FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Logic Programming Approach to Query Completeness

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

im Rahmen des Studiums

Computational Logic (EMCL)

eingereicht von

Sergey Paramonov
Matrikelnummer 1127778

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Prof. Dr. Thomas Eiter, Prof. Dr. Werner Nutt
Mitwirkung: Ognjen Savkovic

Wien, 03.01.13

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

A Logic Programming Approach to Query Completeness

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science (M.Sc.)

in

Computational Logic (EMCL)

by

Sergey Paramonov
Registration Number 1127778

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Prof. Dr. Thomas Eiter, Prof. Dr. Werner Nutt
Assistance: Ognjen Savkovic

Vienna, 03.01.13

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Sergey Paramonov
Lienfeldergasse 60C, 16 1160 Wien

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

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First of all, I would like to thank prof. Thomas Eiter for his help on the answer set programming theory and his course “the Theory of Knowledge Representation”.

I would like to thank prof. Werner Nutt for his course “Information Integration” that made me able to start working on this topic and for his help in explaining theory, verifying everything and discussing the directions of the research.

I would like to thank Ognjen Savkovic for his help in almost every step of the work, especially in the implementation of MAGIK, where he has played a key role and also for his help in analyzing my writing.

I would like to thank Simon Razniewski for his clarifications and work on the topic.

Last but not least I would like to thank my parents and Irina for their infinite patience.

Abstract

How to manage incomplete information has been studied in database research almost from the beginning. The focus was on how to represent incomplete information and how to compute certain answers. Less attention has been given to describing which parts of a database are complete and how to find out whether a query returns complete answers over a partially complete database.

To address these questions, we build on previous work by Motro (1989) and Levy (1995) who formalized when a query is complete over a partially complete database and what it means that parts of the tables are complete. Recently, Razniewski and Nutt (2011) characterized the complexity of various reasoning tasks in this setting. It was open, however, how to implement completeness reasoners in practice.

In this work, we introduce the problem of query completeness reasoning and show that it can be mapped elegantly to answer set programming (ASP) over datalog with negation. Then we consider extensions of the original problem that take into account foreign key constraints and finite domain constraints. To encode the extensions, we make use of Skolem functions and of datalog rules with disjunctions in the head. We show the correctness and completeness of the encodings by several characterization theorems. We also discuss a possible approach that takes into account comparisons and show how it can solve this problem if comparisons satisfy some syntactical restrictions.

With our encodings we can solve completeness reasoning tasks using the DLV system, which implements answer set programming for disjunctive logic programs. DLV is being developed at TU Vienna and at the University of Calabria.

Contents

CHAPTER 1

Introduction

Dealing with databases, it is quite common to assume that everything outside of the relation tables is false. In other words, if a fact is not known, then it is false. In practical scenarios, this assumption leads to inconsistency in case of distributed information.

It becomes obvious that this assumption does not hold, in case of physically and logically distributed information sources. By definition, every information source stores only a part of the database. Every source contains only partial information. In particular, in the fields like data exchange and data integration this problem becomes more pressing. In this case, aggregation of information like statistical reports [17], web-aggregators and wrapped-bases systems [16] cannot be considered reliable without completeness analysis.

Furthermore, there are plenty of natural and common reasons for data to be incomplete: ignorance about parts of the domain, maintenance problems, inconsistent updates, accidental deletions of data, corruption of parts of the database [20].

For some time in database theory the closed world assumption (when the whole database is complete; everything outside of it is false) had been considered as an essential assumption [13]; however, it has become obvious that it is not always the case. In real-life, it is more realistic only for a part of the database to be complete [19], [9] (that is referred as Extended Closed World Assumption). In this context there are well-known and investigated problems like computing of certain and possible answers.

On the contrary, a situation when only some parts of the database are complete had not been investigated in detail until a very recent time [17], [4], [20], especially from an algorithmic and complexity point of view. We are addressing the following problem: how to design an efficient completeness reasoner in practice.

We consider query completeness theory presented by Nutt and Razniewski [17] as a strong theoretical foundation, nevertheless, real algorithms implementing the current theory of query completeness are missing. In this work, we concentrate on offering practical implementations of the theory.

