Completeness of Queries over Incomplete Databases

Diplomarbeit

vorgelegt an der

Technischen Universität Dresden,
Fakultät für Informatik

eingereicht von Simon Razniewski

Betreuender Hochschullehrer:
Prof. Dr.-Ing. Franz Baader

Externer Betreuer:
Prof. Dr. rer. nat. Werner Nutt
Freie Universität Bozen

Dresden, im November 2010

SIR HUMPHREY: If local authorities don't send us the statistics that we ask for, then government figures will be a nonsense.

JIM HACKER: Why?

SIR HUMPHREY: They will be incomplete.

JIM HACKER: But government figures are a nonsense anyway.

BERNARD WOOLLEY: I think Sir Humphrey wants to ensure they are a complete nonsense.

Yes, Minister, Series Three Episode Three, BBC 1982.

Acknowledgements

First of all, I would like to thank my supervisor, Werner Nutt. Without him, his patience and his advice this thesis would not have been possible at all.

I am thankful to Dmitrij Miliaev who has written a Bachelor thesis on the topic of incompleteness and who helped me much to get into the topic.

I am also thankful to Zeno Moriggl and Martin Prosch from the school IT department of the province of South Tyrol who initiated the research collaboration that led to my thesis and who invested their time to give me an understanding of their practical problems.

Many thanks go to Franz Baader, who immediately agreed to take supervision from Dresden and made it possible for me to write my thesis in Bozen.

Thanks to all people from the KRDB group in Bozen who welcomed me in their group and provided a friendly and productive atmosphere for working.

Without my teacher and organizer of the Erasmus program, Uwe Petersohn, I might have never come to Bozen, thank you.

Finally, thank you to my family for everything.

Contents

1 Overview 9

2 Introduction 11
2.1 Motivation 11
2.2 Related Work 14
2.3 Our Contribution 17

3 Problem Formalization 21
3.1 Standard Definitions 21
3.2 Running Example 22
3.3 Problem-specific Definitions 23

4 Characterizing Query Completeness 27
4.1 Multiset Semantics 28
4.2 Set Semantics 29

5 Completeness Reasoning 33
5.1 Inferring Local Completeness from Local Completeness 34
5.2 Inferring Query Completeness from Local Completeness 36
5.3 Inference from Query Completeness 38
5.3.1 Inferring Query Completeness from Query Completeness . . 39
5.3.2 Inferring Local Completeness from Query Completeness . . . 42

6 Reasoning with Additional Information 43
6.1 Schema Constraints 43
6.1.1 Keys 45
6.1.2 Foreign Keys 46
6.2 Finite Domain Constraints 46
6.3 Extensional Information 53
6.4 Schema and Extensional Information 58

6.4.1 Exact-cardinality constraints 63

7 Strategies for Stating Completeness 65

7.1 Where Completeness Information Comes from 66

7.2 Partitioning Local Completeness Statements 67

7.3 Stating Completeness with Cardinality Assertions 68

8 Implementation Issues 71

8.1 General Issues 71

8.2 Practical Finite Domain Containment 72

8.3 The MAGIK Implementation 76

9 Conclusion 81

Chapter 1

Overview

In this thesis we investigate the problem of query completeness over partially incomplete databases.

Incomplete data is an ubiquitous problem in practical data management. Reasoning over incomplete data has been studied extensively for many years. In the commonly used formalisms, arbitrarily large database instances are considered possible, and research focuses on possible and certain answers for queries over incomplete data.

In many practical scenarios, however, information is present that certain parts of the generally incomplete data are complete. Then, the question arises whether for a given query its answer over the incomplete data can be deemed to be complete.

In this thesis, we present a novel approach to the problem of deciding query completeness over incomplete databases, which is the first by which the problem can generally be decided.

This thesis is divided as follows: In chapter 2 we present application scenarios and discuss earlier work on the problem. In chapter 3 we formalize the problem and give important definitions like incomplete databases, completeness assertions and query completeness. In Chapter 4 we discuss the necessary and sufficient conditions for query completeness over incomplete databases. Chapter 5 shows the completeness reasoning problems along with decision procedures for them. Chapter 6 shows how database schema information and extensional information can be utilized in these reasoning processes to deduce more completeness. Chapter 7 discusses concepts and strategies for stating completeness of parts of incomplete databases. Chapter 8 discusses implementational issues and a prototypical implementation developed in the MAGIK project. Chapter 9 draws a conclusion and discusses possible future research questions.

Chapter 2

Introduction

2.1 Motivation

Databases as information storing systems are everywhere nowadays. In formal logic, reasoning under incomplete information has been studied since its very beginning. With the emergence of computer science and database theory, many approaches to management of and reasoning over incomplete information have been developed. With relational databases as the most widespread technology for information storing systems, incomplete information management in relational databases is of particular interest.

The first contribution to management of incomplete information in relational databases was the introduction of the unknown data [C75] and the understanding of incomplete databases by the Open World Assumption (OWA). Next milestones were the introduction of v- and c-tables and the theory of representation systems capturing both incomplete databases and answers to queries [IL84]. More recent approaches are Probabilistic Databases [DRS09] or Description Logic knowledge bases [B03].

Traditionally, databases are understood as being complete with respect to their application domain. If a fact is not mentioned in a database, then it does not hold in the application that is modelled. This understanding is called the *Closed World Assumption*. The OWA differs from it in the point that any fact not explicitly mentioned in the database that is consistent with the database, may hold or not. Thus, the semantics of an incomplete database under the OWA is the set of all possible complete databases that extend the incomplete data.

Databases with Codd tables or with v- and c-tables differ in the way of how unknown data fields may be substituted. While every occurrence of a null value in a Codd table may be replaced by an arbitrary value independent of other replacements, v-tables introduce the concept of named null values, which allow to express

the presence of the same unknown value in different tuples. The c-tables allow a further refinement of the unknown information by enhancing it with conditions, e.g., comparisons or range restrictions. The basic idea of the semantics of all three formalisms is the same: It is the set of all databases that can be derived from the incomplete database by substituting the unknown values with constants that respect possible constraints and adding further tuples.

For a fixed formalism for representing incomplete databases, the most investigated questions are how the general query answers over incomplete databases can be represented, and how the set of *possible answers* and of *certain answers* can be computed. For a fixed incomplete database and query, the set of possible answers contains all tuples that are in the query answer over some valid extension of the incomplete database, while the set of certain answers contains all tuples that are in the answer over all valid extensions.

However, little attention has been paid so far to combinations of the open- and the closed-world assumption in databases. In many practical scenarios, databases are generally incomplete, but some parts of the data may be known to be complete. Answering queries in such scenarios requires both open- and closed-world reasoning. Especially interesting is there the question of *query completeness*: Over a database that is partially incomplete, is the query answer over the database complete, or may tuples in the answer be missing? In other words, are the sets of possible answers and of certain answers the same?

This question was first investigated by Amihai Motro in [Mo89] and later by Alon Levy in [Le96]. Both investigations address the same question but use different formalisms for expressing partial completeness of a generally incomplete database. A more recent work with focus on practice is that of Dmitrij Miliaev [Mi10]. We discuss all three investigations in more detail in the next section.

We see a general interest in this question in the field of *data quality*. Generally, data quality studies how good data serves its purpose. Aspects of data quality concern consistency, validity, timeliness and accuracy, and also completeness. Traditionally, work on data quality deals more with statistical characterizations of these aspects. Whether some generally incomplete data is complete enough to serve its purpose of allowing to answer a query completely, is a relevant question in this field.

A special interest for the question is also in the field of *data integration*. Data integration investigates methodologies to combine heterogeneous data sources and allow unified access to them. When some data sources are stated to be complete, it becomes an interesting question, in how far the combined data is complete [D97].

Next, we present two application scenarios for the question of query completeness over partially incomplete databases. The first is the scenario of school data management of a provincial school administration. It is a real-world problem from

the school IT department of the province of South Tyrol, which also was the starting point of our work. For illustrative reasons, we will use a drastically scaled down version of the original problem, reduced to a single school only as the running example in this thesis. The second example is an invented business world scenario that shows the relation to the problem of data integration.

Example 1: Provincial School Data Management

The province of South Tyrol uses a distributed database system for managing school data. School data includes in particular information about students regarding school, class and language enrolment, mother tongue, nationalities, disabilities and other.

The data is maintained in a decentralized manner, which means that every school is responsible for maintaining its own data. Commonly, the maintenance is done by the school administrators.

Periodically, the statistical department of the province queries the database and derives statistics, which are the basis of administrative decisions of the school department. These decisions include in particular the assignment of teachers to schools. In order to make the right decisions, it is important to have complete knowledge of the student enrolments per school.

This however is a serious problem, since schools are notoriously late with submitting their data, and often submit incomplete data, e.g., forgetting complete classes sometimes.

For that reason, the province IT department is interested in setting up a system that allows to track database completeness and enrich query answers with completeness information. For query answers guaranteed to be complete, one would like to understand on which basis and how the completeness guarantee was derived. For incomplete answers, one would like to understand which parts of the database need to be completed in order to give a guarantee for query completeness.

To illustrate this, consider a query *Give the enrolments of all students in South Tyrol* is derived to be incomplete. Then, one would like to get the additional information *The query is incomplete because the Volksschule Mals did not submit any data, and the Handelsoberschule Bozen forgot to submit data for levels 10 to 12.* If the query above is incomplete, the school department might formulate a second query *Give all enrolments for ladin schools* and find out that this query is complete, then being able to start planning for the ladin schools already.

Example 2: Business Data Integration

As a second example, consider two companies that are merging. Merging the organizational structures includes in particular the merge of their IT structures.

Assume both companies have maintained databases which were only partially complete. When joining the databases in a common schema it becomes important to know in which parts the joint database is complete, and in which not.

As a more specific example, consider that company one mainly produced products of class A, while company two mainly produced products of class B. Neither one produced B nor two produced A. Company one listed all kinds of A's it was producing in a table which was complete, company two did the same for the B's. Then, joining the two tables into one table for all products will not yield a generally complete table, because both companies did not list all the minor items they produced. However, if someone is interested in a product group that is contained in A and B, he could still find a complete listing due to the completeness of the original tables A and B (illustrated in figure 2.1).

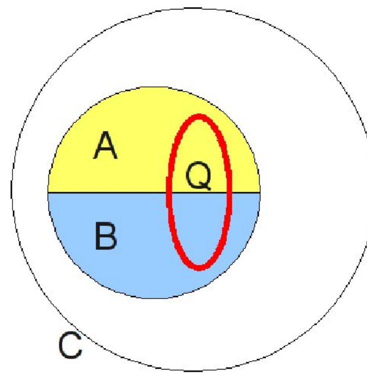

Figure 2.1: A table C containing the data from tables A and B can not be guaranteed to be complete. However, the query Q would still be complete.

2.2 Related Work

Amihai Motro [Mo89] was the first to introduce the concept of partial incorrect and incomplete databases and to analyse how query integrity can be derived from integrity assertions. By the term integrity, he subsumes both completeness and validity. A partially incorrect and incomplete database is a database that may contain both facts that do not hold in the real world, and miss facts that hold in the real world. A relation or a part of a relation then is considered to be *complete*, if it contains all facts that hold in the real world. It is considered to be *valid*, if it contains no facts that do not hold in the real world.

His idea is to introduce a meta relation for each database relation, that contains information about the integrity of views of the relation. That is, database integrity

is expressed in terms of view integrity. A query answer can be derived to be valid or complete, if a rewriting exists that uses only parts of relations that are also asserted to be valid or complete by the meta relations.

This approach is correct, that is, whenever a query is derived to be complete, it really is. However, the method is not complete [1], because the method searches only for conjunctive-query rewritings. Segoufin and Vianu have shown in [SV05], that conjunctive queries are not a complete rewriting language between conjunctive queries, that is, there may exist a rewriting of one conjunctive query in terms of conjunctive-query views, that is not a conjunctive query itself. Then, completeness of the views would imply completeness of the query, but Motro's method would not detect it because no conjunctive query rewriting would exist.

Another approach was presented by Alon Levy in [Le96]. He introduced a different formalism for stating completeness of parts of a generally incomplete database, namely that of *local completeness statements*. Local completeness statements only allow specification of completeness for relations or parts of relations without further projections. The satisfaction of local completeness statements is necessarily independent of the completeness of other relations, a property we consider to be highly important in practice. Local completeness statements and view completeness statements are different concepts that cannot be reduced to each other. For the problem of deciding query completeness with respect to local completeness, Levy proposed a reduction to the problem of query independence from updates. However, the reduced problems are generally undecidable. Also, he presented the idea of utilizing schema and extensional information in order to deduce completeness that holds in specific cases only.

The main idea about deciding query completeness with respect to local completeness in Levy's paper is that a query can be concluded to be complete with respect to given local completeness statements, if the query result is independent from updates on those parts of relations that *lie outside* the parts stated to be complete. These outside parts are exactly the complements of those parts that are asserted to be complete. They can be characterized by the negations of the statements that describe the complete parts. Testing query independence of updates then can either be reduced to unsatisfiability of a query [El90] or query containment [LS93]. The reduction of query completeness to query independence of updates is correct and complete, meaning that a query is complete with respect to local completeness statements exactly if it is independent of updates on the parts outside the complete parts. However, the generated problems of query independence of updates are generally undecidable. This is because the reduction introduces negation symbols, and query containment for queries containing

[1] *Complete* refers here to the completeness of a decision procedure, not to query completeness

negation is only decidable if no projection is applied to negated parts of a query [SY81].

The second main contribution of Levy's paper is the idea of deciding query completeness with respect to a concrete database instance. He observed that in some cases, completeness of a query does not follow from a set of assertions alone, but follows if one both considers the data actually stored in the database and takes into account additional conditions that the complete database has to satisfy.

More detailed, the observation is that whenever an incomplete relation is joined with another complete relation over the key attribute of the latter, and the incomplete relation contains one tuple for each tuple in the complete one, the result of the join is always complete. That holds because when the join attribute is the key attribute for the second relation, then the second relation can contain at most one tuple per join attribute value.

The last issue discussed in Levy's paper is the connection between query correctness and query completeness. He shows that the two problems are closely related and introduces the concept of local correctness statements as analogon to local completeness statements. Because query completeness can be reduced to queries independent of insertion updates, query correctness to independence of queries independent of deletion updates, and for a fixed query and update, the latter implies the former, he argues that query correctness implies query completeness.

As stated in the beginning, the most serious shortcoming in Levy's approach is the fact that query completeness is reduced to a generally undecidable problem. Only for projection-free queries or trivial local completeness statements, that are local completeness statements only containing self-joins, query completeness is decidable.

Besides, the formalization contains several minor flaws. It is nowhere stated that any completeness assertions are required for the main theorem to hold, which can easily be seen to be wrong. On the contrary, in the proof of the theorem it seems as if there existed one statement for every relation, which would make the theorem become correct, but would be too strong a requirement, since only statements for relations used in a query are needed. Furthermore, the formalization allows only at most one completeness statement per relation, which is not wrong but again an unnecessary restriction.

The discussion of the problem of deciding query completeness with respect to concrete database instances presents an interesting idea, however it does not give any motivation for why the presented technique could work in practice. In fact, without considering foreign keys, we believe there is no motivation for why the method could work in practice. Also, the method does not consider the effect of schema information and extensional information separately, but only discusses the

case of both appearing together, omitting the discussion of the effects that schema and extensional information already have separately.

Finally, Levy's argumentation about query correctness implying query completeness is not correct. In partially incomplete and partially incorrect databases, it is not sufficient to detect query correctness to conclude query completeness, because the statements about which parts are correct and which are complete can well be very different.

A third relevant work is the Bachelor thesis of Dmitrij Miliaev [Mi10]. It focuses more on practical aspects of ensuring query completeness, in particular, it presents a prototypical implementation of a system for managing database completeness.

The first contribution of this work is the discussion of a real-world problem about query completeness over a partially incomplete database. The problem is that of the school data management by the school IT department of the province of South Tyrol, which we also refer to in this thesis. The discussion of that problem gives an interesting insight, namely that it is not only important to be able to decide whether a query is complete with respect to given completeness statements, but rather that it is interesting to know for a given query, which statements have to be made in order to ensure that the query will be complete.

The discussion also directly leads to a second contribution, namely the introduction of the idea of partitioning completeness statements using finite domains or other tables.

Also the idea of cardinality assertions as expressions for stating conditional completeness of a partially incomplete databases is a result of this discussion.

On the conceptual side a contribution is the comprehensible discussion of the impact of foreign key assertions on completeness checking in presence of extensional information. While the theory is already contained in Levy's work about completeness checking in presence of functional dependencies, the discussion of foreign keys is a valuable contribution to the understanding of practical database completeness management.

The biggest contribution of Miliaev's work is the presentation of a prototypical implementation of a system for managing database completeness. It implements some of the results presented in the work such as the completeness statement partitioning and the idea of completeness checking using foreign key assertions, showing that these results can well be turned into practice.

2.3 Our Contribution

The contribution of this thesis is twofold. First, it presents a comprehensive discussion of the topic of query completeness. Second, it contains a number of technical

results, with the most important ones being the following:

- *Reduction of query completeness reasoning to query containment, yielding a generally decidable decision problem.* Motro's implicit reduction to the problem of whether a query can be rewritten using views (implicit because this problem was not formalized at the time when Motro published his work) is a correct reduction to a problem decidable for conjunctive queries, however the reduction is not complete. Levy's reduction to the problem of queries independent of updates is correct and complete, however the reduction yields problems that are decidable only in very special cases.

