their constructions are grouped in an RC module to avoid code duplication.

RC This module contains the part that is common to DRC and TRC. The parameters that make these two languages differ are what is an atom and what

is a variable.

```
module type RC = sig
  type atom
  type variable

  type formula = private
    | Certain of formula
    | True
    | False
    | Atom of atom
    | And of formula Lset.t
    | Or of formula Lset.t
    | Neg of formula
    | Implies of formula * formula
    | Equiv of formula * formula
    | Exists of variable Lset.t * formula
    | Forall of variable Lset.t * formula

  val vars : formula -> variable Lset.t
  val map : (formula -> formula option) -> formula -> formula
end
```

A relational calculus formula is an atom, a constant truth value, a Boolean combination of formulas, or a quantification of a formula. Note that conjunctions and disjunctions are not necessarily binary, and that quantifications may quantify over several variables at once. There is an additional construct, `Certain`, which stands for a subquery that cannot be rewritten. In addition to the type definition (which is private) and the variable extraction function, there are also functions not shown here that allow the user to build an RC formula. The map function navigates through the formula structure and applies a function to all of the subformulas when appropriate.

Finally, the functor allowing to specialize RC to either DRC or TRC,

or any other relational calculus language, is given below. Once atoms and variables are defined, the “language” is created and ready to use. Note that simplifications are made when a formula is created. For example, conjunctions of conjunctions will be unfolded and constant truth values will be eliminated when possible.

```
module type EQ = sig
  type t
  val ( = ) : t -> t -> bool
end

module Rc :
  functor (Atom : EQ) ->
  functor (Variable : EQ) ->
  (RC with type atom = Atom.t
       and type variable = Variable.t)
```

DRC Domain Relational Calculus uses our previously defined notions of variable and atom. A DRC query is a set of free symbols (not only variables) along with a DRC formula. A function translating a conjunctive query into a DRC query is also given. The rewrite function constructs a consistent query rewriting from a conjunctive query. The user can choose to apply the grouping and splitting strategies (see Chapter 6).

```
include Rc.RC
with type atom = Atom.t
 and type variable = Symbol.variable

type t = {
  free : Symbol.t Lset.t;
  formula : formula;
}

val conjunctive : Conjunctive.t -> t
val rewrite : ?group:bool -> ?split:bool ->
  Conjunctive.t -> t
```

TRC Tuple Relational Calculus uses equality between symbols as its atoms. The variables are strings associated to relation names. Otherwise the module signature is the same as the DRC module.

```
type symbol =
  | Map of (Symbol.variable * RelationName.t) *
          RelationName.attribute
  | Constant of Symbol.constant

include Rc.RC
with type variable = Symbol.variable * RelationName.t
 and type atom = symbol * symbol

type t = {
  free : symbol Lset.t;
  formula : formula;
}

val conjunctive : Conjunctive.t -> t
val rewrite : ?group:bool -> ?split:bool ->
  Conjunctive.t -> t
```

SQL The SQL module is basically a compiler from TRC formulas to SQL strings.

```
type t = string
var from_trc : Trc.t -> t

val conjunctive : Conjunctive.t -> t
val rewrite : ?group:bool -> ?split:bool ->
  Conjunctive.t -> t
```

7.1.4 Under-Approximations

This package of Canswer implements the theory developed in Chapter 5, which studied first-order under-approximations for queries that have no consistent first-order rewriting.

CqaFO This module implements **CQAFO** queries of the form $\exists\vec{X}\,\lfloor\mathcal{Q}\rfloor$. A variable that occurs in a **CQAFO** query of this form is called *outsourced* if it occurs in $\vec{X}$; otherwise it is *insourced*.

```
type t = private {
  query : Conjunctive.t;
  insourced : Symbol.variable Lset.t;
  outsourced : Symbol.variable Lset.t;
}
val make : Conjunctive.t -> t
val outsource : t -> Symbol.variable -> t
val substitute : t -> Symbol.variable -> Symbol.variable -> t
```

The `make` function transforms a conjunctive query into a **CQAFO** query where all the variables are insourced. The two other functions transform **CQAFO** queries by "outsourcing" a variable or by applying a substitution; the transformed **CQAFO** queries are contained in the original, non-transformed query.