The problem, when the closed world assumption holds only locally, can naturally occur in the field of data quality. In general, in this field questions about properties of data such as

accuracy, completeness and consistency are under investigation. Our problem can be viewed as a problem in this field. We describe completeness analytically, precisely and not by means of an approximation or statistics.

The problem of query completeness can also arise in the field of information integration. In this setting, every data source has only local information. This leads to a notion of locality and local completeness of the sources. It brings up the question when a query can be answered completely if only some parts of the sources are known to be complete.

1.1 Motivation

High quality data is critical to success in the information age. The Data Warehousing Institute estimates that data quality problems cost U.S. businesses more than 600 billion dollars a year [6]. From a data quality point of view it gives us a direction to investigate. It also indicates what kind of features must be essentially reflected in the work to be applicable in practise.

Data quality degenerates over time. Experts say, two percents of records in a customer file become obsolete in one month because customers die, divorce, marry, and move [6]. This explains possible reasons for data to be inadequate. It is an important question to analyze possible sources of data incompleteness. It is essential to provide a description of data with respect to the interesting properties that are relevant for the industry.

It is important to reason about quality of data to provide a decision support. It is uncommon for existing systems to analyze data completeness in the databases. However, it is an essential property for making a decision based on a set of data, especially, to reflect how complete the data is.

A lack of practical algorithms that reason about completeness quality of the data is the main motivation for this work.

1.2 Related Work

In this section, we discuss different works in the area of query completeness. We also provide an overview of the most notable works that influenced this thesis. Even though all the works listed below are prominent, only few of them are really relevant for the problems we focus on in this work.

We divide them into two groups, the first of which consists of the works discussing the problem from the same perspective. They make a significant contribution to this thesis. We denote them as “Closely related work”. The second group consists of the works with significantly different assumptions. They go into a different direction and investigate the problem from a different point of view. Their results are not applicable in our setting, we denote them as “Remotely related work”.

In our setting, we investigate the problem when completeness of parts of a database entails completeness of a query. We do not consider a query to be fixed but we restrict the class of queries.

Closely related work

Theoretical foundations of the query completeness problem were investigated by Levy [12] and Motro [14]. In their works they provided only initial and basic definitions and introduced the problems without any decision procedure or any criteria how to establish query completeness except of trivial cases. Alon Levy in his work showed that the query completeness problem can be in general reduced to the query independence from updates. It is an undecidable problem. He was also the first to point out that in some cases query completeness follows if we take into account a database instance in reasoning. However, reasoning with a database instance lies beyond of the scope of this work.

Recently, Nutt and Razniewski have established complexity results for different classes of query languages and theoretical foundations that can be used as criteria for a decision procedure [17]. However, no practical algorithm was provided. Even though query completeness has a very strong theoretical background, it lacks implementations of systems and tractable decision procedures for this kind of problems.

Remotely related work

There are several works that provide a decision procedure for related problems or for the similar problem but with different assumptions. Demolombe provided a general calculus based on the Motro's work that can decide whether a complete query entails completeness of another query [3]. However, complexity and even termination of such algorithms were not investigated. It is worth mentioning that Motro and Demolombe as well presented rewriting procedures deciding query completeness, however, these procedures were definitely not complete. The main reason is that the considered class of queries is not a rewriting complete language [15]. It means a query might be complete, even though the rewriting algorithm described in [3] cannot establish this fact.

Assuming the domain to be finite, Denecker et al. used an approximation algorithm taking additionally into account a database instance [4]. They showed that under the finite domain assumption the query completeness problem with a fixed query and a database instance belongs to the *CoNP* complexity class. They also provided some conditions under which their reasoning becomes precise. This work differs from the present thesis in the following: we do not make a finite domain assumption, we define completeness differently, we do not work with an instance, we do not assume the query to be fixed.

Fan et al. investigated the problem of query completeness in [8]. However, this work heavily relies on a notion of master data. It represents an upper-bound of the given data. This strong assumption makes this work incomparable. Definitions of their work depend on this notion. Their decision procedure cannot be used or adapted in this work.

This related work shows that different problems (approximation reasoning, reasoning with an instance, reasoning with master data) can be described and analyzed under the title of "Query Completeness", however, all of which are orthogonal to the problems described in this thesis.

1.3 Contribution

Aims

There are several aims of this work.