- *Characterization of which parts of a database have to be complete for query completeness.* Using Levy's solution to the query completeness problem, no information could be gained in which parts a database really has to be complete when a query shall be complete. When returning that a query was not complete with respect to given local completeness statements, the algorithm for deciding query completeness could tell on which relation the local completeness statement was not sufficient. It could not tell which statement would have been required instead, and the topic of characterizing conditions for query completeness was not discussed at all. We given an extensive characterization in chapter 4.

- *Identification of core assumptions.* Although Levy's reduction is correct, it contains several unnecessary assumptions (at least and at most one local completeness statement per relation, no self join in local completeness statements). We drop unnecessary assumptions and present a more general framework for deciding query completeness.

- *Detailed examination of how schema and instance information affects completeness reasoning.* Levy already showed for a special case, how schema information together with instance information can lead to further completeness conclusions. We provide a systematic analysis, how several kinds of schema information (keys, foreign keys, finite domain constraints) and instance information separately and combined affect completeness reasoning.

- *Investigation of how completeness management and assertions can work in practice.* Earlier work focused mostly on the theory. We investigate how completeness management and asserting can work in practice, and illustrate our investigations with an example of a school data management system.

- *Discussion of implementational issues.* As we participated in the implementation of a completeness management prototype, we are able to discuss practical implementational issues.

- *Presentation of an algorithm for containment checking under finite domains.* As a side result of our investigation of finite domain constraints, we are the first to present an algorithm for effective containment checking under finite domain constraints. Finite domains are a constraint concept that limits the set of possible values of certain attributes of relation instances to finite sets, which seems to have practical relevance.

Chapter 3

Problem Formalization

3.1 Standard Definitions

In this section, we give standard definitions about databases and queries.

Assume a countably infinite set *Dom* of *constants* and a countably infinite set *Var* of *variables*. The union of the sets of variables and constants we denote as *terms*. By convention, we will start variable names with capital letters and constant names with small letters, with the exception of the lower-case letter s, which will be used for denoting terms.

A *database schema* is a set Σ of relation symbols together with an arity for each.

An *atom* over a relation R with arity n is an expression $R(s_1, \ldots, s_n)$, where $s_1, \ldots, s_n \in$ are terms. A *ground atom* is an atom where $s_1, \ldots, s_n$ are constants.

A *relation instance* of relation a R is a finite set of ground atoms over R. A *database instance* I of Σ is a finite set of ground atoms over the symbols in Σ.

In the following, we assume Σ to be fixed.

A *literal*, denoted as L, is a positive or negated atom. A *comparison predicate* is one of the symbols in $\{=, \leq, <\}$. A *comparison*, denoted as V, is an atom where the predicate is a comparison predicate.

A *conjunctive condition*, denoted as G, is an expression of the form $L_1(\bar{s}_1), \ldots, L_m(\bar{s}_m), V_{m+1}(\bar{s}_{m+1}), \ldots, V_n(\bar{s}_n)$, where the $\bar{s}_i$ are vectors of terms each time with the same arity as the relation symbol of literal L_i, and V_{m+1} to V_n are positive or negated comparisons.

A *conjunctive query* is an expression of the form $Q(\bar{X}) :\!- B$, where B is a conjunctive condition. A *simple conjunctive query* is a conjunctive query containing no comparisons and no negation. We call the variables in $\bar{x}$ the *distinguished variables* of Q. We call B the *body* of Q, and the variables appearing in B but not in $\bar{x}$ the *nondistinguished variables.* The *frozen body* of a query is the set of ground

atoms that corresponds to the body of the query, when each variable is interpreted as a new constant. The term *view* will be used synonymously to the term query.

A conjunctive query is *safe*, if each variable in $\bar{x}$ also appears in B, and every variable appearing in a negated literal or a comparison predicate also appears in a positive literal in B.

A *valuation* υ for a query Q is a mapping from the variables in Q, and *dom* to *dom*, that is the identity on *dom*. A valuation υ *satisfies* a query Q over a database instance D, if the tuples derived from applying υ to each set of terms s_i in Q are contained in the relation extension of relation R_i over D each.

The *result* of a query Q with distinguished variables $\bar{X}$ over a database instance D is a set defined as follows: $\{\upsilon\bar{X} \mid \upsilon$ is a valuation for Q, and υ is satisfying for Q over $D\}$.

A conjunctive query Q_1 is *contained* in another query Q_2, denoted by $Q_1 \subseteq Q_2$, if the result of Q_1 is a subset of the result of Q_2 over all database instance. Two conjunctive queries Q_1 and Q_2 are *equivalent*, if Q_1 is contained in Q_2 and Q_2 is contained in Q_1. For simple conjunctive queries, existence of a query homomorphism is a characterizing condition for query containment. For conjunctive queries in general this also holds when applying linearisation [M92].

A query Q_1 is a *subquery* of a query Q_2, if Q_1 and Q_2 have the same head and the body of Q_1 is a subset of the body of Q_2.

A conjunctive query is *minimal*, if no equivalent subquery exists with its body being a strict subset of the original query's body.

Relational algebra [AHN95] is another notation for conjunctive queries that we employ occasionally. We use relational algebra mostly in the unnamed perspective, that is in particular, projections can be specified in terms of sets of positions of the argument expression. That is, if A is a set of positions in an expression E, then $\pi_A(E)$ is the projection on these positions.

3.2 Running Example

In this section we formalize the example of the administrative school database presented in chapter 2. As mentioned there, for simplicity we use only a small portion of the original schema.

In our example database schema, there exist the following five relations:

class(<u>level</u>, <u>code</u>, primary_language)

student(<u>name</u>, level, code)

person(name, gender)

language(language)

language_attendance(name, course)

The underlined attributes form the *key* of each relation. Furthermore, the *student.name* is a *foreign key* referring to *person.name*, *(student.level, student.code)* together to *(class.level, class.code)*, *language_attendance.course* to *language.language* and *language_attendance.name* to *person.name*. Figure 3.1 shows how the tables are linked by foreign keys.

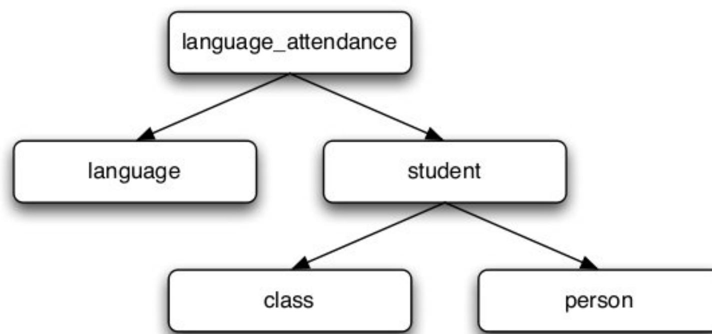

Figure 3.1: Example database structure

3.3 Problem-specific Definitions

In this section, we provide additional definitions that are needed for our treatment of incomplete databases and completeness reasoning. Some of these definitions are adapted from [Le96], some are our own work.

The first and very basic concept is that of a *partially complete database* (from now on just partial database). A database can only be incomplete with respect to another database that is considered to be complete. So we model partial databases as pairs of database instances: One database instance that describes the complete state, and one instance that describes the actual, possibly incomplete state.

To make this formal, we first introduce for each relation symbol R distinct symbols $\hat{R}$, $\check{R}$, which we call as the *ideal* and the *available* version of R. Then we introduce the schemas $\hat{\Sigma} = \{\hat{R} \mid R \in \Sigma\}$ and $\check{\Sigma} = \{\check{R} \mid R \in \Sigma\}$ which we call the *ideal database* and the *available database*. We denote the union $\hat{\Sigma} \cup \check{\Sigma}$ as $\tilde{\Sigma}$.

A *partial database instance* over Σ is a database instance D of $\tilde{\Sigma}$ such that $\check{R}(D) \subseteq \hat{R}(D)$ for all $R \in \Sigma$. That means, a partial database instance is a tuple $(\check{D}, \hat{D})$ of database instances of the extended schema $\tilde{\Sigma}$ such that the available database instance $\check{D}$ contains at most as much as the ideal database instance $\hat{D}$.

Next, we define the statements that are used to express completeness over partial databases. We start with the general concept of completeness statements, and as specialisations we define statements to express query completeness and local completeness.

A *completeness statement* is a statement of the form $Q_1 \dot{\subseteq} Q_2$ where Q_1 and Q_2 are conjunctive queries over the schema $\tilde{\Sigma}$, both with the same arity.

A partial database instance D *satisfies* the completeness statement $Q_1 \dot{\subseteq} Q_2$, denoted as $D \models Q_1 \dot{\subseteq} Q_2$, if $Q_1(D) \subseteq Q_2(D)$.

For a conjunctive condition G over Σ, we denote as $\hat{G}$ the condition obtained by replacing each relation symbol R with the symbol $\hat{R}$. For instance, if $G = \mathit{class}(X, Y)$, then $\hat{G} = \widehat{\mathit{class}}(X, Y)$. The condition $\check{G}$ is defined analogously.

If $Q(\bar{x}) :- G$ is a query, then the queries $\hat{Q}$, $\check{Q}$ are defined as $\hat{Q}(\bar{x}) :- \hat{G}$ and $\check{Q}(\bar{x}) :- \check{G}$.

For a conjunctive query Q, the *query completeness statement* for Q consists of the two completeness statements $\hat{Q} \dot{\subseteq} \check{Q}$ and $\check{Q} \dot{\subseteq} \hat{Q}$. We denote the query completeness statement for Q as *Compl*(Q).

A query completeness statement allows to express that a certain query (or view) is complete over a partial database.

Next we define the local completeness statement for relation R and condition G. A local completeness statement allows one to express that a certain part of relation R is complete, without requiring completeness of other parts of the database.

Note that G can contain relational and built-in literals and that we do not make any safety assumptions for G alone. Let $\bar{X}$ be the set of attributes of R. Let $Q_{\hat{R},\hat{G}}$ denote the query $Q_{\hat{R},\hat{G}}(\bar{X}) :- \hat{R}(\bar{X}), \hat{G}(\bar{X})$. Note that the entire query has to be safe. The *local completeness statement* for relation R with condition G then is the completeness statement $Q_{\hat{R},\hat{G}} \dot{\subseteq} \check{R}$, which we denote as *Compl*(R, G). Later on, we will refer to the query $Q_{\hat{R},\hat{G}}$ for a given local completeness statement C simply as $\hat{C}$.

To make clear where the difference between stating completeness of a relation by a query completeness statement and by a local completeness statement lies, consider the following example:

Example 3.1. Suppose we want to express that in a partial database instance D, the table *student* is complete for all female students. With the help of the foreign key from *student.name* to *person.name*, one would formulate the following query:

$$Q(N, L, C) :- \mathit{student}(N, L, C), \mathit{person}(N, \mathit{female})$$

Using this query, one would express the completeness as view completeness statement C_V as

$$C_V = (\hat{Q} \dot{\subseteq} \check{Q}).$$

As local completeness statement C_{LC}, one would formulate

$$C_{LC} = \mathit{Compl}(\mathit{student}(N, L, C), \mathit{person}(N, \mathit{female})).$$

The difference between C_V and C_{LC} now is that for C_V to hold in the partial database instance D, also the table *person* must satisfy a certain level of completeness. For every tuple of a female student in the ideal *student* relation, not only the *student* tuple must be in the available *student* relation, but also the corresponding *person* tuple in the available *person* table.

In a partial database instance D_0 where

$$\begin{aligned}
\hat{\mathit{student}} &= \{\, (\mathit{antonia}, 3, a, \mathit{false}) \,\} \\
\hat{\mathit{person}} &= \{\, (\mathit{antonia}, \mathit{female}) \,\} \\
\check{\mathit{student}} &= \{\, (\mathit{antonia}, 3, a, \mathit{false}) \,\} \\
\check{\mathit{person}} &= \{\ \},
\end{aligned}$$

C_{LC} would hold but C_V not. □

Given a query Q and a set of local completeness assertions $\mathcal{C}$, we say that $\mathcal{C}$ is *characterizing* for Q provided $D \models \mathcal{C}$ if and only if $D \models \mathit{Compl}(Q)$, for all partial database instances D. Thus, a characterizing set of local completeness assertions is a set, whose satisfaction is both a necessary and sufficient condition for query completeness.

A set of local completeness assertions $\mathcal{C}$ entails another set $\mathcal{C}'$, denoted $\mathcal{C} \models \mathcal{C}'$, if and only if for all partial database instances D we have that $D \models \mathcal{C}$ implies $D \models \mathcal{C}'$.

Our last definition is that of canonical completeness statements. Let $Q(\bar{Z}) :- G$ be a query, $L(\bar{Y})$ be a literal in G, R be the relation symbol of L, and $G' = G \setminus \{L\}$. The *canonical completeness condition* for L in Q is the local completeness condition $\mathit{Compl}(R(\bar{X}), G', \bar{X} = \bar{Y})$, where the variables in $\bar{X}$ are new distinct variables. The set of canonical completeness conditions for all literals in Q we denote by $\mathcal{C}_Q$.

Example 3.2. Consider a partial database instance D where

$$\begin{aligned} \hat{student} &= \{ (matteo, 3, a), (sonja, 2, b) \} \\ \check{student} &= \{ (matteo, 3, a) \} \\ \hat{class} = \check{class} &= \{ (3, a, it), (2, b, ger) \}. \end{aligned}$$

Furthermore, let Q be the query

$$Q(N) :- student(N, L, C), class(L, C, P), P = it$$

that asks for the names of all the students that are enrolled in a class with Italian as primary language. Then, consider local completeness statements

$$\begin{aligned} C_1 &= Compl(student(N, L, C), C = a) \\ C_2 &= Compl(class(L, C, P), True). \end{aligned}$$

The completeness statement C_1 holds in D, because we find all students that are in the ideal instance of the *student* relation with class code a also in the available instance. However, C_2 does not hold in D, because $(sonja, 2, b)$ is in the ideal class relation, but not in the available one.

The canonical completeness conditions for Q are

$$\begin{aligned} C_3 &= Compl(student(N, L, C), class(L, C, P), P = it) \\ C_4 &= Compl(class(L, C, P), student(N, L, C), P = it) \end{aligned}$$

The query completeness statement $Compl(Q)$ holds in D, because $Q(\hat{D}) = Q(\check{D}) = \{ matteo \}$.

Finally, observe that C_2 entails C_4, because whenever in any partial database instance the class table is complete for all classes, it is also complete for all classes with Italian as primary language. □

Chapter 4

Characterizing Query Completeness

Given a query over a fixed database schema, an important question is what the necessary and the sufficient conditions for query completeness are, and how they can be expressed. Necessary conditions are conditions that must hold in a partial database instance where the query is complete, while sufficient conditions are conditions that whenever satisfied in a partial database instance, imply query completeness over that partial database instance.

The conditions depend on the semantics of queries, therefore we distinguish between set- and multiset semantics. Also, expressibility of conditions depends on the language chosen for expressing them, therefore we try to express the conditions both by local completeness statements and in first-order logic.

While local completeness statements seem to be more of practical interest because of their better understandability, we will show that there exist cases where necessary and sufficient conditions cannot be expressed by local completeness statements, but only in first-order logic.

We remind the reader that we call conditions that are necessary and sufficient for query completeness *characterizing conditions.*

Example 4.1. To get a basic idea of the problem, consider the following query asking for all classes in which there is some student taking Latin courses:

$$Q(L, C, P) :- \mathit{lang_att}(N, \mathit{Lng}), \mathit{student}(N, L, C), \mathit{class}(L, C, P), \mathit{Lng} = \mathit{latin}.$$

It is clear that completeness of the *person* table has no influence on the completeness of the query. Whether or not person tuples present in the ideal relation are missing in the available one, cannot have any influence on query completeness as that relation is not even used in the query.

But completeness of the *class* table has an influence on the completeness of the query. If a class tuple is missing in the available database it might be one where someone takes Latin, so the query becomes incomplete. What does this mean? Not the whole *class*table has to be complete, only tuples for those classes must be present where actually someone takes Latin. □

We will come back to this example later.

4.1 Multiset Semantics

A multiset (or bag) is distinguished from a set in that an element can occur multiple times in a multiset while it can occur at most once in a set. Accordingly, under multiset semantics, a query returns a multiset of tuples while under set semantics it returns a set of tuples. However, we consider relation extensions to be sets only. This semantics, sometimes also referred to as *bagset* semantics, are the most often used in practice. In SQL for example, queries are always evaluated under multiset semantics, except when the DISTINCT keyword is used. Multiset semantics are also essential for aggregate queries with the operators *sum*, *count* or *avg*.

As sketched in the introduction to this chapter, all parts of the database that may contribute to the query result have to be complete for query completeness.

With the following theorem we show that these parts are precisely described by the canonical completeness conditions, and therefore the canonical completeness conditions conditions $\mathcal{C}_Q$ of a query Q are characterizing conditions for query completeness under multiset semantics.

Theorem 4.2. *Let Q be a conjunctive query and D be a partial database instance. Then, the following two are equivalent:*

1. *Q is complete over D under multiset semantics*
2. *$D \models \mathcal{C}_Q$.*

Proof "$\Rightarrow$" Indirect proof: Suppose, one of the completeness assertions in $\mathcal{C}_Q$ does not hold over D, for instance, assertion C_1 for literal L_1. Suppose, R_1 is the relation symbol of literal L_1. Let C_1 stand for the completeness statement $\hat{C}_1 \dot{\subseteq} \check{R}_1$. Then $\hat{C}_1(D) \not\subseteq \check{R}_1(D)$. Let t be a tuple that is in $\hat{C}_1(D)$ and therefore in $\hat{R}_1(D)$ but not in $\check{R}_1(D)$. By the fact that $\hat{C}_1$ has the same body as Q, the valuation v of $\hat{C}_1$ over D that yields t is also a valuation for $\hat{Q}$ over D. So we find one multiplicity of some tuple $t' \in \hat{Q}(D)$, where t' is v applied to the distinguished variables of Q.