Expanded This module generates the full set of queries and places them in a directed graph where an arrow from a query to another denotes the inclusion of the former into the latter.

```
type id = int
type node = private {
  id : id;
  cqafo : CqaFO.t;
  ag : AttackGraph.t;
}
type edge = {
  source : id;
  dest : id;
  equiv : bool;
}
type t = {
  nodes : node Lset.t;
  edges : edge Lset.t;
}
val make : Conjunctive.t -> t
```

Collapsed In the `Expanded` module, some edges are marked as "equiv". This means that the queries at the end of the edge are equivalent. This module collapses all the queries that are equivalent, producing a smaller graph.

```
type collapsed = private Expanded.t
val collapse : Expanded.t -> t
val remove_transitive_edges : collapsed -> Expanded.t
```

The `remove_transitive_edges` function removes an edge $A \to C$ if there exist two edges $A \to B$ and $B \to C$.

7.2 The Canswer Language

As keyboards featuring the characters $\forall$ or $\exists$ are not widespread, DRC could not be used as an input language for Canswer. Hence a new "DSL" (Domain-specific language) has been created. Its goals are to be simple enough for

the researcher who just wants to see an attack graph, but also sufficiently complete for the student who has to obtain pertinent SQL queries to conduct experiments.

The basic structure of a query goes as follows:

$$\mathtt{s_1 s_2 \ldots s_n \mid F_1 F_2 \ldots F_n}$$

where the s_i are symbols and the F_i are atoms. A symbol is an alphanumeric sequence starting with a lowercase character. If the first character is a, b, c, d, e, f, g or h, then the symbol is a constant; otherwise it is a variable. The part before the pipe is the free part of the query. To avoid errors, every variable occurring in this part must also occur in the atoms. If the query uses no free symbols, the pipe can be omitted.

An atom has the following form:

$$\mathtt{R(s_1 s_2 \ldots s_k \mid s_{k+1} \ldots s_n)}$$

where the s_i are symbols and `R` is the relation name (an alphanumeric sequence starting with an uppercase character). The symbols before the pipe form the key of the atom.

Although any conjunctive query can be presented in this syntax, most Canswer functionality only applies to self-join-free queries. When the user provides no attribute names, attribute names will be derived from attribute positions. For example, Canswer generates the following consistent SQL rewriting for the query `R(a b |)`:

```
SELECT TRUE FROM R WHERE R.1 = 'a' AND R.2 = 'b';
```

User-defined attribute names can be provided using the following syntax within atoms:

$$\mathtt{A : s}$$

where A is an attribute name and s is a symbol. An attribute name is an alphanumeric sequence that must start with an uppercase character. Named and unnamed attributes can be mixed. Canswer generates the following consistent first-order rewriting for `R(A:a B:b |)`:

```
SELECT TRUE FROM R WHERE R.A = 'a' AND R.B = 'b';
```

More syntactic sugar can be found on the home page of Canswer.

7.3 Top level interface

Once the library has been compiled, it is possible to compile a top-level interface by issuing "make top" in the root directory of Canswer. Then one can execute the "top.sh" script to get an interactive shell where Canswer is preloaded. This allows for rapid experimenting.

For example, the following session shows the input of a conjunctive query and the output of its rewriting in DRC and SQL.

```
# let q = "b x | R(A:x | B:y) S(B:y | C:a)";;
# let c = Canswer.Core.FromString.conjunctive q;;
# let drc = Canswer.Rewrite.Drc.rewrite q;;
# print_endline (Canswer.ToString.drc drc);;
# let sql = Canswer.Rewrite.Sql.rewrite q;;
# print_endline sql;;
```

The modules `ToString`, `ToLatex` and `ToDot` export structures in the corresponding formats.

7.4 Cansweb

7.4.1 Cansweb Interface

Canswer also comes with a web-oriented interface, named Cansweb. The starting page gives usage instructions and an input box allowing the user to input a conjunctive query in the Canswer format. When the "analyze" button is clicked, the tool proceeds by computing the attack graph as well as consistent first-order rewritings in DRC, TRC, and SQL (if they exist).