- To develop a sound and complete algorithm that reduces the problem:

  When does completeness of parts of the database entail completeness of the query?

  into the evaluation of an answer set program.
- To automate the decision procedure for query completeness with respect to a query and a set of table completeness statements.
- To develop a system working on top of existing database systems and reasoning about completeness of the data.

Obtained results

There are several points in which this work makes a contribution.

- It provides a well-defined decision procedure for completeness reasoning with respect to the different languages and constraints.
- It provides an encoding of the query completeness problem into the evaluation of an answer set program. The reduced problem is always as hard as the initial problem. Simply speaking, it requires exactly the computational power as required to solve the problem. Strictly speaking, the given problems are in the same complexity class as the problems we reduce them into.
- It provides an implementation of the reasoning system as a web application.

1.4 Thesis Structure

The thesis has the following structure:

- Chapter 3 introduces the problem of completeness reasoning from a formal perspective. It starts with an example. Then, it introduces a decision procedure of the problem under set and bag semantics without any schema constraints. It presents an encoding for both types of semantics.
- Chapter 4 introduces the problem in the presence of finite domain constraints. It starts with an example. It introduces a decision procedure and an encoding theorem for the reasoning problem in the presence of additional constraints.
- Chapter 5 introduces foreign key constraints. It provides a decision procedure and an encoding theorem for completeness reasoning in the presence of foreign key constraints.

- Chapter 6 indicates an approach to reasoning with built-in predicates. It proposes a possible way to reduce this problem to the completeness reasoning problem with finite domain constraints.
- Chapter 7 describes the implementation of the reasoning system that has been developed based on the algorithms and the decision procedures and theorems presented in the work.
- Chapter 8 discusses conclusions, limits of the approach and possible future work.

CHAPTER 2

Preliminaries

2.1 Database Theory

Databases and queries

We start with a definition of a relation schema which is a relation name and an ordered list of attributes.

We say that a *signature* Σ is a set of relation schemas $R_1/n_1, \ldots, R_k/n_k$ where $n_1, n_2, \ldots$ are arities of relations, normally, we refer to the relation schema by its name e.g. R_5.

We assume it is known what it means for an interpretation I and an assignment α to satisfy a formula ϕ

$$I, \alpha \models \phi. \tag{2.1}$$

A *term* is a constant or a variable. A *tuple* is a *finite* vector of terms which can be empty, this tuple has a special name and the notation $()$, it is called the *empty tuple*.

An *atom* is an expression of the form $R(\bar{t})$ where R belongs to the set of relations and $\bar{t}$ is a tuple. We say that $R(\bar{t})$ is a *ground atom* or a *fact* if $\bar{t}$ does not contain a variable.

We define a *database* D as a finite set of ground atoms. For a relation R and database D, we denote as $R(D)$ the set of all tuples in the R-atoms in D.

We define a *condition* G as a conjunction of atoms.

A *conjunctive query* (CQ) is an expression of the form

$$Q(\bar{X}) \leftarrow A_1(\bar{X}_1), \ldots, A_m(\bar{X}_m). \tag{2.2}$$

where $A_1(\bar{X}_1), \ldots, A_m(\bar{X}_m)$ are atoms, $\bar{X}$ is a vector of variables, the variables in $\bar{X}$ are called the *distinguished* variables of Q. If $\bar{X}$ is empty, then the query is a *boolean query*.

Definition 2.1.1 (Answer Tuple) *Let $Q(\bar{X}) \leftarrow B$ be a query, D be a database, then $\bar{c}$ is an* answer tuple *iff there is a mapping α s.t.*

- $\bar{c} = \alpha\bar{X}$;

- $D, \alpha \models B$.

where all operators like $\models$ are defined classically as in the first order logic. In case of a boolean query, the vector of distinguished variables is empty, and we say that the empty tuple $()$ belongs to the answer iff there is an α that satisfies the second condition above, otherwise the answer is the empty set $\emptyset$.

An answer to a conjunctive query $Q(\bar{X})$ with a finite vector of distinguished variables X over a database D is the set of all answer tuples which is denoted as $Q(D)$.