However, v does not satisfy $\check{Q}$ over D because t is not in $\check{R}_1(D)$. By the monotonicity of conjunctive queries, we cannot have another valuation yielding t'

over $\check{D}$ but not over $\hat{D}$. Therefore, $\check{Q}(D)$ contains at least one multiplicity of t' less than $\hat{Q}(D)$, and hence Q is not complete over D.

"$\Leftarrow$" Direct proof: We have to show that if t is n times in $\hat{Q}(D)$ then t is also n times in $\check{Q}(D)$. For every multiplicity of t in $\hat{Q}(D)$ we have a valuation of the variables of Q which is satisfying over $\hat{D}$. We show that if a valuation is satisfying for Q over $\hat{D}$, then it is also satisfying for Q over $\check{D}$. A valuation for a conjunctive condition G is satisfying over a database instance if we find all implied ground atoms of G in that instance. If a valuation satisfies Q over $\hat{D}$, then we will find all implied ground atoms also in $\check{D}$, because the canonical completeness conditions hold in D by assumption. Satisfaction of the canonical completeness conditions requires that for every satisfying valuation of Q, for every literal, the implied ground atom is in $\check{D}$. Therefore, each valuation for a tuple t on Q over $\hat{D}$ is also a valuation over $\check{D}$ and hence Q is complete over D. □

4.2 Set Semantics

Under *set semantics*, relation extensions and results of queries are always sets. That is, if a tuple were in a query's result several times under multiset semantics, it is in the query result one time under set semantics. As we show in this section, under set semantics satisfaction of the canonical completeness statements is not a necessary conditions for query completeness in general. But it is still a sufficient conditions, and necessary in the special case of projection-free queries.

A query is *projection free*, if all variables that appear in its body are distinguished variables. We remind that queries are evaluated under set semantics now.

Theorem 4.3. *Let Q be a conjunctive query and D be a partial database instance. Then:*

1. $D \models \mathcal{C}_Q$ *implies* $D \models$ *Compl*(Q).

2. $D \models$ *Compl*(Q) *implies* $D \models \mathcal{C}_Q$*, provided Q is projection free.*

Proof 1. Follows from Theorem 4.2. When a query is complete under multiset semantics, it is also complete under set semantics.

2. Follows from Theorem 4.2. Under multiset semantics, violation of a necessary completeness assertion leads to a difference of at least one multiplicity of some tuple in the results over $\hat{D}$ and $\check{D}$. Under set semantics multiplicities collapse so the query could be still complete, if there existed another way to compute that tuple in the result. However, observe, that without disjunction and projection, under set semantics, there exists only exactly one valuation per tuple in the result. □

The next example illustrates that satisfaction of the canonical completeness conditions is not a necessary condition for query completeness.

Example 4.4. Consider again our query asking for all classes where some student takes Latin. Consider an ideal database instance where there is only one class with 20 students, all of them taking Latin.

The canonical completeness conditions would require that all these 20 students are in the available version of the *student* table, and for all of them their Latin course attendance is present in the available *language_attendance* table.

However, it is only necessary to have one of the 20 students together with his/her Latin attendance in the available database, in order to get the complete answer that in this class someone takes Latin. □

The next theorem expresses the observation from above in a formal way.

Theorem 4.5. *Let Q be a conjunctive query where at least one variable in the body is not a distinguished variable. Then, no set of local completeness statements exists such that satisfaction of it is a characterizing condition for query completeness of Q.*

Proof We show the nonexistence of a characterizing set of local completeness statements for a simple query first, and describe afterwards, how this proof extends to arbitrary queries.

Assume a relation schema $\Sigma = \{ R/1 \}$ and a boolean query $Q() :- R(X)$. Furthermore, assume a characterizing set of local completeness conditions $\mathcal{C}$ for Q existed. Now consider the partial database instances D_1, D_2 and D_3 such that:

$$\hat{D}_1 = \{ \hat{R}(a), \hat{R}(b) \} \qquad \check{D}_1 = \{ \check{R}(a) \}$$
$$\hat{D}_2 = \{ \hat{R}(a), \hat{R}(b) \} \qquad \check{D}_2 = \{ \check{R}(b) \}$$
$$\hat{D}_3 = \{ \hat{R}(a), \hat{R}(b) \} \qquad \check{D}_3 = \{ \}$$

Then, $Compl(Q)$ holds in D_1 and D_2 but not in D_3, and therefore all local completeness conditions in $\mathcal{C}$ have to hold in D_1 and D_2, but at least one of them must not hold in D_3. Let us call that condition C.

The statement C must be of the form $Compl(R(X), G)$. Then $G = \top$ does not hold in D_1 and D_2 (because in both cases there is a tuple in $\hat{R}$ that is not in $\check{R}$). Other relation symbols to introduce do not exist and repeating R with a variable generates only equivalent conditions. Adding an equality atom for x with some constant generates a local completeness statement that does not hold either in D_1 or D_2. So the only form G can have such that $Compl(R(X), G)$ holds in D_1 and

D_2 is $G = \bot$. However, $Compl(R(x), \bot)$ holds in D_3 as well.

The proof for this specific query can be extended to any query with projection. The idea is to construct three partial database instances, where the ideal database instances contain the frozen body of the query plus an isomorphic structure differing only in a nondistinguished variable's name. The three available database instances are once the frozen body, once the isomorphic structure differing in a nondistinguished variable's name, and once the empty set. If the completeness statements cannot detect that in the first two instances once the frozen body and once the isomorphic structure is missing, they will not detect that in the third instance both are missing. But over the third instance, the query is clearly incomplete.

Let Q be a conjunctive query with projection, so suppose X is a variable that appears in the body of Q and is nondistinguished. Let Y be a new symbol not appearing in Q and let $B(Q)$ denote the frozen body of Q. Furthermore let $[X/Y]$ describe the operation of replacing every occurance of symbol X by symbol Y. Then, the three instances are as follows:

$$\hat{D}_1 = B(Q) \cup B(Q)[X/Y] \qquad \check{D}_1 = B(Q)$$
$$\hat{D}_2 = B(Q) \cup B(Q)[X/Y] \qquad \check{D}_2 = B(Q)[X/Y]$$
$$\hat{D}_3 = B(Q) \cup B(Q)[X/Y] \qquad \check{D}_3 = \{\ \}$$

Suppose there exists a set of completeness statements $\mathcal{C}$, satisfaction of which is a characterizing condition for query completeness of Q. Observe that in order to be a necessary condition, $\mathcal{C}$ must not require more completeness than needed for the completeness of Q, that is, there must exist a homomorphism from the body of each statement in $\mathcal{C}$ to the body of Q. Furthermore, in order to be a sufficient condition, there must exist at least one statement for each relation appearing in $B(Q)$.

Let $R(\bar{s})$ be an atom appearing in $B(Q)$ that contains the variable X, and t be the tuple that is the argument of $R(\bar{s})$ where all variables are interpreted as constants. Let $t_{[X/Y]}$ be the modification of t where X is replaced by Y.

Observe that there has to exist a local completeness statement C over relation R such that $t \in \hat{C}(D_1)$. Because if no such statement C existed, then for a Q that is minimal the partial database instance $D' = (\hat{B}(Q), \check{B}(Q) \setminus R(t))$ would satisfy $\mathcal{C}$ but Q would be incomplete over it. For a non-minimal Q also such a partial database instance can be constructed, however one has to distinguish whether $R(\bar{s})$ is a redundant atom or not.

So there exists a C such that $t \in \hat{C}(D_1)$. As $B(Q)$ and $B(Q)[X/Y]$ are isomorphic, either $t_{[X/Y]}$ also has to be in $\hat{C}(D_1)$, which would imply that C does not

hold in D_1 and hence $\mathcal{C}$ is not a necessary set for query completeness of Q. Or, C has to contain the constant X at some position such that $B(Q)[X/Y]$ does not satisfy C, but then C does not hold in D_2, and again, $\mathcal{C}$ would not be a necessary set for query completeness of Q. □

The preceding theorem states that it is impossible to express characterizing conditions for query completeness under set semantics with local completeness statements.

One may ask whether it is possible at all to characterize these conditions.

The next proposition shows that with a first order formula we can characterize those partial database instances such that a fixed query is complete over them.

Proposition 4.6. *Let $Q(\bar{X}) :- B(\bar{X}, \bar{Y})$ be a conjunctive query with projection. Let $\phi_{\bar{X},B}$ denote the formula $\forall \bar{X}\ (\exists \bar{Y} \hat{B}(\bar{X}, \bar{Y}) \rightarrow \exists \bar{Z}\ \check{B}(\bar{X}, \bar{Z}))$. Then for all partial database instances D:*

$$D \models \mathit{Compl}(Q) \text{ if and only if } D \models \phi_{\bar{X},B}$$

The proposition above is not very surprising, as it is just a rewriting of the definition of query completeness.

Chapter 5

Completeness Reasoning

In this section we discuss different reasoning tasks regarding completeness. The main result of this chapter is that deciding query completeness under given local completeness statements can be reduced to query containment, and is therefore decidable if containment for the class of queries is.

The reasoning tasks we present are 1) inferring local completeness from local completeness, 2) inferring query completeness from local completeness, 3) inferring query completeness from query completeness and 4) inferring local completeness from query completeness. For the first two problems we will provide precise solutions by reductions to decidable problems, while for the last two problems, we only provide correct but possibly incomplete solutions.

For the problem of local completeness entailed by local completeness, no discussion exists in the literature. We will reduce the problem to the problem of query containment in such a way that the complexity results from query containmet carry over to our problem.

The main reasoning task of inferring query completeness from local completeness was first discussed by Levy [Le96]. His reduction of the problem led to a generally undecidable problem. We present a better solution by reducing the problem to the problem of inferring local completeness from local completeness, allowing to carry over the complexity results from that problem.

The problem of inferring query completeness from query completeness was first discussed by Motro [Mo89]. He presented a solution by reducing it to the problem of conjunctive-query rewritability. We will show two open issues regarding that approach, first, that the decidability of the existence of a more general class of rewritings is not yet solved [SV05], second, that it is not clear whether existence of a rewriting is a necessary condition for query completeness implying query completeness.

For inference of local completeness from query completeness, we give a correct

but possibly incomplete solution by a reduction to the previous problem.

5.1 Inferring Local Completeness from Local Completeness

As stated in the introduction of this chapter, deduction of local completeness statements from local completeness statements can be reduced to query containment.

As an intuitive idea, note that local completeness statements describe views of relations that should be complete. So when a local completeness statement entails another statement, the view described by the second statement should be a subset of the view described by the first statement. Inclusion of views corresponds to the problem of containment, so the problem can easily be reduced to query containment.

We remind the reader that for a local completeness statement C over a relation R, the query $\hat{C}$ is the query that selects all the tuples from an ideal relation extension $\hat{R}(D)$ that also have to be in the available relation extension $\check{R}(D)$ in a partial database instance D, to make C satisfied over D.

Lemma 5.1. *Let C_1 to C_n and C be local completeness statements over a relation R. Then*

$$C_1 \wedge \ldots \wedge C_n \models C \text{ if and only if } \hat{C} \subseteq \hat{C}_1 \cup \ldots \cup \hat{C}_n.$$

Proof "$\Leftarrow$" Let $\hat{C} \subseteq \hat{C}_1 \cup .. \cup \hat{C}_n$ and let D be a partial database instance where C_1 to C_n hold. We have to show that C holds in D as well. Let t be a tuple in $\hat{C}(D)$. We have to show that t is also in $\check{R}(D)$.

As $\hat{C}$ is contained in $\hat{C}_1 \cup \ldots \cup \hat{C}_n$, the tuple t is also in $\hat{C}_1(D) \cup \ldots \cup \hat{C}_n(D)$. As C_1 to C_n hold in D, we have that t is also in $\check{R}(D)$.

"$\Rightarrow$" When $\hat{C}$ is not contained in $\hat{C}_1 \cup .. \cup \hat{C}_n$, there exists a database instance D such that there is a tuple t in $\hat{C}(D)$ that is not in $\hat{C}_1 \cup .. \cup \hat{C}_n$. We construct a partial database instance $D_0 = (\check{D} \setminus \{ \check{R}(t) \}, \hat{D})$ from it where ideal and available database are exactly the same except that $\check{R}(t)$ is not in the available database. As t is not in $\hat{C}_1(D_0) \cup .. \cup \hat{C}_n(D_0)$, C_1 to C_n hold on this partial database instance, whereas C does not. $\square$

With this reduction, we get the complexity of containment reasoning as an upper bound for the complexity of local completeness entailment. To find out a lower bound for the complexity of containment reasoning, we have to show that query containment problems for unions of queries can be reduced to local completeness entailment problems.

Lemma 5.2. *Let* $Q(\bar{X}) \,{:}\!-\, B$ *and* $Q_1(\bar{X}_1) \,{:}\!-\, B_1, \ldots, Q_n(\bar{X}_n) \,{:}\!-\, B_n$ *be conjunctive queries with the same arity. Let* R *be a new relation symbol with same the same arity as the queries. Let* C *and* C_1 *to* C_n *be local completeness statements such that* $C_i = Compl(R(\bar{X}_i), B_i)$. *Then*

$$Q \subseteq Q_1 \cup \ldots \cup Q_n \text{ if and only if } C_1 \wedge \ldots \wedge C_n \models C.$$

Proof "⇒" Let $Q \subseteq Q_1 \cup \ldots \cup Q_n$. We have to show that in every partial database instance D where C_1 to C_n hold, also C holds. That is, we have to show that whenever a tuple t is in $\hat{C}(D)$, it is also in $\check{R}$.

Suppose in some partial database instance D there is a tuple t in $\hat{C}(D)$. Then, it is also in $\hat{R}$. Furthermore, as $\hat{C}$ has the same body as Q except of the additional literal with relation symbol R, the tuple t must also be in $\hat{Q}(D)$. By the containment, t then is also in $\hat{Q}_1(D) \cup \ldots \cup \hat{Q}_n(D)$. By that and the fact that t is in $\hat{R}$, we find that t is also in $\hat{C}_1(D) \cup \ldots \cup \hat{C}_n(D)$. As C_1 to C_n hold in D, the tuple t then is also in $\check{R}$.

"⇐" Assume $Q \not\subseteq Q_1 \cup \ldots \cup Q_n$. We have to show that $C_1 \wedge \ldots \wedge C_n \not\models C$.

If $Q \not\subseteq Q_1 \cup \ldots \cup Q_n$, then there exists a database instance D such that there is a tuple t which is in $Q(D)$ but not in $Q_1(D) \cup \ldots \cup Q_n(D)$.

Let R_D denote the sets of all atoms with relation symbol R where the argument is in $Q(D) \cup Q_1(D) \cup \ldots \cup Q_n(D)$.

We construct a partial database instance $D_0 = (\check{D}_0, \hat{D}_0)$ out of D, where $\hat{D}_0 = D \cup R_D$ and $\check{D}_0 = D \cup R_D \backslash \{\, R(t) \,\}$. That is, ideal and available database contain both D, and both all R atoms for tuples which were in the answer of one of the queries over D, except that $\check{R}(t)$ is not in $\check{D}_0$.

Over D_0, the completeness statements C_1 to C_n hold because all tuples that were in $Q_1(D)$ to $Q_n(D)$ are also in $\check{R}$. However, the completeness statement C does not hold in D_0 because t is not in $\check{R}(D_0)$. □

Example 5.3. Consider the local completeness statements

$$C_1 = Compl(R(X), S(X, X))$$
$$C_2 = Compl(R(X), S(X, Y)).$$

The corresponding queries $\hat{C}_1$ and $\hat{C}_2$ are

$$\hat{C}_1(X) \,{:}\!-\, \hat{R}(X), \hat{S}(X, X)$$
$$\hat{C}_2(X) \,{:}\!-\, \hat{R}(X), \hat{S}(X, Y).$$

Observe, that $\hat{C}_1 \subseteq \hat{C}_2$, because in every database, a valuation that satisfies the symmetric atom $R(x, x)$ in $\hat{C}_1$, also satisfies the unrestricted one $R(x, y)$ in $\hat{C}_2$.

Therefore, satisfaction of C_2 implies holding of C_1. That is reasonable, because whenever in a partial database instance, R is complete for all tuples which have an arbitrary successor tuple in S, then R is also complete for all tuples which have a symmetric successor. □

Having that the problems of local completeness entailment and query containment can be reduced to each other, we can conclude that the problems have the same complexity.

Formally, for a class $\mathcal{Q}$ of conjunctive queries, the problem of *union containment* ($UC_{\mathcal{Q}}$) is the problem of whether a query $Q \in \mathcal{Q}$ is contained in a union of queries from $\mathcal{Q}$. The problem of *entailment of local completeness statements* ($ELCS_{\mathcal{Q}}$) is the problem of whether a set of local completeness statements using only conditions from $\mathcal{Q}$, entails another local completeness statement with a condition from $\mathcal{Q}$.