Since the URL of the output page contains the query being analyzed, one can bookmark queries in a browser for faster access. The tool itself does not propose a way to store queries.

Cansweb organizes its results in several categories:

Query gives the conjunctive query in Canswer syntax, TRC, DRC, and SQL.

Basic Properties shows the free and non-free variables and indicates whether the input query is self-join-free.

Atoms enumerates, for each atom F of the input query $\mathcal{Q}$, the following sets: $\mathbf{keyvars}(F)$, $F^{+,\mathcal{Q}}$, the set of variables attacked by F, the set of atoms attacked by F. For unattacked atoms F, the tool allows the user to continue with the subquery that remains after rewriting F.

Attack Graph shows the attack graph of $\mathcal{Q}$, where, as illustrated by Figure 7.3, unattacked atoms and attack cycles are typeset in a distinguished way. For each atom F, the key closure $F^{+,\mathcal{Q}}$ and the set of variables attacked by F are also shown.

FO definition gives consistent rewritings in DRC, TRC, and SQL. For each language, four rewritings are given, featuring or not the grouping and splitting strategies from Chapter 6.

Lattices give three directed graphs over query nodes.

- In the first one, each query $\mathcal{Q}'$ in the graph is such that $\mathcal{Q}' \sqsubseteq \mathcal{Q}$. Red nodes represent queries that have no consistent first-order rewriting. An arrow from $\mathcal{Q}_1$ to $\mathcal{Q}_2$ indicates that $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$. A bold arrow indicates that the two queries are equivalent.
- The second graph is obtained from the previous one by merging equivalent queries into a single node (some query is arbitrarily picked to represent the equivalence class).
- The third graph is the transitive reduction of the second graph.

We now look at the output of Cansweb for some queries encountered in this thesis. As Cansweb is built upon Canswer, it uses the input syntax of Section 7.2.

7.4.2 Cansweb Examples

In this section, we illustrate the functionality of Cansweb. Consider the following query $\mathcal{G}$ over the schema of Figure 1.

$$\mathcal{G} = \left\{ (n, p) \mid \texttt{Cars}(\underline{p}, \mathsf{Xiou}) \land \texttt{Hierarchy}(\underline{n}, n) \right\},$$

which can be expressed in Cansweb as follows:

```
n p |
Cars(LicensePlate:p | Employee:cXiou)
Hierarchy(Employee:n | Boss:n)
```

Since constants need to start with a lowercase character between a and i, Xiou is encoded as `cXiou`.

Query In addition to the Canswer syntax, Cansweb provides expressions in TRC, DRC, and SQL. The DRC expression for $\mathcal{G}$ shown next was obtained from the LaTeX code provided by Cansweb, extended with user-defined LaTeX macros for `variable`, `constant`, and `atom`.

$$\{p, n \mid ($$
$$\texttt{Cars}(\underline{\mathsf{LicensePlate} : p}, \mathsf{Employee} : \mathsf{cXiou})$$
$$\land \texttt{Hierarchy}(\underline{\mathsf{Employee} : n}, \mathsf{Boss} : n))\}$$

Attack Graph Cansweb renders attack graphs in SVG, an XML-based vector image format; see Figure 7.1. Cansweb also provides dot code and tikz code. Typically tikz code needs some tweaking in order to fit the document's general style.

Note from Figure 7.1 that Cansweb detects that our example query $\mathcal{G}$ consists of two subqueries that have no variables in common. In this example,

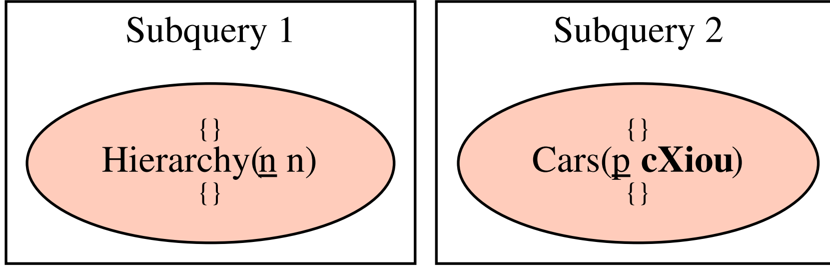

Figure 7.1: The attack graph of $\mathcal{G}$ as shown by Cansweb.

the edge-set of the attack graph is empty. Figure 7.2 shows how Cansweb renders the attack graph of Figure 6.2. The graph shows, for each atom F, the set $F^{+,\mathcal{Q}}$ and the set of variables attacked by F.