We define a *prototypical database* D_Q for a query $Q(\bar{X}) \leftarrow A_1(\bar{t}_1), \ldots, A_n(\bar{t}_n)$ as a set of ground facts

$$\{A_1(\theta\bar{t}_1), \ldots A_n(\theta\bar{t}_n)\}. \tag{2.3}$$

where θ is an assignment such that for every variable X that occurs in Q it holds

$$\theta X = c_X. \tag{2.4}$$

where c_X is a fresh constant that does not occur anywhere else.

Example 2.1.1 *Assume Q is a query*

$$Q(X) \leftarrow A(X), B(X, Y). \tag{2.5}$$

then, the prototypical database D_Q is

$$\{A(c_X), B(c_X, c_Y)\}. \tag{2.6}$$

where c_X, c_Y are constants that do not occur anywhere else.

An intuition behind θ is the following: θ is a mapping that freezes variables. For every variable X, it introduces a fresh constant c_X and $\theta_{|\bar{x}}$ denotes a mapping that freezes only head variables $\bar{X}$. It is applied only to variables from the vector $\bar{X}$.

Database running example

In this subsection, we present a schema that we use for examples and illustrations.

The schema Σ has two relations

$$\textit{Employee}(\underline{\textit{Name}}, \textit{DeptName}, \textit{Birthday}). \tag{2.7}$$

$$\textit{Department}(\underline{\textit{DeptName}}, \textit{Location}). \tag{2.8}$$

In both relations the first column is a primary key, which is indicated by underscoring.

The central topic of this work is completeness reasoning. Simply speaking, we would like to answer the question if completeness of parts of a database entails completeness of a given query. The completeness reasoning problem consists of

- input: parts of a database that are complete;
- input: constraints on a schema;

- question: is a given query complete?

If the set of constraints is empty, we call it a *relational case*. In database theory queries can be evaluated under set and bag semantics, but the databases are still sets of facts. We consider two types of schema constraints: foreign keys and finite domain constraints.

In this work we consider the following reasoning cases:

- completeness reasoning under set semantics;
- completeness reasoning under bag semantics;
- completeness reasoning with foreign key constraints;
- completeness reasoning with finite domain constraints;
- completeness reasoning with foreign key and finite domain constraints.

Normally, SQL queries are evaluated under bag semantics since it is costly to remove duplicates. In SQL, set semantics is activated by the keyword DISTINCT. In this work, we always make a clear distinction which semantics is used.

To reflect FK (foreign keys) and FDC (finite domain constraints), we will change our schema by imposing additional constraints.

2.2 Completeness Theory

A real life example where query completeness can be applied has arisen from a story of a school information system (SIS) in the province of Bolzano. The current SIS is maintained in a distributed fashion and that this gives rise to incompleteness. The provincial administration asks for annual statistical reports to ensure completeness of information aggregation of students and schools. For yearly school statistics, however, completeness of those parts of the database that are relevant for the statistics is necessary.

Query completeness

To start discussion about query completeness, we introduce a definition of a partial database. When talking about the completeness of a database, there is an implicit reference to another database that is complete. Namely, there are two databases: the *ideal database* D^i that reflects the real world, what is really true, and the *available database* D^a that reflects the data we physically store. This idea is captured by the definition of Motro.

Definition 2.2.1 (Motro 1989 [14]) *A partial database $\mathcal{D}$ is a pair of databases (D^i, D^a) such that $D^a \subseteq D^i$.*

Note 2.2.1 *We work under the assumption that the available database is sound.*

To illustrate this definition, we construct a partial database, according to the schema from the previous section. As explained before, it is quite common for real databases to lose some data.

Example 2.2.1 *An example of the ideal database* D^i

Name	***DeptName***	***Birthday***
Gil	*Sales*	*10/10/87*
Sergey	*IT*	*25/10/88*
Peter	*IT*	*21/04/86*

DeptName	***Location***
Sales	*Trieste*
IT	*Vienna*

The available database D^a *might miss some facts. E.g., we miss the record with Gil in the Employee table.*

Name	***DeptName***	***Birthday***
Sergey	*IT*	*25/10/88*
Peter	*IT*	*21/04/86*

DeptName	***Location***
Sales	*Trieste*
IT	*Vienna*

It can be written using as a set equation:

$$D^a = D^i \backslash \{\textit{Employee}(\textit{Gil}, \textit{Sales}, 10/10/87)\}.$$

We would like to illustrate what it means for a query to be complete. In this example, we can use the commonsensical meaning of completeness. If we want to query only the relation *Department*, then the answer is complete. Intuitively, if an answer of the query must contain the tuple *(Gil,Sales)*, then the answer is not complete.