Theorem 5.4. *Let $\mathcal{Q}$ be a class of conjunctive queries that contains for every relation the identity and that is closed under intersection. Then the two problems $UC_{\mathcal{Q}}$ and $ELCS_{\mathcal{Q}}$ can be reduced to each other in polynomial time.*

Proof Follows from lemmas 5.1 and 5.2. □

Corollary 5.5. *The entailment problem of local completeness statements is*

- *NP-complete for simple conjunctive queries,*
- *polynomial for simple conjunctive queries without repeated relation symbols,*
- *piptwo-complete for each of the classes of conjunctive queries with comparisons and equations/disequations, respectively.*

5.2 Inferring Query Completeness from Local Completeness

In this section we give our solution to the question of query completeness entailed by local completeness statements. Instead of reducing it to the problem of query independence of updates, as done by Levy, we reduce it to the problem of local completeness statement entailment by local completeness statements, which was presented in the previous section.

By the following theorem, given a conjunctive query Q' and a set of local completeness statements $\mathcal{C}$, it suffices to test whether $\mathcal{C}$ entails $\mathcal{C}_Q$, where Q is a minimal version of Q'.

Theorem 5.6. *Let Q be a minimal conjunctive query and $\mathcal{C}$ be a set of local completeness statements. Then*

$$\mathcal{C} \models Compl(Q) \text{ if and only if } \mathcal{C} \models \mathcal{C}_Q$$

Proof "$\Rightarrow$" Proof by contradiction. Assume a minimal Q and a set $\mathcal{C}$ such that $\mathcal{C} \models Compl(Q)$, but $\mathcal{C} \not\models \mathcal{C}_Q$. Then, because $\mathcal{C} \not\models \mathcal{C}_Q$ there exists some partial database instance D such that $D \models \mathcal{C}$ but $D \not\models \mathcal{C}_Q$. When $D \not\models \mathcal{C}_Q$, we find that D violates some canonical completeness assertion in $\mathcal{C}_Q$. We assume that this is C_1, which means that $D \not\models C_1$. This implies that there exists some tuple t_1 such that $t_1 \in \hat{C}_1$ but $t_1 \notin \check{R}_1$.

Now we construct a second partial database $D_0 = (\check{B}(Q) \setminus \{ L_1 \}, \hat{B}(Q))$, where $B(Q)$ is the frozen body of Q and L_1 is the literal at the first position in the body of Q (for which the local completeness statement is C_1). We show that D_0 satisfies $\mathcal{C}$ as well. The only difference between $\hat{D}_0$ and $\check{D}_0$ is L_1, therefore all local completeness statements in $\mathcal{C}$ which describe local completeness for relations different from R_1 are satisfied immediately. To show that D_0 satisfies also all local completeness statements in $\mathcal{C}$ which describe local completeness for relation R_1, we assume the opposite and show that this leads to a contradiction.

Assume, D_0 does not satisfy some local completeness statement C in $\mathcal{C}$. Then, we must have that $\bar{X}_1 \in \hat{C}(D_0)$, where $\bar{X}_1$ are the variables in the literal L_1. Let $B(C)$ be the body of C. Then, $\bar{X}_1 \in \hat{C}(D_0)$ means that there must exist a valuation δ such that $\delta B(C) \subseteq B(Q)$ and $\delta \bar{X}_C = \bar{X}_1$, where $\bar{X}_C$ are the distinguished variables of C. As $t_1 \in \hat{C}_1(D)$ there exists another valuation θ such that $\theta B(Q) \subseteq \hat{D}$ and $\theta \bar{X}_1 = t_1$, where $\bar{X}_1$ are the variables in literal L_1. Putting θ and σ together, we would find that $\theta\sigma B(C) \subseteq \hat{D}$ and $\theta\sigma\bar{X}_C = t_1$. In other words, t_1 would be constrained by C over D. As t_1 is not in $\check{R}_1$, $\mathcal{C}$ would not hold in D, which was our assumption. Hence we can conclude that $\bar{X}_1$ can not be in $\hat{C}(D_0)$ and therefore D_0 satisfies $\mathcal{C}$ whenever D satisfies it.

Now assumed that D_0 satisfies $\mathcal{C}$, Q is complete over D_0 because we assumed that $\mathcal{C} \models Compl(Q)$. As $\hat{D}_0 = B(Q)$, we find that $\bar{X} \in \hat{Q}(D_0)$, with $\bar{X}$ being the distinguished variables of Q. As Q is supposed to be complete over D_0, $\bar{X}$ should also be in $\check{Q}(D_0)$. As $\hat{D}_0 = B(Q) \setminus \{ L_1 \}$, this would require a mapping from $B(Q)$ to $B(Q) \setminus \{ L_1 \}$ which is the identity on the distinguished variables of Q. Then, this mapping would be a non surjective homomorphism from $B(Q)$ to $B(Q)$ and hence Q would not be minimal.

"$\Leftarrow$" Follows from theorem 4.3(2). For a query Q, when its projection free variant Q' is complete for all partial databases that satisfy $\mathcal{C}_{Q'}$, that means that its result over the available and the ideal database of that partial databases is the same. When two sets are the same, also the sets derived by projections on them are the same. □

With the next example, we show that query minimality is really a necessary condition for the theorem given above to hold.

Example 5.7. Consider the query $Q(X) :- R(X, X), R(X, Y)$. It is not minimal, because there exists a homomorphism, mapping y to x that allows to drop the second literal. Its characterizing completeness statements are

- $C_1 = Compl(R(X, X), R(X, Y))$
- $C_2 = Compl(R(X, Y), R(X, X))$.

The statement C_2 entails the statement C_1. However, Q is complete already whenever C_1 holds. This follows from the fact that whenever we have a symmetric tuple which satisfies the first literal in Q, the same tuple satisfies also the second literal without influence on the query result. Hence, having any other tuples specified by C_2 is not needed, and hence holding of C_1 and C_2 is not a necessary precondition for query completeness of Q. □

Using this reduction to the problem of local completeness entailment, which is equivalent to the problem of query containment, we can carry over complexity results from query containment to determining query completeness from local completeness with our approach as follows:

- Polynomial: Simple conjunctive queries without repeated relation symbols, simple conjunctive local completeness statements (that is, both, queries and completeness statements are without comparison).
- NP-complete:
  - Simple conjunctive queries without repeated relation symbols, conjunctive completeness statements (that is, comparison in the completeness statements only).
  - Conjunctive queries, simple conjunctive local completeness statements (that is, comparison only in the query).
- Π_2^P-complete: Conjunctive queries, conjunctive local completeness statements (that is, comparison in both the query and the completeness statements).

5.3 Inference from Query Completeness

In addition to inferring completeness from local completeness, there is the question of how to infer completeness statements from the completeness of queries. Again, one can consider deducing query completeness or local completeness.

While the second question has not been approached so far, the first has been discussed by Amihai Motro in [Mo89]. There, Motro gave a correct but incomplete algorithm for deducing query completeness from query completeness, based on query rewriting by views. Since then, no more work on that question has been published. However, in 2005 a paper has been published by Segoufin and Vianu, that lays the theoretical foundations for why this question is difficult [SV05]. The work shows the difficulty of deciding query determinacy, a problem very close to the problem of inferring query completeness from query completeness, and leaves its decidability open. In section 5.3.1, we show how our inference problem is related to the problem of query determinacy.

In section 5.3.2, we discuss the second problem of deducing local completeness from query completeness. We will give a correct but incomplete reduction to the problem of query completeness entailed by query completeness.

Although from our investigations it seems that both problems are hard, we have not been able to come up with a conclusive answer and future work on the topic is needed.

5.3.1 Inferring Query Completeness from Query Completeness

Inferring query completeness from query completeness is the problem of whether over all partial databases completeness of a set of queries implies completeness of a certain query. We remind the reader that partial databases are pairs of database instances where the second instance is a subset of the first.

The question is of practical interest because query completeness statements are an interesting alternative for stating database completeness instead of local completeness statements, as first considered by Motro in [Mo89].

A related problem is the problem of query determinacy. Formally, a set of queries $\mathcal{Q}$ *determines* a query Q, if $\mathcal{Q}(D_1) = \mathcal{Q}(D_2)$ implies $Q(D_1) = Q(D_2)$ on all database instances D_1 and D_2. We note that $\mathcal{Q}$ determines Q by $\mathcal{Q} \twoheadrightarrow Q$. The following proposition expresses the obvious relation between query determinacy and query completeness deduction.

Proposition 5.8. *Let $\mathcal{Q}$ be a set of views, Q be a query.*

$$Compl(\mathcal{Q}) \rightarrow Compl(Q) \quad if \quad \mathcal{Q} \twoheadrightarrow Q.$$

Proof Obvious. Whenever the answer to $\mathcal{Q}$ being the same implies the answer to Q being the same over arbitrary pairs of database instances, that also holds over the restricted set of pairs of database instances where one is contained in the other. □

Whether the entailment holds also for the other direction, which would yield that the problems of query determinacy and query completeness deduction are equivalent, we could not identify.

Whether query determinacy is decidable, is an open problem so far.

If the problems could be shown to be equivalent, this would not give an immediate insight, but would tell at least that the problem of query completeness deduction is equally hard. If some solution were found about decidability of query determinacy in the future, it would then apply to query completeness deduction as well. Finally, showing equivalence might be helpful for deciding decidability of query determinacy itself, as it would tell that it is sufficient to only consider pairs of database instances where one is a subset of the other.

If the problems were shown to be not equivalent, there would exist the possibility that the problem of query completeness inference has an easier solution than that of query determinacy.

We tried to get an insight in the question of whether the two problems are equivalent by considering the following three classes of simple conjunctive queries (simple conjunctive omitted from here on): boolean queries, monadic queries and unrestricted queries.

Boolean Queries

For boolean queries, we can give an exact characterization of when a query is determined by a set of queries that is the same for when completeness of a set of queries implies completeness of a query. Thus, the two problems are equivalent for boolean queries. Let $\doteq$ denote the semantic equivalence operator extended to sets of boolean queries, such that $\mathcal{Q} \doteq Q$ if and only if $Q' \equiv Q$, where $Q'() :- \bigcup_{q \in \mathcal{Q}} B(q)$, w.l.o.g. under the assumption that no variable name appears in more than one body of the queries in $\mathcal{Q}$.

Theorem 5.9. *Let Q be a simple conjunctive boolean query and let $\mathcal{Q}$ be a finite set of such queries. Then*

$$\mathcal{Q} \twoheadrightarrow Q \iff \mathit{Compl}(\mathcal{Q}) \rightarrow \mathit{Compl}(Q) \iff \exists S : S \subseteq \mathcal{Q} \;\wedge\; S \doteq Q.$$

Proof For readability, let us abbreviate the theorem above by "1" ⇔ "2" ⇔ "3". As "1" implies "2" is known already from proposition 5.8, we only need to show "3" ⇒ "1" and "2" ⇒ "3".

"3" ⇒ "1". That is, $\exists S : S \subseteq \mathcal{Q} \;\wedge\; S \doteq Q$ implies $\mathcal{Q} \twoheadrightarrow Q$. Obvious.

As Q is equivalent to some S which is subset of $\mathcal{Q}$, whenever $\mathcal{Q}(D_1) = \mathcal{Q}(D_2)$, also $S(D_1) = S(D_2)$ and hence $Q(D_1) = Q(D_2)$.

"2" ⇒ "3". That is, $\mathit{Compl}(\mathcal{Q}) \rightarrow \mathit{Compl}(Q)$ implies $\exists S : S \subseteq \mathcal{Q} \;\wedge\; S \doteq Q$. We show it by contradiction.

Assume, $\neg\exists S : S \subseteq \mathcal{Q} \ \wedge \ S \doteqdot Q$. We have to show, that *Compl*$(\mathcal{Q}) \not\rightarrow$ *Compl*(Q). Let $\mathcal{Q}_{in}$ denote the set of all $q \in \mathcal{Q}$, such that q is more general than Q (i.e., there exists a homomorphism from q to Q). As $\neg\exists S : S \subseteq \mathcal{Q} \ \wedge \ S \doteqdot Q$, $\mathcal{Q}_{in}$ can not be equivalent to Q. W.l.o.g. we assume that the variable names used in Q and the queries in $\mathcal{Q}$ are pairwise distinct.

Now, consider a partial database instance D, where $\hat{D}$ contains exactly the frozen body of Q and the frozen bodies of the queries in $\mathcal{Q}_{in}$, and $\check{D}$ the frozen bodies of the queries in $\mathcal{Q}_{in}$.

Over $\hat{D}$, exactly the queries in $\mathcal{Q}_{in}$ return true, because for them homomorphisms to Q existed. No other queries return true, because the additional structures in $\hat{D}$ are homomorphic to the frozen body of Q. The queries in $\mathcal{Q}_{in}$ also return true over $\check{D}$, hence we find that $\mathcal{Q}(\hat{D}) = \mathcal{Q}(\check{D}) = \{\, ()\, \}$. But $Q(\hat{D}) = \{\, ()\, \}$ and $Q(\check{D}) = \emptyset$, and hence, completeness of $\mathcal{Q}$ does not imply completeness of Q. □

Monadic Queries

Monadic queries are queries with exactly one distinguished variable. For monadic queries, the determinacy problem is decidable.

This holds because for monadic views and queries, conjunctive queries are a complete rewriting language. That is, whenever a monadic conjunctive query can be rewritten in terms of monadic views, the rewriting can be expressed as a conjunctive query. Existence of some rewriting is characterizing for query determinacy, and existence of conjunctive rewritings is decidable.

Thus, if we were able to show that the existence of a conjunctive rewriting is a necessary precondition for query completeness following from query completeness, it would follow that the two problems are equivalent on the class of monadic queries.

The proof of existence of a conjunctive rewriting as necessary precondition for query determinacy shown in [SV05] makes use of a construction of a pair of database instances, where the one is not included in the other, and hence does not trivially apply to our case of pairs of partial database instances, where the one is included in the other. We did not investigate further.

Unrestricted Queries

In the unrestricted case, decidability of query determinacy for conjunctive queries is an open question. It is known that existence of a conjunctive rewriting, which is decidable, is not a necessary precondition. Conversely, existence of a rewriting in second-order logic is a necessary precondition, however, this is in general not decidable. It might be the case, still, that the necessary rewritings belong to

a decidable fragment of second-order logic (what Segoufin and Vianu found out already is that $\exists SO \cap \forall SO$ is an upper bound for the expressivity of the language in which the rewritings have to be).

It is an open question whether entailment of query completeness and query determinacy are equivalent for arbitrary conjunctive queries.

An attempt for proofing equivalence of the problems that did not succeed is enclosed in the appendix of this thesis.

5.3.2 Inferring Local Completeness from Query Completeness

For the problem of whether a local completeness statement follows from a query completeness statement we can give a sufficient condition. The condition is that the query is projecting on the attributes of the relation over which the local completeness statement is, and is equal or more general. Whether this condition is also a necessary condition for the entailment to hold, remains an open question.

Proposition 5.10. *Let S and G be conjunctive conditions. Let $Q(\bar{X}) :- R(\bar{X}), S$ be a query and let $C = Compl(R(\bar{X}), G)$ be a local completeness statement. Then*

$$Compl(Q) \models C \text{ if } S \subseteq G.$$

Proof Obvious. Whenever relation R is complete for all tuples which satisfy a condition S that is more general than G, it is also complete for all tuples that satisfy the specific condition G. $\square$

Chapter 6

Reasoning with Additional Information

In this chapter we present completeness reasoning in the presence of additional information about the database. In practice this can be important because often one is interested in the completeness of a specific state of the database.

The additional information we consider are schema constraints and extensional information. We show that both kinds of informations can allow the derivation of additional completeness statements, that is, statements that did hold before. In particular, we show the effect that foreign key constraints have in the presence of extensional information, which can lead to a significant reduction of the size of the completeness statements required for query completeness.

In section 6.1, we show how schema constraints such as keys and foreign keys can allow to conclude completeness statements would not hold without this additional information. In section 6.2 we introduce the new concept of finite domain constraints, and show how it also affects completeness reasoning. Section 6.3 shows the effect of extensional information, that is, knowledge of the available database part. In section 6.4, we show how schema and extensional information interfere and how they allow to replace necessity for completeness assertions by automated checks at runtime.

6.1 Schema Constraints

In this section, we describe the effect of schema constraints on completeness reasoning. The constraints we consider are keys of relations, foreign key dependencies between relations, and finite domain constraints.

We show how all three kinds of constraints can allow to derive additional completeness statements. These additional derivations are all based on the fact that

query completeness deduction can be reduced to query containment, and all three kinds of constraints can allow to derive additional containment.

Example 6.1. In this example we show how a finite domain constraints can allow to derive completeness for a query that does not hold without that constraint.

Consider the following query Q, its canonical completeness statement C_Q and two more local completeness statements C_1 and C_2.

$$Q(X) :- \mathit{person}(X, Y)$$
$$C_Q = \mathit{Compl}(\mathit{person}(X, Y), \top)$$
$$C_1 = \mathit{Compl}(\mathit{person}(X, Y), Y = \mathit{male})$$
$$C_2 = \mathit{Compl}(\mathit{person}(X, Y), Y = \mathit{female})$$

By theorem 5.6, satisfaction of C_Q is the characterizing condition for completeness of Q under multiset semantics. Suppose we want to decide whether C_Q is entailed by C_1 and C_2. According to theorem 5.1, this holds exactly if the query $\hat{C}_Q(X, Y) :- \mathit{person}(X, Y)$ is contained in $\hat{C}_1 \cup \hat{C}_2$, where with $\hat{C}_1$ and $\hat{C}_2$ we denote, as usual, the queries corresponding to C_1 and C_2 as usual.

In the general case, C_Q is not contained in $\hat{C}_1 \cup \hat{C}_2$. However, observe that with respect to the information that persons can only have gender male or female, the containment would hold. □

We will give more examples in the next subsections.