FO definition The following TRC query is a consistent first-order rewriting for $\mathcal{G}$ that minimizes the quantifier rank; its LaTeX code was generated by Cansweb. Listing 7.1 shows how Cansweb translates this TRC query into SQL.

$$
\begin{aligned}
&\{f_{Cars}.LicensePlate, t_{Hierarchy}.Employee \mid (\\
&\ \exists e_{Hierarchy} \forall u_{Hierarchy}(e_{Hierarchy}.Employee = u_{Hierarchy}.Employee \rightarrow (\\
&\quad u_{Hierarchy}.Employee = t_{Hierarchy}.Employee\\
&\quad \wedge u_{Hierarchy}.Boss = t_{Hierarchy}.Employee))\\
&\ \wedge \exists e_{Cars} \forall u_{Cars}(e_{Cars}.LicensePlate = u_{Cars}.LicensePlate \rightarrow (\\
&\quad u_{Cars}.LicensePlate = f_{Cars}.LicensePlate\\
&\quad \wedge u_{Cars}.Employee = cXiou)))\}
\end{aligned}
$$

Finally, we illustrate how Cansweb deals with queries that have no consistent first-order rewriting. The following query $\mathcal{C}$ has a cyclic attack graph which is shown in Figure 7.3.

$$
\mathcal{C} = \exists x_1 \exists x_2 \exists y_1 \exists y_2 \exists z \left(\mathtt{R}(\underline{\mathtt{a}}, x_1, x_2) \wedge \mathtt{S}_1(\underline{x_1, y_1}, z) \wedge \mathtt{S}_2(\underline{x_2, y_2}, z) \right).
$$

If the attack graph of an input query is cyclic, Cansweb will rewrite unattacked

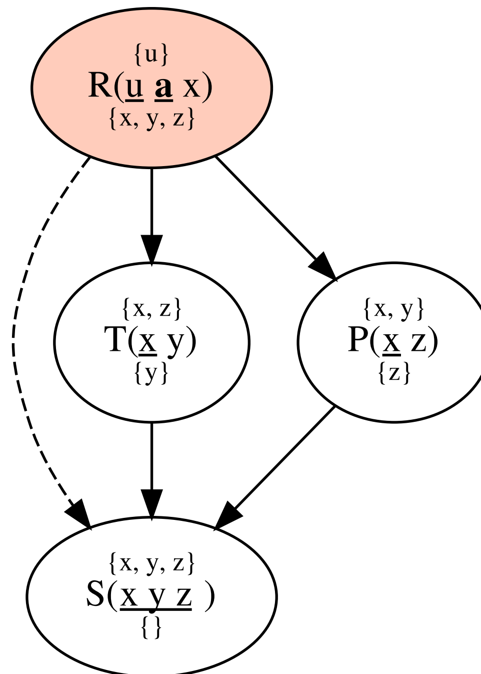

Figure 7.2: The attack graph of $\mathcal{Q}$ from Figure 6.2 on page 128, rendered by Cansweb.

```
SELECT f_Cars.LicensePlate, t_Hierarchy.Employee
FROM Cars AS f_Cars CROSS JOIN Hierarchy AS t_Hierarchy WHERE (
  EXISTS (
    SELECT *
    FROM Hierarchy AS e_Hierarchy
    WHERE NOT EXISTS (
      SELECT *
      FROM Hierarchy AS u_Hierarchy
      WHERE NOT (
        (
          u_Hierarchy.Employee = t_Hierarchy.Employee
          AND u_Hierarchy.Boss = t_Hierarchy.Employee)
        OR NOT e_Hierarchy.Employee = u_Hierarchy.Employee)))
  AND EXISTS (
    SELECT *
    FROM Cars AS e_Cars
    WHERE NOT EXISTS (
      SELECT *
      FROM Cars AS u_Cars
      WHERE NOT (
        (
          u_Cars.LicensePlate = f_Cars.LicensePlate
          AND u_Cars.Employee = 'cXiou')
        OR NOT e_Cars.LicensePlate = u_Cars.LicensePlate))))
```

Listing 7.1: $\mathcal{G}$ rewriting in SQL, built by Cansweb.