- "SELECT DeptName FROM Department" is complete;
- "SELECT Name FROM Employee " is not complete.

Now we formally state this intuitive description of query completeness.

Definition 2.2.2 (Motro 1989 [14]) *Let* Q *be a query and* $\mathcal{D} = (D^i, D^a)$ *be a partial database. Then,* Q *is complete wrt* $\mathcal{D}$*, written* $(D^i, D^a) \models \textit{Compl}(Q)$*, iff*

$$Q(D^i) = Q(D^a). \tag{2.9}$$

Several questions follow this definition. How can we in general state whether a database or a part of a database is complete? How can we obtain completeness statements (concerning the database) in general? In our setting, it is natural to assume that some people are sources of information: heads of the departments in the company, the head of the company itself, the HR department. In general, there is a variety of sources of completeness information. We illustrate this statement with an example.

"*Head of a department says: all employees of the sales department are in the database.*"

Levy introduced formal statements, which he called local completeness statements, to describe which parts of a database are complete [12].

Definition 2.2.3 (Table Completeness Statement) *Let R be a relation and G be a condition. We say that $\mathit{Compl}(R(\bar{s}); G)$ is a table completeness statement. It has an associated query $Q_{R(\bar{s}),G}(\bar{s}) \leftarrow R(\bar{s}), G$. The statement is satisfied by $\mathcal{D} = (D^i, D^a)$, written $\mathcal{D} \models \mathit{Compl}(R(\bar{s}); G)$, iff*

$$Q_{R(\bar{s}),G}(D^i) \subseteq R(D^a). \tag{2.10}$$

The condition G in 2.2.3 can be $\top$ (standing for "true", the trivial condition), written $\mathit{Compl}(R(\bar{s}); \top)$. It means the condition is trivially satisfied. In this case, in the definition 2.2.3 the associated query is $Q_{R(\bar{s})}(\bar{s}) \leftarrow R(\bar{s})$ and the rest of the definition stays the same.

Let us consider an example of a table completeness statement. In natural language, it can be formulated as:

"*All employees who work in some department located in Vienna are in the database.*"

We can formulate this natural language statement as a table completeness statement over the signature Σ from the previous section:

$$\mathit{Compl}(\mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}); \mathit{Department}(\mathit{DeptName}, \text{"}\mathit{Vienna}\text{"})). \tag{2.11}$$

The associated query is

$$Q_{\mathit{Employee}(\mathit{Name},D,B),\mathit{Department}(D,\text{"}\mathit{Vienna}\text{"})}(\mathit{Name}) \leftarrow \mathit{Employee}(\mathit{Name}, D, B), \mathit{Department}(D, \text{"}\mathit{Vienna}\text{"}).$$

Table and Query Completeness

In this section, we are going to discuss the main question of the work.

Informally speaking, we describe it as

"When does completeness of parts of the database entail completeness of the query".

Formally speaking, we describe it as Table Completeness Query Completeness (TC-QC) entailment, i.e. when a set of table completeness statements (TC) entails completeness of a query (QC).

Definition 2.2.4 (TC-QC Entailment under Set Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, then $\mathcal{C}$ entails completeness of Q under set semantics written $\mathcal{C} \models \mathit{Compl}^s(Q)$ iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \implies Q(D^i) = Q(D^a).$$

Simply speaking, the query returns the same answer for both D^i and D^a for any partial database (D^i, D^a) satisfying the set of constraints $\mathcal{C}$.

The setting of the school problem corresponds precisely to the informal description of TC-QC entailment. We formulate this completeness problem as described in the definition 2.2.4, that is why we consider TC-QC entailment in this work.

Local school records are represented as parts of the global database schema of the regional administration. We have completeness information only about these local records. The administration queries the database. They would like to ensure the answer to be complete.

2.3 Answer Set Programming

Answer set programming

In this section, we introduce the main definitions of answer set programming. We use them to define answer set programs later.