In our setting, schema constraints restrict the ideal database. That is, a schema constraint holds on a partial database instance, if it holds on the ideal version of it, whereas in general it may be violated in the available database. As the ideal database is normally considered to be unknown, schema constraints merely have the effect of restricting the set of possible ideal database instances for a given available database instances.

To illustrate this idea, consider the foreign key from *student.level* and *code* to *class.level* and *code*. It restricts the possible ideal database instances for available database instances in such a way, that whenever there is a student tuple in the available database, the ideal database must contain the student's *level* and *code* together as tuple in the *class* relation.

Observe that key constraints and finite domain constraints trivially also hold in an available database instance, if they hold in the ideal database instance.

Note also that deciding query completeness under all the schema constraints presented here using our solution also leads to a generally undecidable problem, because we reduce query completeness entailment to local completeness entailment, and this to query containment, which, with both keys and foreign keys, is undecidable [JK82].

We will discuss possible incomplete but practical approaches in the chapter on implementation.

6.1.1 Keys

Keys are one of the most basic concepts of relational databases. Formally, keys are subsets of the attributes of a relation, whose values must be unique in any relation instance. An equivalent definition is that these subsets functionally determine all other attribute values in that relation.

Key constraints are special cases of functional dependencies. We restrict our considerations to key-based functional dependencies (keys) because they are, in contrast to non key-based functional dependencies, much more often used in practice. Also, the treatment of non key-based functional dependencies does not give any new insights.

Knowledge of keys that hold in database instances can allow to derive more containment relationships. For any two literals in some expression which have the same symbols at the position of the key attributes, database valuations have to map the literals to the same ground atoms. Thus, the two expressions can be combined, which may lead to further containment.

Example 6.2. Suppose we are interested in deciding whether a given local completeness statement C_1 entails another statement C_2, where C_1 is the completeness statement for all students which are enrolled in class $3a$, and C_2 the completeness statement for all students which are enrolled in some class in level 3, and in some class with code a. Formally, the completeness statements and their corresponding queries are

$$
\begin{aligned}
C_1 &= \mathit{Compl}(\mathit{student}(N, L, C), L = 3, C = a)\\
C_2 &= \mathit{Compl}(\mathit{student}(N, L, C), \mathit{student}(N, 3, C_1), \mathit{student}(N, L_2, a))\\
\hat{C}_1(N, L, C) &:- \hat{\mathit{student}}(N, L, C), L = 3, C = a\\
\hat{C}_2(N, L, C) &:- \hat{\mathit{student}}(N, L, C), \hat{\mathit{student}}(N, 3, C_1), \hat{\mathit{student}}(N, L_2, a)
\end{aligned}
$$

According to theorem 5.1, C_2 is entailed by C_1 exactly if $\hat{C}_2$ is contained in $\hat{C}_1$.

In general, this containment does not hold, because we do not find a homomorphism from C_1 to C_2. However, with the knowledge that the *name* attribute of the student relation is its key, we can minimize C_2 in such a way that the three $\hat{\mathit{student}}$ atoms with the same variable as key are summarized in one atom so that C_2 becomes

$$\hat{C}_2(N, 3, a) :- \hat{\mathit{student}}(N, 3, a).$$

Then, a homomorphism from $\hat{C}_1$ to $\hat{C}_2$ exists, hence $\hat{C}_2$ is contained in $\hat{C}_1$, and hence C_1 implies C_2. □

6.1.2 Foreign Keys

Foreign keys are another commonly used constraint concept of relational databases. Existence of a foreign keys expresses that the values of a certain set of attribute of one relation always have to be a subset of the values of a certain set of attributes of another relation, for which they also are the key. That is, foreign keys allow to model strict $n:1$ relationships. They are a foundational concept for modelling data. Moreover, they are essential for normalizing database schemas.

Foreign key constraints are a special case of inclusion dependencies. We restrict our considerations to foreign keys, because non key-based inclusion dependencies are rarely used in practice, and their treatment would not lead to new insights.

Like keys, foreign keys affect containment reasoning, as knowledge of foreign keys allows to derive more containment. That is, with foreign keys one can add not explicitly mentioned atoms to queries, which must be there due to foreign key relationships. Then, more containment may be derivable.

Example 6.3. Consider a completeness statement C_1 for all students that are enrolled in some class, and a completeness statement C_2 for all students:

$$C_1 = \mathit{Compl}(\mathit{student}(N, L, C), \mathit{class}(L, C, P))$$
$$C_2 = \mathit{Compl}(\mathit{student}(N, L, C), \top).$$

In general, statement C_1 does not implies statement C_2, because students who are not in any class might be missing in when only C_1 is satisfied. However, observe, that by the fact that there is a foreign key asserted from students level and code attributes to the class table that holds in the ideal database, every student necessarily is enrolled in some class. So if the *student* table is complete for all students which are ideally enrolled in some class, it is complete for all students. That is, C_1 entails C_2 when considering the foreign key assertion.

Like in the previous example about keys, this is exactly reflected on the level of containment. In general, $\hat{C}_2$ is not contained in $\hat{C}_1$, however with respect to the foreign key, it is. □

While the impact of foreign keys on containment checking is well known, we will show a more interesting effect of foreign keys in connection with extensional information in subsection 6.4.

6.2 Finite Domain Constraints

Finite domain constraints are a constraint concept expressing that only a certain finite set of values may appear for some attribute in some part of a relation.

Although not included in the SQL standard, finite domain constraints are implemented in important relational database systems as PostgreSQL or MySQL.

For complete relations, finite domain constraints are used explicitly in practice. We present an even stronger concept by allowing finite domain constraints that apply only to parts of relations. We are not aware of any explicit use of this, however these concepts are important in explaining some effects in completeness reasoning with extensional data that is shown in the next section.

Example 6.4. As example for a finite domain constraint for a complete relation consider the finite domain constraint for the relation *person* that requires the *gender* attribute to take its values only in the set $\{\, male, female \,\}$.

With one local completeness statement for all male persons and one for all female persons, one then can derive completeness of the *person* table. □

Example 6.5. As example for a finite domain constraint over a part of a relation only, consider a finite domain constraint stating that values of the *code* attribute of tuples in the *class* table, where *class.level* is greater 8, should always be in the set $\{\, a, b, c \,\}$. This constraint would express that classes in a level higher than 8 may only have code a, b or c, whereas in lower levels still codes like e or g might occur.

With completeness statements for all classes with code a, b and c, one then can derive that a query asking for all classes in level 10 are complete, whereas classes in level 7 might not be. □

Formally, a *finite domain statement* is a four-tuple consisting of a relation R, a set A of attribute names of that relation, a conjunctive condition G and a set of tuples T over the domain, with the tuples having the same arity as the set of attribute names. We will denote such a finite domain constraint by the following expression:

$$Dom(R, A, G, T).$$

For a finite domain constraint F, we define the corresponding query Q_F as follows:

$$Q_F(A) :- R, G.$$

A finite domain statement *holds* in a partial database instance D, if $Q_F(\hat{D})$ is a subset of $\mathcal{S}$.

As shown in the example above, finite domain statements can affect containment reasoning. We are not aware of any discussion of containment with respect to finite domain constraints so far.

Given a set of finite domain statements, a query Q and a set of queries $\mathcal{Q}$, for checking whether $Q \subseteq \mathcal{Q}$ with respect to finite domain constraints, we investigate two possible approaches:

1. We add local completeness statements for all parts of relations that are outside the finite domains and then run standard containment algorithms.

2. We modify the containment algorithm in such a way that it takes into account the finite domains constraints.

The first technique has the advantage of not changing the containment algorithm and hence allowing to use standard algorithms. However, it requires to add polynomial many completeness statements with respect to the length of the query and the number of finite domain constraints. Also, the disequality operator is needed in the added local completeness statements, therefore this method is not applicable when the query language shall be that of simple conjunctive queries.

The second technique does not need to generate further completeness statements, however it may require exponentially many containment checks in the number of finite domain constraints.

Next, we show the two approaches more in detail.

Method 1. Adding Completeness Statements

The basic idea of this method is to transform finite domain statements into local completeness statements.

A finite domain constraint restricts the possible tuples for some specific parts of relations to finite sets, that is, no other tuples may be present in database instances satisfying the finite domain constraints. Over partial databases, that means that those other parts have to be empty both over the ideal and the available database. When such parts are empty over both database instances, then these parts are also complete, because no tuple can be missing from the empty set. Thus, there exist corresponding descriptions of finite domain constraints by local completeness constraints.

The method only works with respect to a concrete query, i.e., one cannot just add the local completeness constraints apriori, as their structure depends on the structure of the query.

Given a query and a finite domain constraint, one asserts that all the restricted canonical completeness conditions of the query hold, where the values of the attributes of the finite-domain constrained relation are restricted to be outside the finite domain. If statements about the values inside the finite domain are added, then these statements together yield satisfaction of the canonical completeness conditions.

Example 6.6. Consider our school database is that of an elementary school, that is, there is a finite domain constraint F on *class.level* which allows classes only to have levels from 1 to 4. Consider a local completeness statement C for all classes

in levels 1 to 4 and consider a query Q asking for all classes. Obviously, we should find out that Q is complete with respect to F and C. With C_Q being the canonical completeness condition of Q, $\hat{C}_Q \subseteq \hat{C}$ does not hold immediately. Now observe, that we can transform the finite domain constraint F into a local completeness statement

$$F_Q = \mathit{Compl}(\mathit{student}(N, L, C), L \neq 1, L \neq 2, L \neq 3, L \neq 4).$$

Now, $\hat{C}_Q \subseteq \hat{C} \cup \hat{F}_Q$ holds, hence C and F imply C_Q and hence completeness of the query Q. □

Method 2. Modifying the Containment Algorithm

The idea of this method is to modify the containment algorithm such that it checks containment for all possible instantiations with respect to given finite domains.

The idea is loosely related to that of linearization for containment checking with comparisons. However, generating all possible linear orderings of a query, we generate all possible instantiations of it with respect to finite domains. If containment holds for all possible instantiations with respect to finite domains, then, containment holds with respect to the finite domains.

For a query $Q(\bar{X}) :- L_1, \ldots, L_n$ and a finite domain constraint $F = \mathit{Dom}(R, \mathit{attributes}, G, S)$, we define the instantiations of Q with respect to F to be all the queries where *attributes* of relations R that satisfy G have been substituted by elements of S. A query Q is then contained in a set of queries $\mathcal{Q}$ with respect to a finite domain constraint F, if all instantitations of Q with respect to F are contained in $\mathcal{Q}$.

An efficient algorithm for finite domain containment checking using this method we show in section 8.2. The key of this algorithm is to not instantiate every variable with a finite domain, but only those for which containment of the instantiations is possible.

Example 6.7. Following our previous example, the query for all classes Q would be replaced by the four instantiations of it where the variable in the *class* literal is replaced by the constants 1 to 4. As for each of those instantiations, the containment with respect to the query corresponding to the given completeness statement holds, we could then conclude that the given completeness statement and finite domain statement entail query completeness. □

Next we discuss the complexity of finite domain containment. Notably, the problem is strictly harder than containment without finite domains.

First, let us discuss an upper bound for the complexity. The most naive way of checking whether a query Q is contained in queries Q_1 to Q_n under a set of finite

domain constraints $\mathcal{F}$ is to check for each possible instantiation of the variables in Q to which finite domain constraints in $\mathcal{F}$ apply, whether the instantiated version of Q is contained in Q_1 to Q_n.

Unluckily, it is not trivial to find all possible instantiations of Q with respect to $\mathcal{F}$, because guarded finite domain constraints can interact. E.g., when one or several finite domain constraints together restrict a variable to only be allowed to take one value, then this may trigger the condition of another finite domain constraint to become satisfied allowing another finite domain constraint to be applied. Still the possible interactions always terminate, because every finite domain constraint can be applied to every position in a query at most once. But an algorithm taking into account these interactions seems not trivial.

A Naive Algorithm

We present an easier way of how to get all valid instantiations of Q with respect to $\mathcal{F}$: We generate the maximum set of instantiations possible with respect to $\mathcal{F}$, then for each instantiation we check if it is valid, i.e., whether it does not violate any finite domain constraint in $\mathcal{F}$.

To get all possible instantiations, we first generate a set T of constants as the union of all constants appearing in finite domain constraints in $\mathcal{F}$ together with as many new constants not appearing in $\mathcal{F}$ nor in the queries Q and Q_1 to Q_n as there are variables in Q. Then we generate all possible instantiations of Q with respect to T. For variables to which finite domain constraints are applicable, we for sure have all possible values captured since their finite domain then is a subset of T. For variables to which no finite domain constraint is applicable, we cover sufficiently many cases by introducing as many new constant symbols as there are variables in the query.

For each instantiation we then check whether it is valid with respect to $\mathcal{F}$, i.e., whether it does not violate any finite domain constraint in $\mathcal{F}$. This is done by checking for every finite domain constraint at every position of the instantiated query, whether it is applicable, and if so, if the instantiated value is member of the finite domain that is applicable. Testing applicability of finite domain constraints just requires homomorphism testing.

The number of possible instantiations is exponential in the number of finite domain constraints and in the size of the query. Assumed having an NP-oracle, testing whether for all valid instantiations the containment holds, is a problem in Co-NP. To find out that finite domain containment does not hold, one just needs to guess one instantiation for which the containment does not hold. However, to show that containment holds, one needs to consider all possibly exponentially many instantiations. For a fixed instantiation, the remaining procedure is a standard containment check (e.g., by homomorphism testing), which is an NP problem.

Combining the two steps, we find out that the naive algorithm can decide the problem in Π_2^P.

The process of testing whether an instantiation is valid with respect to a set of finite domain constraints $\mathcal{F}$ has no significance for the complexity of the algorithm, since it just requires homomorphism testing which is an NP problem.

Note that this algorithm can also be extended to conjunctive queries containing comparison, however then

Lemma 6.8. *Containment of simple conjunctive queries in the presence of finite domain constraints can be decided by naive instantiation in* Π_2^P.

To find the lower bound for the problem of finite domain containment of simple conjunctive queries, we provide a reduction from a Π_2^P-complete problem to the problem of finite domain containment of simple conjunctive queries.

It is the problem of deciding validity of universally quantified 3-satisfiability (∀3-SAT). The ∀3-SAT formulas are of the following form

$$\forall X_1, \ldots, X_m \exists Y_1, \ldots, Y_n : C_1 \wedge \ldots \wedge C_k,$$

where clause C_i is of the form $L_{i1} \vee L_{i2} \vee L_{i3}$ with L_{i1} to L_{i3} being literals using propositions from $X_1, \ldots, X_m$ and $Y_1, \ldots, Y_n$.

Let ϕ be a formula of the above form. We create an instance of a finite domain containment problem such that containment holds exactly if ϕ is valid.

The database schema we use is $\Sigma = \{ R_1/2, \ldots, R_m/2, S/2, C'_1/3, \ldots C'_k/3 \}$.

For a clause C_i with literals L_{i1}, L_{i2} and L_{i3} in ϕ, we define conjunctive conditions

$$G_i = R_i(a, W_i), R_i(W_i, b), S(W_i, 0), S(b, 1)$$
$$G'_i = R_i(a, b), S(b, X_i).$$

Furthermore, the set $C_i'^{(7)}$ denotes the set of the 7 ground instances of predicate C'_i over the domain $\{ 0, 1 \}$, such that constant 1 at position j in C'_i corresponds to the variable Z_{ij} being mapped to true, and the 7 ground instances are the ones where C_i evaluates to true under the variable mapping.

Let F be the finite domain constraint with

$$F = Dom(S, 1, \top, \{a, b\})$$

Let Q_1 and Q_2 be the following queries:

$$Q_1() :- G_1, \ldots, G_m, C_1'^{(7)}, \ldots, C_k'^{(7)}$$
$$Q_2() :- G'_1, \ldots, G'_m, C'_1, \ldots, C'_k$$

Lemma 6.9. *Let ϕ be a $\forall$3-SAT formula as shown above and let Q_1, Q_2 and F be constructed as above. Then*

$$\phi \text{ is valid exactly if } Q_1 \subseteq_F Q_2.$$

Proof For containment to hold, both each conjunctive condition C_i' has to be contained in the conjunctive condition $C_i^{(7)}$, and the conjunctive condition G_j' in the conjunctive conditions G_j.

The key of that containment is that the variables W_i in G_i are not restricted further than to have the constant values a or b. The value of W_i determines which value X_i has, thus, X_i can be both 0 or 1. Thus, the indeterminacy of W_i leads to containment having to hold both in the case of X_i being 0 or 1, representing the universal quantification of the X variables in ϕ.

The remaining part of the containment problem is standard, for every possible assignment of the X variables, there must exist a valuation of the Y variables such that each C' clause becomes a ground instance for which the clause evaluates to true.

The reduction is correct, because whenever containment holds, an assignment for the Y variables for each combination of X variables has to exist that enables the containment of the C' in the $C^{(7)}$, it is complete because whenever ϕ is valid, for every combination of the X variables, some combination of the Y variables has to exist that makes all the clauses true. □

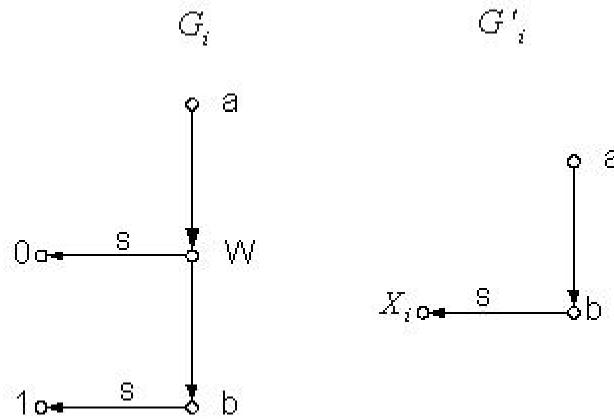

Figure 6.1: Structure of G_i and G_i'. Depending on the value assigned to W, X_i becomes either 0 or 1.