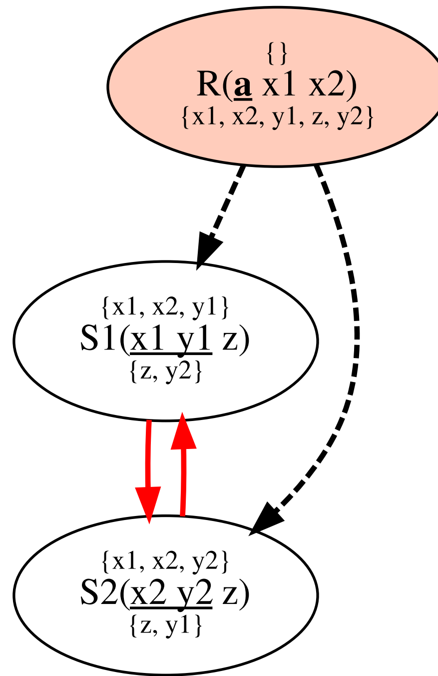

Figure 7.3: The attack graph of $\mathcal{C}$, rendered by Cansweb.

atoms until it reaches an attack graph in which every atom has non-zero indegree. The remaining subquery, which has no consistent first-order rewriting, is then provided as the argument of the function **CERTAIN**. For example, Listing 7.2 shows the SQL code for $\lfloor \mathcal{C} \rfloor$ constructed by Cansweb.

```sql
SELECT TRUE FROM DUAL WHERE EXISTS (
  SELECT *
  FROM R AS e_R
  WHERE NOT EXISTS (
    SELECT *
    FROM R AS u_R
    WHERE NOT (
      (
        u_R.199 = 'a'
        AND CERTAIN (EXISTS (
          SELECT *
          FROM S2 AS t_S2
          CROSS JOIN S1 AS t_S1
          WHERE t_S1.204 = t_S2.207)))
      OR NOT e_R.199 = u_R.199)))
```

Listing 7.2: $\mathcal{C}$ rewriting in SQL, built by Cansweb.

7.5 Conclusion

Canswer and Cansweb are ready-to-use tools that support research activities in the domain of CQA. These tools allow constructing attack graphs and constructing consistent first-order rewritings in DRC, TRC, and SQL. Their outputs are rendered in different formats (LaTeX, SVG, dot, tikz...) and can thus be easily be copy-pasted in research papers or other documents. The source code of Canswer and Cansweb is available on GitHub:

Canswer `https://github.com/shepard8/canswer`

Cansweb `https://github.com/shepard8/cansweb`

Chapter 8

Conclusion

In this thesis, we have made a number of theoretical and practical contributions in the field of Consistent Query Answering. We focused on primary key constraints and queries in the class **SJFCQ**, the class of conjunctive queries without self-joins. Our main contributions are summarized next.

- We are now able to use external knowledge about the data, in the form of satisfied join dependencies (of a restricted form) and satisfied functional dependencies. This information allows us to construct consistent first-order rewritings for queries that would otherwise not be rewritable. We have thus widened the class of queries that can be handled by the query rewriting technique.
- We developed a method that, given an input query $\mathcal{Q}$ in **SJFCQ**, returns a first-order query φ, called first-order under-approximation, such that φ is contained in $\lfloor \mathcal{Q} \rfloor$ (according to the standard definition of query containment). If $\mathcal{Q}$ has a consistent first-order rewriting, then $\varphi \equiv \lfloor \mathcal{Q} \rfloor$. More significantly, if no consistent first-order rewriting for $\mathcal{Q}$ exists, then φ is maximally contained in $\lfloor \mathcal{Q} \rfloor$ under particular syntactic restrictions imposed on under-approximations.
- Although **FO** is generally viewed as a low complexity class, static query optimization is nevertheless a major topic in research on database sys-

tems. We therefore investigated methods for reducing the quantifier rank and the number of quantifier blocks in consistent first-order rewritings. This amounts, among others, to reducing the depth of `NOT EXISTS` nesting in SQL. Further experiments are needed to study whether these syntactic simplifications effectively result in lower query execution times.