Definition 2.3.1 (Disjunctive Datalog Program) *A disjunctive datalog program is a finite set of rules:*

$$a_1 \vee a_2 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, \textit{ not } c_1, \ldots, \textit{ not } c_h.$$

Where $a_1, \ldots, a_n, b_1, \ldots, b_k,\ c_1, \ldots c_h$ are atoms of a function-free first order language L. The list $a_1, \ldots, a_n$ is the disjunctive head of the rule, the b-s are the positive part of the body and the c-s are the negative part of the body.

We refer to the head of the rule r as $H(r) = \{a_1, \ldots, a_n\}$ and the body as $B(r) = B^+(r) \cup B^-(r)$ where $B^+(r) = \{b_1, \ldots, b_k\}$ is the positive part of the body and $B^-(r) = \{c_1, \ldots, c_h\}$ is the negative part of the body.

Example 2.3.1 (A Disjunctive Datalog Program) *Let us illustrate this definition with an example.*

$$\begin{aligned} a \vee c &\leftarrow b. \\ b &\leftarrow a, \textit{ not } c. \\ a &\leftarrow a, \textit{ not } c. \\ \bot &\leftarrow \textit{ not } a. \\ a. & \end{aligned}$$

If a disjunctive datalog program P has variables, then its semantics is considered to be the same as the one of its grounded version of it (all variables are substituted with constants from the Herbrand Universe H_P). Semantics of a program with variables is defined by semantics of the corresponding grounded version.

Note 2.3.1 (Safety Condition) *All non-ground rules are considered to be safe i.e. every variable in the head or in a negative atom must occur in some positive atom in the body.*

Let us consider the program P

$$\begin{aligned} A(a). \quad A(b). & \\ B(X) \quad \leftarrow \quad A(X). & \end{aligned}$$

Then, the grounded version of it is *ground*(P)

$$\begin{aligned} A(a). \quad A(b). & \\ B(a) \quad \leftarrow \quad A(a). & \\ B(b) \quad \leftarrow \quad A(b). & \end{aligned}$$

It is called grounded because all variables are substituted with constants. This process of substituting variables is called grounding i.e. it makes all atoms ground.

An interpretation I w.r.t. to a program P is a set of ground atoms of P, e.g. $I = \{A(a), B(b)\}$. Let P be a positive disjunctive datalog program (i.e. without negation), then an interpretation I is called closed under P, if for every $r \in \mathit{ground}(P)$ it holds that $H(r) \cap I \neq \emptyset$ whenever $B(r) \subseteq I$.

Definition 2.3.2 (Answer Set of a Positive Program) *An answer set of a positive program P is a minimal (under set inclusion) interpretation among all interpretations that are closed under P.*

Example 2.3.2 *An answer set of both program above is*

$$\{A(a),\ A(b),\ B(a),\ B(b)\} \tag{2.12}$$

Definition 2.3.3 (Gelfond-Lifschitz Reduct) *A reduct of a ground program P wrt an interpretation I, written P^I, is a positive ground program P^I obtained by:*

- *removing all rules $r \in P$ for which $B^-(r) \cap I \neq \emptyset$;*
- *removing literals not a from all remaining rules.*

Intuitively, the reduct of a program is a program where all rules with bodies contradicting I are removed and in all non-contradicting all negative ones are ignored. The interpretation I is a guess as to what is true and what is false.

Example 2.3.3 (Reduct of Example 2.3.1) *Assume, we have guessed the interpretation $I = \{a, b\}$ then reduct of P wrt to I is $P^I =$*

$$\begin{aligned} a \vee c &\leftarrow b. \\ b &\leftarrow a. \\ a &\leftarrow a. \\ a. & \end{aligned}$$

An answer set of a disjunctive program P is an interpretation I such that I is an answer set of $\mathit{ground}(P)^I$.

Example 2.3.4 (Answer Set of 2.3.1) *Let us take an interpretation $I = \{a, b\}$ of P be a candidate for an answer set. Then, we compute the reduct P^I which is presented above. A minimal interpretation closed under P^I is $I = \{a, b\}$, therefore I is an answer set of P^I, that is why I is an answer set of P.*

For details on answer set programming we refer to [7], [11].