Having the reduction stated in Lemma 6.9 from universally quantified 3-satisfiability and knowing that the problem of universally quantified 3-satisfiability is Π_2^P-complete [St76], we now can formulate the following lemma:

Lemma 6.10. *Containment of simple conjunctive queries in the presence of finite domain constraints is* Π_2^P*-hard.*

Putting the lemmas above together, we can state:

Theorem 6.11. *Containment of simple conjunctive queries in the presence of finite domain constraints is* Π_2^P*-complete.*

Proof Follows from lemmas 6.8 and 6.10. □

6.3 Extensional Information

Completeness reasoning with extensional information means completeness reasoning with knowledge of the available database. We consider this to be an important practical case.

In this case of knowledge of the content of the available database, given local completeness statements do not only express equivalence between parts of the available and the ideal database. They also allow to explicitly know the content of parts of the ideal database.

Whenever the content of a part of the ideal database is known, this directly yields finite domain constraints for attributes in that part. I.e., there may not be other tuples than the ones present in the available database.

While that fact by itself is trivial, it becomes interesting when join operations are performed. As shown in the previous section, finite domain constraints are propagated between relations when joins are performed. A finite domain on an attribute of one relation also applies to an attribute of another relation, when the two attributes are the join attributes. Then, local completeness assertions may yield completeness which did not hold before.

Example 6.12. Consider a database schema $\Sigma = \{ R/1, P/1 \}$ and a query $Q(X) :- R(X), P(X)$. Consider an available database instance where the extension of $\check{R}$ is empty and consider R is known to be complete by a completeness statement C. Although no completeness is known for P, one can conclude that Q is complete with respect to $\check{D}$ and C because the result of Q will always be empty over any $\hat{D}$ where $R(\hat{D})$ is empty. □

Example 6.13. As a more complex example, consider our standard schema without knowledge of the foreign keys. Consider a database instance of it, where in the *class* table we only find classes with levels from 1 to 8. Suppose, completeness for the *class* table was stated.

Now, consider a query $Q(N) :- \mathit{student}(N, L, C1), \mathit{class}(L, C2, P)$ asking for all student names for which exists a class in their level (i.e., a join is performed between *student.level* and *class.level*).

Since there is a finite domain constraint on *class.level*, it also applies to *student.level* by the join. Thus, given a local completeness statement for students in levels 1 to 8, one could conclude query completeness.

Without the extensional knowledge of the *class* table this would not be possible, since there could be students in the result, which are in a level higher than 8. □

Let us characterize the impact of extensional information in a formal way. Let Q be a query, $\check{D}$ be an available database instance, and $\mathcal{C}$ be a set of local completeness assertions. We say that $\check{D}$ and $\mathcal{C}$ entail $\mathit{Compl}(Q)$ if no valid extension $\hat{D}$ for $\check{D}$ exists such that the answer of Q is different over $\check{D}$ and $\hat{D}$. Formally:

$$\check{D}, \mathcal{C} \models \mathit{Compl}(Q) \text{ if and only if for all } \hat{D} \text{ with } (\hat{D}, \check{D}) \models \mathcal{C} \text{ we have } Q(\hat{D}) = Q(\check{D}).$$

This definition characterizes what query completeness with respect to an available database instance and a set of local completeness assertions means. That is, for any ideal database valid with respect to the set of local completeness assertions, the query should be complete.

A Naive Decision Procedure

The proposition also gives rise to a decision procedure immediately. There can be infinitely many valid ideal databases for a given available and local completeness statements. However, it suffices to inspect finitely many in order to find out, whether there exists one valid ideal database instance where $Q(\hat{D}) \neq Q(\check{D})$.

An ideal database instance $\hat{D}$ can only violate the query completeness if the result of Q over $\hat{D}$ contains a tuple which is not in the output of Q over $\check{D}$. With respect to a fixed $\check{D}$, there are only finitely many new different tuples to consider. And for every new tuple, there has to exist a valuation of Q over $\hat{D}$ yielding that tuple. Again, it suffices to consider finitely many valuations.

Only finitely many new tuples need to be considered, because we only need to consider result tuples over the active domain joined with a set of new constants of the arity of the distinguished variables of the queries. A new tuple cannot be more different than different in every constant from the active domain.

And for each new tuple, only finitely many valuations need to be considered, because we only need to evaluate Q over the active domain joined with a set of new constants with the size of the number of variables in the body of Q. More new constants than the queries has variables do not result in anything new.

Example 6.14. Consider a database schema $\Sigma = \{ R/2, P/1 \}$, a query Q, a partial database instance $\check{D}$ and a local completeness statement C where

$$
\begin{aligned}
&Q(X) :\!- R(X,Y), P(Y)\\
&R(\check{D}) = \{ (a,b), (a,c) \}\\
&P(\check{D}) = \{ b \}\\
&C = \mathit{Compl}(R(X,Y), \top).
\end{aligned}
$$

Here, $C \models \mathit{Compl}(Q)$ does not hold because no completeness for relation P is asserted. However, $C, \check{D} \models \mathit{Compl}(Q)$ does hold, because whatever tuples might be missing in $\check{P}(D)$, they can not lead to any additional tuple in the output of Q, because the extension of R in $\check{D}$ allows only constant a to be in the output, and relation R is asserted to be complete.

With the decision procedure sketched above, one has to test for all tuples over the active domain joined with a finite set of new constants, whether they can be in the output of a $\hat{D}$ that does not violate C. So here, one would consider constants b, c and w.

We only consider constant b here, the other cases are analogous. For constant b to be in the result of Q over a $\hat{D}$, there has to exist a valuation over $\hat{D}$ where X is mapped to b. For mapping Y, again finitely many possibilities of mapping it would have to be considered. But no matter what Y is mapped to, following the first literal in Q, $\hat{R}(D)$ would have to contain a tuple where the first component is b. This would violate the completeness statement C, and hence it is not possible. Considering the other cases analogously, we find that C and $\check{D}$ imply completeness of Q. □

Note that the decision procedure as sketched above directly only applies to completeness reasoning under set semantics. To use it also for completeness reasoning under multiset semantics, a slight change has to be made: Instead of considering all possible new tuples in the result of Q over $\hat{D}$, one also has to consider old ones, that are tuples which are already in $Q(\check{D})$, because they may be yielded again by other valuations.

In the example above, C and $\check{D}$ do not imply completeness of Q under multiset semantics, because if $P(\hat{D})$ contained c, then Q would yield tuple a two times over $\hat{D}$. If $P(\check{D})$ also contained c, then Q would be complete with respect to $\check{D}$ and C under multiset semantics.

The Complexity of the Naive Decision Procedure

Regarding the complexity of the algorithm, observe that in order to return the answer *Yes* (that means, a query Q is complete with respect to given $\check{D}$ and $\mathcal{C}$), for every valuation that leads to a new tuple being in the result of Q with respect to

$\check{D}$, one needs to find a violation of a completeness assertion. In the size of Q, there are exponentially many valuations leading to new tuples, and for every of them, the queries corresponding to the local completeness statements have to be evaluated to see whether none of the statements is violated. For conjunctive queries, query evaluation is an NP-complete problem. So the overall decision procedure is in Π_2^P, because if an NP oracle existed for solving the query evaluation problem, the overall problem with interchanged answers would be in NP: For finding that $\check{D}$ and $\mathcal{C}$ do not entail completeness of Q, one just needs to guess one valuation of Q that yields a new tuple not in $Q(\check{D})$, and then evaluate the queries from the local completeness statements to show that the valuation does not violate any local completeness statement.

Lemma 6.15. *The problem of query completeness under local completeness and extensional information can be decided in Π_2^P by the algorithm sketched above.*

This lemma gives us an upper bound for the complexity of the decision problem.

To find a lower bound, we again provide a reduction of the problem of validity of universally quantified 3-SAT formulas.

Consider ϕ to be a again an universally quantified 3-SAT formula of the form

$$\forall X_1, \ldots, X_m \exists Y_1, \ldots, Y_n : C_1 \wedge \ldots \wedge C_k,$$

as used in the proof of lemma 6.10.

We define a query completeness problem $\Gamma_\phi = (\mathcal{C}, \check{D} \models \mathit{Compl}(Q))$ as follows:

Let the relation schema Σ be $\{\, B/1, R_1/1, \ldots, R_m/1, C_1'/3, \ldots, C_k'/3 \,\}$. Let Q be a query such that

$$Q() \coloneq B(X_1), R_1(X_1), \ldots, B(X_m), R_m(X_m).$$

Let $\check{D}$ be such that $B(\check{D}) = \{\, 0, 1 \,\}$, and for all i from 1 to m let $R_i(\check{D}) = \{\}$ and $C_i'(\check{D})$ contains all the 7 triples over $\{\, 0, 1 \,\}$ such that C_i becomes true when the variables in C_i become the truth values assigned that correspond to 0 and 1.

Let $\mathcal{C}$ be a local completeness statement such that

$$\mathcal{C} = \mathit{Compl}(R_1(X_1), \ldots, R_m(X_m), C_1(\bar{Z}_1), \ldots, C_k(\bar{Z}_k)),$$

and the $\bar{Z}_i$ are variables from X_1 to X_m union Y_1 to Y_n as in ϕ.

Lemma 6.16. *Let ϕ be a $\forall$3-SAT formula as shown above and let Q, $\mathcal{C}$ and $\check{D}$ be constructed as above. Then*

$$\phi \text{ is valid if and only if } \mathcal{C}, \check{D} \models \mathit{Compl}(Q).$$

Proof Observe first, that validity of ϕ implies that for every possible combination of the X variables, there exist Y variables such that C_1 to C_k in $\mathcal{C}$ evaluate to true.

Completeness of Q follows from $\mathcal{C}$ and $\check{D}$, if Q returns the same result over $\check{D}$ and any ideal database instance $\hat{D}$ that subsumes $\check{D}$ and $\mathcal{C}$ holds over $(\hat{D}, \check{D})$.

Q returns nothing over $\check{D}$. To make Q return the empty tuple over $\hat{D}$, one value from $\{0, 1\}$ has to be inserted into each ideal relation instance $\hat{R}_i$, because every predicate R_i appears in Q, and every extension is empty in $\check{D}$. This step of adding any value from $\{0, 1\}$ to the extensions of the R-predicates in $\hat{D}$ corresponds to the universal quantification of the variables X.

Now observe, that for the query to be complete, none of these combinations of addings may be allowed. That is, every such adding has to violate the local completeness constraint $\mathcal{C}$. As the extension of R_1 is empty in $\check{D}$ as well, $\mathcal{C}$ becomes violated whenever adding the values for the R-predicates leads to the existence of a satisfying valuation of the body of $\mathcal{C}$. For the existence of a satisfying valuation, the mapping of the variables Y is not restricted, which corresponds to the existential quantification of the Y-variables.

The reduction is correct, because whenever $\mathcal{C}, \check{D} \models \mathit{Compl}(Q)$ holds, for all possible addings of $\{0, 1\}$ values to the extensions of the R-predicates in $\hat{D}$ (all combinations of X), there existed a valuation of the Y-variables which yielded a mapping from the C-atoms in $\mathcal{C}$ to the ground atoms of C in $\check{D}$, that satisfied the existential quantified formula in ϕ.

It is complete, because whenever ϕ is valid, then for all valuations of the X-variables, there exists an valuation for the Y-variables that satisfies the formula ϕ, and hence for all such extensions of the R-predicates in $\hat{D}$, the same valuation satisfied the body of C_0, thus disallowing the extension. □

Theorem 6.17. *Deciding query completeness in the presence of extensional information is Π_2^P-complete.*

Proof The hardness follows from lemma 6.16 and the fact that deciding validity of universally quantified 3-SAT formulas is Π_2^P-complete. The completeness follows from lemma 6.15, which shows that there exists an algorithm that decides query completeness with extensional information in Π_2^P. □

Interesting about this complexity result is that it shows that completeness reasoning with extensional for simple conjunctive queries is strictly harder than without. For simple conjunctive queries, we showed in 5.2 that the problem of deciding query completeness under given local completeness is in NP.

Note that alternatively one can also reduce the problem of containment with unrestricted finite domain constraints to completeness reasoning with extensional

information. Given two queries Q_1 and Q_2 for which $Q_1 \subseteq_F Q_2$ shall be decided, this can be reduced to deciding whether a $\check{D}$ and an $\mathcal{C}$ models $Compl(Q_1)$, where $\check{D}$ contains the frozen body of Q_2 and $\mathcal{C} = \mathcal{C}_{Q_2}$, and additionally the finite domains are encoded by relation instances in $\hat{D}$ for which completeness is asserted.

That reduction also works vice versa, that is, every completeness reasoning problem with respect to extensional information can naturally be translated in a reasoning problem with finite domain constraints, by translating every data in the extensional information that is asserted to be complete, in a finite domain constraint.

6.4 Schema and Extensional Information

In this section we show how in presence of both schema and extensional information, sufficient conditions for query completeness arise which can strongly reduce the completeness assertions needed to be pregiven for query completeness. In particular, we show how satisfaction of certain canonical completeness conditions can be checked automatically.

Alon Levy was first to observe this effect [Le96]. He showed how in presence of extensional data and functional dependencies, two queries can posed to the extensional data such that containment of their results allowed concluding query completeness.

The method we present is based on Levy's observation, however it allows conclusions also about satisfaction of the canonical completeness conditions instead of the whole query only, and furthermore, in the case of foreign key assertions, it can allow to detect database incompleteness.

The idea is that in presence of extensional information, whenever a query contains a join between two literals L_1 and L_2, where on for L_2 the key attributes are involved in the join, then, whenever L_1 is known to be complete, and in the extension of the relation of L_2 we find for each tuple in the extension of the literal L_1 one tuple joining with it, then the join result is known to be complete. That holds, because there can be only at most one tuple per key value in the extension for L_2. So if we find one tuple per tuple in L_1, then we have found the maximum of what can be there.

The effect is, that some canonical completeness information required to be asserted in order to be able to conclude query completeness, does not need to be asserted any more, but can be checked automatically.

Example 6.18. Consider a query Q asking for all names of students together with their gender. Note that there is a foreign key from *student.name* to *person.name.*

$$Q(N, G) :- \mathit{student}(N, L, C), \mathit{person}(N, G).$$

To conclude completeness of this query, we originally need completeness statements asserted for all students that are persons, and all persons that are students.

Now suppose, we only know that all students which are persons are complete, and suppose the available *student* table contains only $(andrea, 2, b)$ and $(beatrix, 5, d)$.

Since we have the foreign key from *student.name* to *person.name*, it suffices to check whether there are tuples in the available *person* table with *name* equals *andrea* and *beatrix*. Because the foreign key holds in the ideal database, there are such tuples in the ideal relation, and since *name* is the key of the relation, exactly one per name. Thus, if we find this two tuples in the available *person* relation, we can conclude the canonical completeness for the *person* relation and from this, query completeness follows. □

Query Graph

Before presenting our main results formally, we have to introduce some terminology.

Consider a conjunctive query

$$Q(\bar{X}) :- L_1(\bar{Y}_1), \ldots, L_n(\bar{Y}_n), C_{n+1}(\bar{Y}_{n+1}), \ldots, C_m(\bar{Y}_m)$$

where L_1 to L_n are literals and C_{n+1} to C_n are comparisons. Suppose Q contains no equalities, that is, all equalities have been eliminated.

We say that there exists a *join* between literals L_i and L_j on variables $\bar{K}$ whenever $\bar{K}$ is both a subset of $\bar{Y}_i$ and $\bar{Y}_j$. A join is a *key join* from L_i to L_j if the variables $\bar{K}$ are the key variables of the relation symbol of literal L_j. A key join is a *foreign key join*, if a foreign key was asserted for it.

Given a conjunctive query Q, it's *query graph* G_Q is a graph with both unlabelled undirected and labelled directed edges such that:

- for every literal in Q, there is a node in G_Q,
- there is an undirected edge between nodes L_i and L_j, if there is a join between L_i and L_j in Q that is not a key join and
- there is a directed edge with label *key* from node L_i to L_j if there is a key join from L_i to L_j in Q that is not a foreign key join.
- there is a directed edge with label *foreign key* from node L_i to L_j if there is a foreign key join from L_i to L_j in Q.

A node in a query graph is called a *sink*, if it has no undirected edges and no outgoing directed edges.

A node is called a *transitive sink*, if it is a sink or if it has no undirected edges and all outgoing directed edges are directed to nodes which are transitive sinks.

Every node that is not a transitive sink is called a *core* node.

The *core query graph* G_{core} of a query graph G_Q is the subgraph of G_Q, where all transitive sinks have been eliminated. The *core query* Q_{core} is the query corresponding to G_{core}.

The Check$_F$ Algorithm

The *check$_F$* is a boolean function, with the algorithm shown in table 6.1. It takes a query Q, a literal L inside Q and an available database instance $\check{D}$ as input, and returns true exactly if all partial databases instances having $\check{D}$ as available instance and having the canonical completeness for the literals in the core query asserted, satisfy canonical completeness for the input literal with respect to some schema assertions F.

```
CheckF (Li, Q, Ď)
   if not exists a directed path in GQ from Gcore to Li
        return FALSE
   for every t ∈ Qcore(Ď)
      for every maximal directed path L1, ..., Li in GQ
         if not exists l1, ..., li in Ď such that
             t and l1 agree on the join attributes between Qcore
                 and L1
             AND
             every lj and lj+1 agree on the join attributes
                 between Lj and Lj+1
         return FALSE
   return TRUE
```

Table 6.1: Function Check

The following theorem expresses that in presence of schema and extensional information, holding of canonical completeness assertions for literals that are transitive sinks can be checked on the extensional data, and the check is a sufficient condition.