- Our theory has been implemented in a software tool. In a research environment, this tool has already been used for checking new ideas and conjectures on large examples which are tedious to elaborate by hand [Wij]. The tool could be helpful to practitioners who want to learn or adopt consistent query answering. In fact, a major aim of our work is to make consistent query rewriting a viable alternative for data cleaning.

We conclude this thesis by outlining some open research questions.

- In Chapter 4, we investigated how the method of consistent first-order rewriting can take advantage of the presence of key-join dependencies and functional dependencies. It would be interesting to study the impact of other constraints on consistent first-order rewriting.
- In Chapter 5, we only looked at under-approximations of consistent query answers. It would be interesting to also study first-order over-approximations of the consistent answers to a query $\mathcal{Q}$, i.e., first-order queries that contain $\lfloor \mathcal{Q} \rfloor$ and that are otherwise minimal in some sense.
- Our research has focused on consistent first-order rewritings for queries in **SJFCQ**. It is an open problem to extend our results to queries with self-joins, to unions of conjunctive queries, or to queries with some restricted form of negation.
- Our tool, Canswer, constructs first-order under-approximations by traversing a large lattice. This computation takes exponential time in the size of the input query $\mathcal{Q}$, which turns out to be unfeasible if the sum of the

arities of $\mathcal{Q}$'s atoms exceeds 15. Therefore, the existence of more efficient constructions needs to be explored. Also, we have not studied the theoretical complexity of constructing first-order under-approximations.

- In Chapter 6, we studied syntactic simplifications in the context of consistent first-order rewriting. Syntactic simplifications consisted in reducing the quantifier depth and the number of quantifier blocks. We did not show, however, that our methods minimize these metrics. It also remains an open question to what extent these syntactic simplifications result in lower execution times on SQL RDBMSs. An important research task is to investigate the interplay between our syntactic simplifications and the query plan generated by the query optimizer of the RDBMS.

- On a more practical level, Canswer can be improved in several ways. One extension could be an implementation of a polynomial-time algorithm for **CERTAINTY**($\mathcal{Q}$) when the problem is in **P** [KW17].

Unfortunately, despite its elegance, the paradigm of Consistent Query Answering has not yet been implemented in mainstream database systems. We hope that some day, a note about uncertainty management can be found in the changelog of PostgreSQL (or some other RDBMS).

Appendix A

Notations

Notation	Description
$\mathsf{a}, \mathsf{b}, \mathsf{c}$	Constants
u, v, x	Variables
$\mathbf{R}, \mathbf{S}, \ldots$	Schemas
$\mathcal{Q}, \mathcal{R}, \mathcal{S}$	Queries
$\mathtt{R}, \mathtt{S}, \mathtt{T}$	Relations

dom	Set of available constants. (p. 17)
var	Set of available variables. (p. 19)
sym	Set of available symbols ($\mathbf{dom} \cup \mathbf{var}$). (p. 19)
$\|S\|$	Cardinality of the set S (p. 23)
$F_{x \to y}$	The substitution of each (free) occurence of x with y in the object F. (p. 20)
$\mathcal{Q}(x/\mathsf{a})$	$\mathcal{Q}$ in which the variable x is interpreted as the constant a. (p. 38)
$v[\mathbf{Name}]$	The value of attribute **Name** in fact/atom/tuple/tuple variable v. (p. 17)
$[n, k]$	A signature denoting arity n and key length k. (p. 33)
$\mu \circ \nu$	Mapping applying ν then μ (composition). (p. 93)
$\mathcal{K}(\mathcal{Q})$	The set of functional dependencies associated to each atom of query $\mathcal{Q}$. (p. 40)
$F^{+,\mathcal{Q}}$	The key closure of atom F in query $\mathcal{Q}$. (p. 40)