Lemma 6.19. *Let Q be a query, L be a literal in Q that is a transitive sink in the query graph of Q, F be a set of schema assertions, $\check{D}$ be an available database instance and C_{core} be the set of canonical completeness conditions for the literals in the core of Q. Then*

$$Check_F(L, Q, \check{D}) \wedge Compl(C_{core}) \models Compl(L)$$

Proof When the *check* algorithm succeeds, then for every tuple in the core query over $\check{D}$, a tuple in the extension of $\check{R}_i$ in $\check{D}$ of literal L_i exists, that is related via a chain of foreign key joins.

The fundamental reason for that being sufficient is that the joins considered are joins with key attributes, so there can be at most one such tuple per tuple in $Q_{core}(\check{D})$. Therefore, finding just one tuple for every tuple in $Q_{core}(\check{D})$ is a sufficient condition for satisfaction of the canonical completeness of literal L. □

If a literal is connected to the core query by foreign key joins only, the sufficient conditions stated above become necessary conditions. That is, because a foreign key assertion not only states that there can be at most one tuple per join attributes, but it also states that there has to be at least one.

Let L, Q, F and $\check{D}$ be as above.

Proposition 6.20. *If L is connected to the core of Q only by edges that are labelled as foreign key joins, then*

$$Compl(L) \models Check(Q, L, \check{D}, F).$$

Different from any theory presented so far, this proposition allows to explicitely detect database and query incompleteness. All methods presented so far could detect whether a query is necessarily complete under given completeness, schema and extensional information. A query not found to be complete in general, could still be complete on a concrete instance. With this proposition now, one can explicitely find out that some local completeness does not hold. Under multiset semantics, this directly implies query incompleteness, while under set semantics it still depends on the projections.

Lemma 6.19 has an interesting generalization. For queries containing key joins, only for the literals in the core query, canonical completeness needs to be asserted. For all other canonical completeness required for query completeness, satisfaction can be tested by the *check* algorithm.

Theorem 6.21. *Let Q be a query, $\mathcal{C}$ be a set of local completeness assertions, $\check{D}$ be an available database instance and F be a set of schema assertions.*
Then Q is complete over any partial database instance satisfying $\mathcal{C}$ and F and having $\check{D}$ as available database instance if the following holds:

1. *$\mathcal{C}$ implies canonical completeness for all literals in the core of Q,*
2. *the check succeeds for all literals that are not in the core of Q.*

Proof Follows from lemma 6.19.

Example 6.22. Consider the query Q asking for all persons joined with the language courses they attend.

$$Q(N, L) :- \mathit{person}(N, G), \mathit{student}(N, L, C), \mathit{language_attendance}(N, G), \mathit{language}(G).$$

Since *language_attendance* is the only core literal, when deciding completeness over a concrete database instance, only canonical completeness for *language_attendance* needs to be asserted, while canonical completeness for relations *student*, *language_attendance* and *language*, the satisfaction of the canonical completeness conditions can be checked.

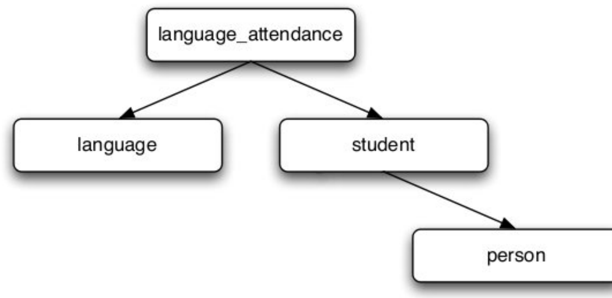

Figure 6.2: Query graph for Q. The only core node is *language_attendance*

□

Checks for literals that are not transitive sinks in the query graph can be done as well. However, succeeding of these checks is a too strong condition for canonical completeness even in the case of foreign key joins. If a literal is not a transitive sink, there exists a directed path from it to a literal with an undirected edge. In other words, it joins directly or indirectly with another literal via a normal join, and this join is a further constraint on the tuples that have to be in an relations extension in order to satisfy canonical completeness.

Example 6.23. Consider a query

$$Q(x) :- R(X, Y), S(Y, Z), T(W, Z).$$

Consider a set of schema constraints F such that the join between R and S is a foreign key join, the one between S and T not. Consider a partial database instance $\check{D}$ where

$$\check{R} = \{ (a, b), (a, c) \},$$
$$\check{S} = \{ (c, d) \},$$
$$\check{T} = \{ (e, d), (f, g) \}.$$

Then, the check algorithm for literal $L_2 = S(Y, Z)$ would return false, because no tuple starting with b is in $\check{S}$. Since the foreign key holds on the ideal database, there has to be some tuple starting with b in $\hat{S}$. However, as it is not known whether it ends with g or not, checking for presence of it is not a characterizing condition for local completeness. If the missing tuple in $\check{S}$ was (b, g), it would be necessary for canonical completeness of S, if it was (b, h) not. Thus, the check can not provide a characterization of local completeness for this non transitive sink literals. □

6.4.1 Exact-cardinality constraints

Exact cardinality assertions are an extension of the concept of foreign key assertions with a similar impact. They can allow to check canonical completeness automatically.

An exact cardinality assertion is a schema constraint expressing that for every tuple in some relation, an exact number of tuples has to be in another relation, such that all tuples agree on a certain set of attributes.

Given exact cardinality assertions, similar checks as for relations joined by foreign keys can be performed. The only difference is that instead of checking existence of tuples, one checks the exact cardinality of the existing tuples satisfying the common attribute condition.

Example 6.24. Consider an exact cardinality assertion that every student has to take exactly two language courses.

Then, for a query asking for all students joined with the language courses they take, it suffices to have completeness of the *student* table asserted, while canonical completeness of the *language_attendance* table can be tested automatically, by checking whether for every student there are two entries in the *language_attendance* table. □

The insight about exact cardinality assertions with extensional information is the same as for foreign keys in that case, namely that local completeness does not always need to be asserted but can be tested.

Chapter 7

Strategies for Stating Completeness

In this chapter we discuss strategies for data management aiming at ensuring query completeness. We show what completeness information really is, where it has to come from and how completeness statements can be split into smaller statements. Finally, we present a specific method for stating conditional completeness information that has the nice property of requiring less knowledge. Parts of the material presented in this chapter have already been discussed by Motro and Miliaev, we will stated that wherever it holds.

In chapter 4 we presented characterizations of query completeness in terms of local completeness statements. Chapter 5 presented general completeness reasoning, and chapter 6 completeness reasoning with respect to schema or instance information. The essence of these chapters was that completeness statements are required for concluding query completeness.

Except of the first of these chapters, the presentation of the results so far was mostly top down in the sense that a query and completeness information were assumed to be given, and then it conclusions were made whether the query could be derived to be complete or not.

In practice however, it is not always of relevance to have the completeness statements given. Instead, for a given query one may be interested in what completeness has to be asserted in order to make the conclusion of query completeness possible. After having found an appropriate set of assertions that would ensure a query to be complete, one then could inspect the data in a second step and try to ensure satisfaction of these statements, or where it does not hold, work on completing the data. So in practice it is not only relevant to decide query completeness, but also to find out what options exist to ensure a query to become complete.

In the first section of this chapter, we revise the facts about required completeness statements for query completeness, and discuss where completeness informa-

tion has to come from. In the second section, we present how local completeness statements can be partitioned in more smaller, equivalent statements. In the third section we present the concept of cardinality assertions for stating conditional completeness.

After reading this chapter, the reader should have an understanding of the implications of the theory presented so far on practical data management.

7.1 Where Completeness Information Comes from

As stated before, the holding of the canonical completeness statements is generally the characterizing condition for deducing completeness of a query. In presence of extensional information, some satisfaction of some canonical conditions can be checked automatically. However also in this case, there always exist at least one statement, for which completeness can not be checked automatically.

So some completeness has to be stated by someone. We call subjects that give completeness statements *competent agents*. Competent, because they need to have some knowledge about the ideal database, agents because the process of giving completeness statements is an active process. Whether the agent is a human or a program does not matter generally.

Usually, we believe the ideal database to be a representation of some "real world", that is not formalized [Mo89]. Thus, given completeness statements can in general not be formally verified. Some justification may be possible, but in general, the competence of the agent in giving completeness information has to be believed.

Without a formalization of the ideal database, for stating completeness the competent agent needs either to have knowledge about how the data was collected (i.e., in a complete way), or it needs to have some representation of what data should be there, and then can compare it with what is there in an available database instance. Both methods require background information, either about completeness of some method of data collection, or about the data in the ideal database ("real world"). Which method is more applicable heavily depends on the scenario, we only have the rough idea that the former methods might be more useful for asserting completeness in larger and/or stricter environments (w. r. t. the amount of data and the data management policy), while the latter might be more applicable in not so strict and smaller environments.

Example 7.1. To illustrate the two methods by examples, for the first method consider a public institution giving a call for a project with some deadline for application. If the method of storing the received applications to the database can be asserted to be complete, then by the deadline one can consider the set of all

valid applications as complete in the database. There may be applications missing, but since the deadline is strict, none of those is valid, and therefore does not miss in the set of valid ones. Manual inspection of the database table would not make much sense if there are sufficiently many applications: No one could judge by manual data inspection, whether all data that should be there is there. □

Example 7.2. In contrast, consider our school database example. Completeness of data collection can hardly be assumed here, and strictness is not a common policy. It is not applicable to conclude that by the deadline of enrolment all enrolment forms that the parents have to sign are really there. Students whose forms are missing by the deadline will hardly be rejected from school just because of that.

To judge data completeness, the second method of manual inspection seems much more applicable. E.g., the secretariat could hand out a print of the list of enrolled students for his/her class to each class teacher. The class teachers then inspect the lists and can compare if some students present in their class are missing on the lists. If this is not the case, the class teachers can assert completeness for the students of his class in the database.

How the single completeness statements for each class can imply completeness of all students is shown in the next section. □

To conclude this section, we note that nonexistence of a formalization of the ideal database seems the more common case, but does not always have to be.

Instead, e.g., in the field of security sensitive databases, data replication or (partially) mirrored databases for performance reasons, it can be that the ideal database exists as formal database. In such cases, completeness information could be determined fully automated.

7.2 Partitioning Local Completeness Statements

Giving completeness statements can be a difficult task. Especially for completeness assertions that shall be given by manual data inspection, a relatively small number of tuples in an available database can make manual assertion infeasible.

Applying the theory presented in chapter 6, we show two methods that can alleviate the process of asserting completeness by dividing local completeness statements in smaller, equivalent statements. Both methods were first presented by Miliaev in [Mi10].

Partitioning by Table

The first method which we (following Miliaev's nomenclatura) call *partitioning by attribute*, represents the idea to split local completeness statements by finite

domain assertions.

Whenever a local completeness statement over a relation with an attribute to which a finite domain constraint applies shall be made, the statement can be replaced by statements for all the values of the finite domain. Applicability of this method heavily depends on the context and on the size of the finite domain, however, well used finite domain constraints can strongly alleviate the process of asserting completeness. Statements about relations for which a foreign key is asserted to a second relation, can also be split along finite domains of the second relation.

Example 7.3. Consider the completeness statement for all persons, which can be split into one statement for all male persons and one for all female ones. As for the *student* relation there is a foreign key asserted to the *person* relation, also a completeness statement for the *student* table can be split along the *gender* attribute. □

Partitioning by Attribute

The second method, also introduced by Miliaev, is the *partitioning by table*. The difference here is that instead of asserted finite domains, finite domains that result from completeness assertions are used for the partitioning. If for a relation, a foreign key to another relation exists which is asserted to be complete, a completeness statement for the first relation can be split along any attribute of the second. If completeness for a part of a relation that joins with another relation shall be asserted, and the second relation is known to be complete, then the completeness statement can again be split along any attributes of the second relation.

Example 7.4. Consider a completeness statement for all students. There exists a foreign key from the *student* table to the *class* table. So if the class table is known to be complete, individual completeness statements each for students that are enrolled in some class in the *class* table, and covering all the classes in the *class* table, are equivalent to a single completeness statement for all students. □

7.3 Stating Completeness with Cardinality Assertions

Cardinality assertions are a weaker way (requiring less knowledge of the ideal database) to give local completeness statements. Instead of saying that a certain part of the available database is the same in the ideal database (each tuple in that part which is in ideal database is also in available database), one just states how many tuples of that kind are present in the ideal database. Then a (count) check

is performed on the available database, whether this number of tuples is there. This works because our assumption of partial databases is that they contain no tuples not in the ideal database ($\check{D} \subseteq \hat{D}$). The idea of cardinality assertions was first given by Miliaev.

Originally, when an agent gives a local completeness statement he needs information about all tuples in that part in the ideal database, to check whether each of them is present in the available.

With cardinality constraints, it is sufficient to know that no wrong tuples enter the database together with the number of tuples present in that part of the ideal database. Moreover, this two preconditions can be separated among two agents.

Example 7.5. Consider in a school a voluntary school trip is organized. There is a notice on the notice board where every student taking part in puts his name. To complete registration, students have to print and fill in a form with personal information and e.g. special wishes and hand it in in the school secretary. The secretary puts all information in a new database table. When do we know that that table is complete?

The student secretary is the agent who can easily give the assertion that no wrong records entered the database, because it doesn't put in data for which no form was handed in, and students do not hand in forms before they did not register. The organizing teacher gives the cardinality assertion how many people have registered by counting the number of names on the board (plus maybe additional students who were ill and registered via phone and handed in the form to the secretariat via email). If this number matches the number of records in the table, then the table is complete.

That follows although the teacher does not know whether the records in the table are correct, nor the secretariat knows how many should be there. [1] □

[1] Deducing completeness using cardinality assertions seems to be an interesting basic human reasoning principle. After a written exam, from what does a teacher conclude that he has collected all exams? He makes the reasonable assumption that he has collected no invalid exams (e.g. because a student submitted two or a student not being present in the class submitted one), counts the number of students in his class and the number of exams he has collected, and if matching, concludes completeness. Observe, he does not need to check whether for every student present in the class he has collected an exam with the students name (which would correspond to the "classical way" of stating completeness, checking for every entity that should be there whether it really is there).

Chapter 8

Implementation Issues

In this chapter we discuss implementational issues of systems for managing and tracking database completeness. Parts of the theory presented in this thesis were implemented in a prototypical system for managing database completeness during the Magik project, a project that is under development at the Free University of Bozen-Bolzano.

In section 8.1, we discuss general issues of interest regarding the construction of such a system. In section 8.2, we present an efficient algorithm for finite domain containment, as this is a problem not having been investigated so far. Section 8.3 present the implementation itself.

8.1 General Issues

In this section we discuss general issues that require thought in an implementation of a system for tracking and managing database completeness. The observations discussed in this section are mostly based on our experiences during the development of the MAGIK system.

The first issue concerns the *interplay between keys and foreign keys*. Key constraints are a necessary precondition for foreign key constraints. Foreign key constraints are the foundation of several important results in this thesis (automated incompleteness checking, partitioning by table). Thus, any system that shall be sufficiently useful should support them. Both constraints also allow to conclude more containment (see chapter 6). However, containment in presence of keys and foreign keys is generally undecidable [JK82]. Thus, one either has to accept a system which is not complete (that is, it does not always find query completeness when it holds), or, one has to put certain restrictions on the key and foreign key constraints allowed, which goes so far as to disallow weak entity sets.

Related to this issue is also the question of where the schema information comes from, whether it is extracted from database constraints or inserted by a user, which then may also lead to wrong or contradictory schema information.

A second issue is the *design of such a system*. For functionality itself it does not matter much how the interface is designed. However, a system that shall support humans in managing and tracking database completeness but which itself is not well structured and understandable, would not make much sense. We found it not trivial to design a user interface and to decide which is both comprehensive and does not lack any information. Also, requiring scalability makes finding an appropriate interface difficult.

A conceptually interesting problem is that of completeness management of a *changing database*, possibly a temporal database. In this thesis we did not investigate in this question, however completeness management over changing data raises several conceptual questions. What happens if data is inserted in a part of a relation that was asserted to be complete? Should certain database changes trigger related completeness statements to become invalid? In temporal databases, how long can completeness statements be valid? What happens when changes to data about the past are made?

The last issue is, that, in order to enable several important features presented in this thesis, specification and *reasoning with finite domains* is necessary. While specification is not a problem, we are not aware of any discussion of query containment under finite domains in the literature. Conceptually, finite domain containment (presented in 6.2) is not complicated. However, as shown there, it is generally Π_2^P-hard, whereas completeness reasoning without comparison predicates is still NP-complete for simple conjunctive queries. Thus, finding an implementation that is still efficient for normal finite domain containment problems, becomes important. We present an algorithm in the next section.

To conclude this section, let us rephrase that there are important issues regarding the reasoning power, the design, and the evolution over time. As we are not aware of any similar system, there exists no prior experience from other sources about implementing such a system.

8.2 Practical Finite Domain Containment

In this section we present an efficient algorithm for finite domain containment. Finite domain containment is generally Π_2^P-hard, as in general, containment for all

possible instantiations with respect to the finite domains has to be tested. Therefore, it becomes important to make only as few instantiations as really necessary.

Our algorithm only works in the case of single-attribute finite domain constraints. And it is not obvious how to extend it to non-single-attribute finite domain constraints. Nevertheless we present the algorithm here, because both we believe single-attribute finite domain constraints to be useful in practice, and the algorithm is a good basis for future work.

Let Q and Q_1 to Q_n be queries, and F_1 to F_n be finite domain constraints. W.l.o.g. let us assume that all equalities in the queries have been eliminated (that is, whenever there was an equality atom stating $x = y$, the atom has been removed and every occurance of one of the symbol has been replaced by the other).

For testing whether $Q \subseteq_F Q_1 \cup \ldots \cup Q_n$, naively one would have to check for every possible instantiation of Q, whether it is contained in $Q_1 \cup \ldots \cup Q_n$.

However, not all instantiations make sense. Consider Q_1 to Q_n to be queries containing no constants at all. In that case, for no instantiation of Q containment could hold, if it did not hold already for Q. So, instantiating could be completely saved in this simple case.

In the following, we present an algorithm, which only instantiates variables if containment for all instantiations of that variable is possible. The basic idea is to check for every finite domain constraint variable in Q, whether all finite domain values are present in Q_1 to Q_n at positions, such that containment for the instantiated versions of Q may hold. Only if this is the case, the variable is instantiated.

The algorithm proceeds in three steps: First, it determines the actual finite domains of the variables in Q. Second, it identifies which of the finite domain constrained variables are worth instantiation. Third, it checks containment for all such instantiations of Q. We describe the first two steps more in detail next.

1. Finite Domain Identification

In the first step, the actual finite domains of variables are determined. Due to possible multiple occurrences, variables can have multiple finite domain restrictions. The overall finite domain restriction is then the intersection of the different finite domains.

Example 8.1. Consider a query

$$Q(N) :- \mathit{person}(N, G), \mathit{student}(N, L, C)$$

asking for all names of persons which are also students. Suppose, there are finite domain constraints F_1 and F_2 as follows:

$$F_1 = \mathit{Dom}(\mathit{person}, \mathit{name}, \top, \{\, \mathit{andrea}, \mathit{beatrix}, \mathit{chiara}, \mathit{daniel} \,\})$$

$$F_2 = Dom(student, name, \top, \{ \, chiara, daniel, erik, francesco \, \}).$$

Both F_1 and F_2 apply to the variable N, and hence the actual finite domain of the variable N is the intersection of the two finite domains from F_1 and F_2, which is $\{ \, chiara, daniel \, \}$. □

Finite domain constraints may come with restrictions, therefore it is additionally necessary to determine which finite domain constraints really apply, by checking whether there is a homomorphism from the finite domain constraint's restriction to the query.

Example 8.2. Consider the query from above, and consider a finite domain constraint

$$F_3 = Dom(person(N', G), 1, student(N', L, C), L = 1, \{ \, erich, francesca, gustav, hanna \, \}).$$

It is a finite domain constraint for the names of persons which are students in first class.

It is not applicable to the variable N in Q, because it restricts only a special subclass of the person names, which we do not query in Q.

Formally, that can be found because there exists no homomorphism from $Q_{F3}(N', G) :- person(N', G), student(N', L, C), C = 1$ to Q, where N' is mapped to N. □

To compute the actual finite domains of variables in a query Q, we propose to iterate once over every position in the query. For every position, the finite domain constraints applicable have to be computed, and the applicable finite domains then have to be intersected with the actual finite domain constraints computed so far.

2. Determination of Instantiation Worthiness

In this step, it is determined which variables with finite domains are worth instantiating. Roughly speaking, a variable is worth instantiation, if all the values in the finite domain appear in some query Q_1 to Q_n at a position in a relation where also the variable appears in Q.

This step can again be divided in four substeps:

1. First, for every variable in Q with finite domain, all its position in relation symbols in Q are determined.

2. Second, for every relation symbol in Q, all symbols appearing at all its positions in Q_1 to Q_n are determined separately

3. Third, for every variable all symbols which appear at its positions in Q_1 to Q_n are calculated (that is, since a variable may appear at several positions, a set union operation).

4. Finally, for each variable the finite domains and the set of symbols occurring at its positions in Q_1 to Q_n are compared. If all elements of the finite domain appear in Q_1 to Q_n, then the variable is worth instantiation.

Example 8.3. Consider a query

$$Q(N) :- \mathit{person}(N, G), \mathit{student}(N, L, C)$$

that asks for the names of all persons which go in some class. Furthermore, consider finite domain constraints for person genders to be only male or female, and for student levels to be only 1 to 3. Consider queries Q_1 to Q_4 as follows:

$$\begin{aligned}
&Q_1(N) :- \mathit{person}(N, \mathit{male}),\\
&Q_2(N) :- \mathit{person}(N, \mathit{female}),\\
&Q_3(N) :- \mathit{student}(N, L, C),\\
&Q_4(N) :- \mathit{student}(N, 3, C), \mathit{person}(N, G).
\end{aligned}$$

Table 8.1 shows the positions of the variables in Q, table 8.2 the symbols appearing at these positions in Q_1 to Q_4.

G	L	N
person/2	*student*/2	*person*/1, *student*/1

Table 8.1: Positions of variables in Q

person/2	*student*/2
male, female	1, 3

Table 8.2: Symbols appearing at positions where finite domain constraints are applicable

Thus, variable G is worth instantiation, because both male and female occur at its positions in Q_1 to Q_4, whereas L is not worth instantiation because the symbol 2 occurs nowhere. □

3. Instantiation Containment Checking

After having determined the finite domain constrained variables in Q that are worth instantiation, the remaining step of our algorithm is a standard containment check between all instantiations of Q with respect to the variables worth instantiation, and the queries Q_1 to Q_n.

We omit a formal investigation but only give an informal argument why our algorithm is correct. If the algorithm would test containment for every instantiation of Q, it would be trivially correct. It omits the instantiation of a variable V with an applicable finite domain, if one constant symbol c of the finite domain cannot be found at a position in a relation symbol in the queries Q_1 to Q_n where also V appears in Q. For containment of all the instantiations of V to hold, in particular it would also have to hold for V being instantiated with c.

For conjunctive query containment to hold, existence of a homomorphism from a query Q_i to each linearisation of Q is necessary. As c does not appear in any Q_i, either the homomorphisms do not map anything to c, or they map a variable to c. In both cases then the instantiation did not lead to a homomorphism and therefore containment which did not hold also without the instantiation.

8.3 The MAGIK Implementation

An implementation of a tool for managing completeness of databases is currently under development at the KRDB group of the faculty of Computer Science at the Free University of Bozen-Bolzano. It is developed as part of a research project with the title 'Managing Incomplete Knowledge' (MAGIK), thus we refer to it as the Magik implementation.

The MAGIK project is done in cooperation with the IT department of the school administration of the province of Bozen-Bolzano. Aim of the implementation is to give a proof of concept for the theoretical ideas about managing database completeness, and in particular allow tracking and managing database and query completeness.

The implementation started as part of Miliaevs Bachelor thesis and is continued by him. The implementation itself was not part of this Diplom thesis, however we collaborated on theoretical foundations of the system, the conceptual design and algorithmic decisions.

The first basic functionality Magik provides is the connection to existing databases. From existing databases, schema and instance information is extracted automatically. Then queries and completeness assertions can be formulated, and the com-

pleteness status of the queries with respect to the completeness statements can be analyzed. Furthermore a main functionality is that for given queries, the required completeness statements can be seen, and options for partitioning them as presented in chapter 7 are given.

Technically, the implementation consists of two parts: The core ('Magik'), and a web interface ('Charm'). The core uses its own database to store completeness, schema and query information and provides functionalities for managing and reasoning about it. The web interface provides convenient access to the core.

Magik is able to automatically extract schema information as keys, foreign keys and finite domains from SQL-databases. Also it allows the manual specification of those. Schema information, queries and completeness statements over databases are stored by Magik, and Magik automatically reasons about query completeness whenever completeness or schema information is changed.

The reasoning system implements the theory presented in this thesis. For every query, it computes its canonical completeness conditions and checks whether they are entailed by the present completeness statements. Also, it implements the check algorithm presented in 6.4, which allows to decide completeness of certain required completeness statements automatically. It provides functionality to partition completeness statements according to the methods shown in chapter 7, and allows specification of completeness statements with cardinalities. The finite domain containment does not work correctly at the moment, but is in ongoing development.

The web interface, Charm, provides a graphical user interface to Magik.

It provides different pages which can be seen in picture 8.1.

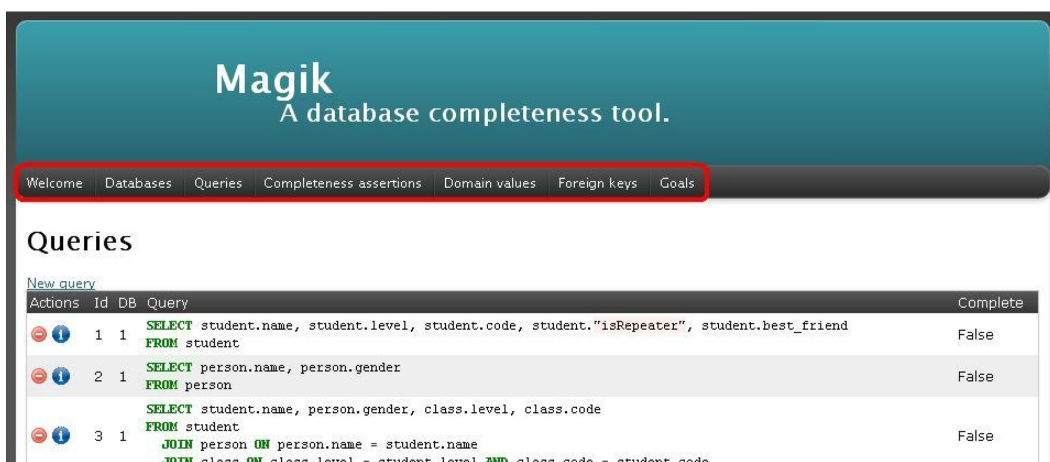

Figure 8.1: Overview of the Charm pages

- *Databases*: Allows to specify connection strings to databases, for which completeness shall be managed.
- *Queries*: Provides all informations about queries. Queries are listed together with their completeness status, additionally for each query a details page can be opened, which allows to see the canonical completeness conditions and allows stating and partitioning them
- *Completeness Assertions*: Lists all asserted completeness statements and cardinality assertions sorted by tables. For cardinality assertions, it is shown whether they hold.
- *Domain Values*: Shows finite domains of attributes extracted from the database schema and allows user specification of finite domain attributes.
- *Foreign Keys*: Shows foreign keys extracted from the database schema and allows user specification of foreign keys.
- *Goals*: Shows the canonical completeness conditions of all stored queries, possibly partitioned, and which of them hold following from the given completeness assertions.

As a main aim of the Magik implementation is to not only provide the reasoning facilities for query completeness from given local completeness statements, but the active support in ensuring query completeness, special effort has been put into facilitating this task. On the page that shows query details, first the query graph as described in section 6.4 is shown. The completeness status of the canonical statements for each literals in this graph is shown by colors (asserted complete, not asserted complete, complete by check). Below then, for each canonical statements options for partitioning are given.

Although the implementation is far not finished with respect to the theory analysed in this thesis, it already shows that important features can well be turned into practice.

Figure 8.2: Detail page for a query. At the top the query in SQL syntax. Below the query graph. In red the literals for which canonical completeness has to be asserted, in yellow the literals for which it can be checked. The canonical completeness goal for the student relation has been split along the person.gender attribute. For students with gender female, satisfaction of the completeness statement has been asserted, for students with gender male not. This is visualized at the bottom of the page in the goal partitioning forest.

Chapter 9

Conclusion

In this thesis, we have discussed the problem of query completeness over incomplete databases. Most notably, we have provided a solution to the problem of deciding query completeness with respect to local completeness that is generally decidable, which was not the case in earlier work.

In chapter 2, we gave a general motivation for the relevance of the problem and showed its placement in the field of general incomplete information. We presented a real-world example of a school database system. We discussed earlier work by Motro, Levy and Miliaev. For Motro's work, we discussed that the idea of stating database completeness in terms of view completeness has limitations in its practical usability. We recapitulated Motro's method for deciding query completeness is incomplete. For Levy's work, we showed that the reduction to the problem of query independence from updates is a reduction to a generally undecidable problem. Also, we discovered several technical limitations of his work. For Miliaev's work, we discussed the useful practical results and observations made in it.

In chapter 3 we gave an extensive formalization of the expressions used later in our work. We recapitulated the concept of partial databases, and, most important, we introduced the concept of completeness statements, of which query completeness statements and local completeness statements are special cases.

In chapter 4, we characterized necessary and sufficient conditions for query completeness under set and multiset semantics. Most importantly, we discovered that the canonical completeness conditions are sufficient but not necessary conditions for query completeness under set semantics, and that no set of local completeness statements can express the characterizing conditions for query completeness.

In chapter 5, we showed that local completeness statement entailment can be reduced to query containment, and that a query can be concluded to be complete exactly if its canonical completeness conditions hold. Furthermore, we discussed

the problem of completeness reasoning in presence of query completeness statements, and showed its relation to the undecided problem of decidability of query determinacy.

In chapter 6 we discussed the effect of schema and extensional information on the completeness reasoning. For schema information, we introduced the concept of finite domain constraints, and explained how query containment in presence of such constraints works. For extensional information, we showed that it can allow to derive more finite domain constraints, which then can lead to more completeness deductions. For the combination of schema and extensional information, we extended and formalized the main result of the work of Miliaev, which says that for some literals in a query, that are related to other literals by a foreign key join, holding of the canonical conditions does not need to be asserted but can be checked on the extensional information.

In chapter 7 we discussed the conceptual problem of stating completeness for parts of a database, and how the theory of the previous chapter can help in this task by allowing to split completeness conditions in smaller, equivalent ones.

In chapter 8 we discussed general issues an implementation of a completeness management system has to take care of, and put a special focus on containment under finite domain constraints. Also we presented the prototypical implementation in the MAGIK project.

The most important items left open by this thesis are whether the problem of completeness reasoning with respect to query completeness statements can be reduced to the problem of query determinacy, and, how completeness management can work in practice with respect to databases changing over time. For the first item, we were not able to come up with a proof or a reduction so far. For the second item, we need more insight into application scenarios of completeness management.

Beside that, future work might include a generalization of the query and completeness statement language to a language allowing safe negation. Also, local completeness statements for views of relations seem to be an interesting object to consider. As the work of Levy, our work can easily be extended to databases being partially incorrect, although we do not consider this to be of much practical relevance. Finally, this work focused on incompleteness of data only in the meaning of presence or absence of tuples. However, incompleteness can also appear within tuples, that is by tuples that may have an undefined value at some positions. This option is ignored in this thesis completely.

Bibliography

[AHN95] Serge Abiteboul, Richard Hull, Victor Vianu, *Foundations of Databases* Addison-Wesley 1995.

[Mo89] Amihai Motro, *Integrity = validity + completeness.* ACM Trans. Database Syst. Volume 14, 1989.

[Mi10] Dmitrij Miliaev, *Ensuring answer completeness of queries over administrative databases* Bachelor thesis, Free University Bozen-Bolzano, Italy, 2010.

[LS93] Alon Y. Levy, Yehoshua Sagiv, *Queries Independent of Updates* Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

[El90] Charles Elkan, *Independence of logic database queries and update* Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, Pages 154-160, Nashville, United States, 1990

[Le96] Alon Y. Levy, *Obtaining Complete Answers from Incomplete Databases* Proceedings of the 22nd VLDB Conference, Mumbai(Bombay), India, 1996.

[SV05] Luc Segoufin, Victor Vianu, *Views and queries: determinacy and rewriting* Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2005.

[JK82] D. S. Johnson, A. Klug, *Testing containment of conjunctive queries under functional and inclusion dependencies* PODS '82: Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems, 1982.

[LLMC09] Jianguo Lu, Minghao Li, John Mylopoulos, Kenneth Cheung, *Complete and Equivalent Query Rewriting Using Views* APWeb/WAIM '09: Proceedings of the Joint International Conferences on Advances in Data and Web Management, 2009.

[EGW94] Oren Etzioni, Keith Golden, Daniel Weld, *Tractable Closed World Reasoning with Updates* Proceedings of the Conference on Principles of Knowledge Representation and Reasoning, KR-94, 1994.

[SY81] Yehoshua Sagiv, Mihalis Yannakakis, *Equivalence among relational expressions with the union and difference operators* Journal of the ACM, 27(4):633-655, 1981.

[M92] Ron van der Meyden, *The Complexity of Querying Indefinite Data about Linearly Ordered Domains* Proceedings of the eleventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, p. 331 - 345, 1992.

[St76] L. J. Stockmeyer, *The polynomial-time hierarchy* Theoretical Computer Science, vol.3, pp.122, 1976.

[IL84] Tomasz Imielinski, Witold Lipski, *Incomplete Information in Relational Databases* Journal of the ACM volume 31, p. 761-791, 1984.

[C75] T. Codd, *Understanding relations (installment 7)* FDT Bull. of ACM Sigmod 7, pages 2328, 1975.

[D97] Oliver M. Duschka, *Query Optimization Using Local Completeness* Proceedings of the Fourteenth National Conference on Artificial Intelligence AAAI-97, 1997.

[DRS09] Nilesh Dalvi, Christopher R, Dan Suciu, *Probabilistic databases: diamonds in the dirt* Communications of the ACM Volume 52 Issue 7, 2009.

[B03] Franz Baader et al., *The Description Logic Handbook* Cambridge University Press, 2003.

Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Dresden, den 29. November 2010