$F \overset{\mathcal{Q}}{\rightsquigarrow} G$	An attack from atom F to atom G in the attack graph of query $\mathcal{Q}$ (omitted if clear from context). (p. 40)
$F \overset{\mathcal{QR}}{\rightsquigarrow} G$	An homomorphic attack from atom F to atom G. (p. 105)
$\mathbf{type}(x)$	An infinite set of constants, such that $x \neq y$ implies $\mathbf{type}(x) \cap \mathbf{type}(y) = \emptyset$. (p. 96)
$\mathtt{R} : A \to B$	A functional dependency from set of attributes A to set of attributes B in relation $\mathtt{R}$. (p. 23)
$\mathtt{R} : \bowtie [\mathcal{A}]$	A join dependency over the sets of attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ in relation $\mathtt{R}$. (p. 26)
$\mathtt{R}[A] \subset \mathtt{S}[B]$	An inclusion dependency from the projection of relation $\mathtt{R}$ onto the attributes A to the projection of relation $\mathtt{S}$ onto the attributes B. (p. 25)
$\lfloor \mathcal{Q}(\vec{x}) \rfloor$	Consistent answer to $\mathcal{Q}(\vec{x})$. (p. 34)
$\mathcal{Q} \otimes \Sigma$	The extension of query $\mathcal{Q}$ using the set of constraints Σ. (p. 59)
$\mathtt{R}_i^\sigma()$	The ith extension of atom $\mathtt{R}$ towards functional dependency σ ($i \in \{1\}, 2$). (p. 60)

$\mathtt{R}^{\bowtie}_{\circ}()$	The all-key extension of the key join dependency on atom $\mathtt{R}$. (p. 60)
$\mathtt{R}^{\bowtie}_{i}()$	The ith extension of atom $\mathtt{R}$ towards its key join dependency ($1 \leq i \leq K$ where K is the number of components in the key join dependency). (p. 60)
$[\![\mathbf{db}]\!]$	A subset of **db**. (p. 62)
SJFCQ	The class of self-join-free conjunctive queries. (p. 20)
CQAFO	A class of queries using consistent answers as atoms. (p. 86)
FD	Functional dependency. (p. 23)
KJD	Key join dependency. (p. 55)
qschema($\mathcal{Q}$)	Set of relation names used in the query $\mathcal{Q}$. (p. 60)
schema($\mathtt{R}$)	Set of attributes mapped from $\mathtt{R}$ by **schema**. (p. 17)
free(ϕ)	Set of free variables of first-order formula ϕ. (p. 20)

vars(S)	Set of variables in the set of symbols S. (p. 20)
vars$(\vec{x})$	Set of variables in the sequence of symbols $\vec{x}$. (p. 20)
vars(F)	Set of variables in the atom F. (p. 20)
keyvars(F)	Set of variables in the key of atom F. (p. 34)
vars(ϕ)	Set of variables in the formula ϕ. (p. 20)
vars$(\mathcal{Q})$	Set of non-free variables in the query $\mathcal{Q}$. (p. 20)
adom(**db**)	The set of constants used in the database **db**. Also called its active domain. (p. 18)
repairs(**db**)	The set of repairs of database **db**. (p. 33)
attackgraph$(\mathcal{Q})$	The attack graph of query $\mathcal{Q}$. (p. 40)
head$(\mathcal{Q})$	The first atom of query $\mathcal{Q}$ which has no incoming attack. (p. 105)

Bibliography

[ABC99] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos H. Papadimitriou, editors, *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 31 - June 2, 1999, Philadelphia, Pennsylvania, USA*, pages 68–79. ACM Press, 1999.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[Ber11] Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[BL13] Leopoldo E. Bertossi and Lechen Li. Achieving data privacy through secrecy views and null-based virtual updates. *IEEE Trans. Knowl. Data Eng.*, 25(5):987–1000, 2013.

[CFG+10] Nguyen Vi Cao, Emmanuel Fragnière, Jacques-Antoine Gauthier, Marlène Sapin, and Eric D. Widmer. Optimizing the marriage market: An application of the linear assignment model. *European Journal of Operational Research*, 202(2):547–553, 2010.

[CM05] Jan Chomicki and Jerzy Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[Dec13] Alexandre Decan. *Certain Query Answering in First-Order Languages.* PhD thesis, Université de Mons, 2013.

[DPW12] Alexandre Decan, Fabian Pijcke, and Jef Wijsen. Certain conjunctive query answering in SQL. In Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger, editors, *Scalable Uncertainty Management - 6th International Conference, SUM 2012, Marburg, Germany, September 17-19, 2012. Proceedings*, volume 7520 of *Lecture Notes in Computer Science*, pages 154–167. Springer, 2012.

[DRS09] Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[DRS11] Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Queries and materialized views on probabilistic databases. *J. Comput. Syst. Sci.*, 77(3):473–490, 2011.

[End72] Herbert B. Enderton. *A mathematical introduction to logic.* Academic Press, 1972.

[FFM05] Ariel Fuxman, Elham Fazli, and Renée J. Miller. Conquer: Efficient management of inconsistent databases. In Fatma Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 155–166. ACM, 2005.

[FM05] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In Thomas Eiter and Leonid Libkin, editors, *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, vol-

ume 3363 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2005.

[FM07] Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007.

[GGZ03] Gianluigi Greco, Sergio Greco, and Ester Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.

[GPW14] Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain query answering in partially consistent databases. *PVLDB*, 7(5):353–364, 2014.

[GPW15] Floris Geerts, Fabian Pijcke, and Jef Wijsen. First-order under-approximations of consistent query answers. In Christoph Beierle and Alex Dekhtyar, editors, *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings*, volume 9310 of *Lecture Notes in Computer Science*, pages 354–367. Springer, 2015.

[GPW17] Floris Geerts, Fabian Pijcke, and Jef Wijsen. First-order under-approximations of consistent query answers. *Int. J. Approx. Reasoning*, 83:337–355, 2017.

[Imm99] Neil Immerman. *Descriptive complexity.* Graduate texts in computer science. Springer, 1999.

[KP12] Phokion G. Kolaitis and Enela Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

[KPT13] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

[KW15] Paraschos Koutris and Jef Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In Tova Milo and Diego Calvanese, editors, *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 17–29. ACM, 2015.

[KW17] Paraschos Koutris and Jef Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[Lib15] Leonid Libkin. Sql's three-valued logic and certain answers. In Marcelo Arenas and Martín Ugarte, editors, *18th International Conference on Database Theory, ICDT 2015, March 23-27, 2015, Brussels, Belgium*, volume 31 of *LIPIcs*, pages 94–109. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[Mai83] David Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

[Mar02] Jerzy Marcinkowski. The [exist]*[forall]* part of the theory of ground term algebra modulo an AC symbol is undecidable. *Inf. Comput.*, 178(2):412–421, 2002.

[MW13] Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.

[Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[RD00] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[Top09] Rodney W. Topor. Safety and domain independence. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 2463–2466. Springer US, 2009.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[Wij] Jef Wijsen. Personal communication.

[Wij04] Jef Wijsen. Making more out of an inconsistent database. In Georg Gottlob, András A. Benczúr, and János Demetrovics, editors, *Advances in Databases and Information Systems, 8th East European Conference, ADBIS 2004, Budapest, Hungary, September 22-25, 2004, Proceesing*, volume 3255 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2004.

[Wij05] Jef Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3):722–768, 2005.

[Wij06] Jef Wijsen. Project-join-repair: An approach to consistent query answering under functional dependencies. In Henrik Legind Larsen, Gabriella Pasi, Daniel Ortiz Arroyo, Troels Andreasen, and Henning Christiansen, editors, *Flexible Query Answering Systems, 7th International Conference, FQAS 2006, Milan, Italy, June 7-10, 2006, Proceedings*, volume 4027 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2006.

[Wij09] Jef Wijsen. On the consistent rewriting of conjunctive queries under primary key constraints. *Inf. Syst.*, 34(7):578–601, 2009.

[Wij10a] Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In Jan Paredaens and Dirk Van Gucht, editors, *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on*

Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA, pages 179–190. ACM, 2010.

[Wij10b] Jef Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Inf. Process. Lett.*, 110(21):950–955, 2010.

[Wij12] Jef Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9:1–9:35, 2012.

[Wij13] Jef Wijsen. Charting the tractability frontier of certain conjunctive query answering. In Richard Hull and Wenfei Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 189–200. ACM, 2013.