2.4 Integrity Constraints

Finite domain constraints

In this section, we briefly introduce finite domain constraints as they are described in the literature; we summarize this description in the context of completeness reasoning. We start with a definition and complexity results presented by Simon Razniewski, Werner Nutt in [18].

For practical reasons, we simplify the definition of a finite domain constraint. A finite domain constraint is a triple $F = \mathit{Dom}(R, i, M)$, where R is a relation, i is a positive integer and M is a finite set of constants. A database instance D satisfies a finite domain constraint F, if the projection on the position i of the extension of R is contained in M, that is, for every atom $R(t)$ in D, it holds that $\pi_i(R(t)) \in M$. We say a database D satisfies a set of FDC constraints $\mathcal{F}$, it it satisfies each of them, written as

$$D \models \mathcal{F}. \tag{2.13}$$

We say that a partial database $\mathcal{D} = (D^i, D^a)$ satisfies a set of FDC $\mathcal{F}$, written as $\mathcal{D} \models \mathcal{F}$ iff

$$D^i \models \mathcal{F} \text{ and } D^a \models \mathcal{F}. \tag{2.14}$$

Deciding TC-QC entailment is Π_2^P-complete [18].

Primary keys

In this section, we discuss primary key constraints. They are widely discussed in the literature; here we recall definitions and properties.

The following definition of satisfaction is due to Abiteboul et al. [1]: a database instance M satisfies a functional dependency $R : X \to Y$ if for each pair s, t of tuples such that atoms $R(t)$ and $R(s)$ are in M it holds that

$$\pi_X(R(s)) = \pi_X(R(t)) \implies \pi_Y(R(s)) = \pi_Y(R(t)). \tag{2.15}$$

where X and Y are lists of indices and $\pi_X(R(s))$ is a projection on the arguments s of R with the indices X. If R is clear from the context, we write the functional dependency without R, namely $X \to Y$.

Definition 2.4.1 *A key dependency is a functional dependency (FD) $X \to U$, where U is the full set of arguments of the atom. [1]*

In general, we assume that any given query complies with primary keys.
Assume, Q does not comply with primary keys. For example, Q can be

$$Q(\mathit{Name}) \leftarrow \mathit{Department}(\mathit{Name}, \text{“}\mathit{Bolzano}\text{”}), \mathit{Department}(\mathit{Name}, \text{“}\mathit{Vienna}\text{”}). \tag{2.16}$$

It needs to be reformulated to comply with the primary key. If Q is not unifiable according to primary keys, then the query is unsatisfiable and completeness is entailed trivially.

In this work, we assume that any given query is processed by the primary key compliance algorithm from Abiteboul et al. [1]. Without any loss of generality, we assume that all queries comply with primary keys.

Foreign keys

In this section, we are going to recall and discuss foreign keys (FK). Foreign keys are a combination of two types of constraints:

1. inclusion constraints (IND, inclusion dependency [1], [2], [10]);
2. primary key constraints (FD, functional dependency [1], [2], [10]).

We have already discussed primary key constraints; now we are going to discuss inclusion constraints.

According to Abiteboul et al. [1] an inclusion constraint is defined as follows:

Definition 2.4.2 *An* inclusion dependency (R, S, A, B) *(IND) is an expression σ where*

- R, S *are (possibly the same) relations;*
- $A = A_1, \ldots, A_m$ *is a sequence of indices;*
- $B = B_1, \ldots, B_m$ *is a sequence of indices.*

A database instance M is said to satisfy σ written as $M \models \sigma$ iff for every atom $R(t)$ and $S(s)$ in M it holds that

$$\pi_A(R(t)) \subseteq \pi_B(S(s)). \tag{2.17}$$

Then we say that $F = (R, S, A, B)$ written as $R[A] \subseteq S[B]$ is a foreign key, if the following conditions are met:

1. (R, S, A, B) is an inclusion dependency;
2. B is a key dependency in S.

We are going to introduce the definition of the foreign key satisfaction. A database instance M satisfies a foreign key iff both of these dependencies are satisfied.

Definition 2.4.3 *Let $\mathcal{K}$ a set of FK, M be a database instance, then M satisfies foreign key constraints $\mathcal{K}$ written as $M \models \mathcal{K}$, if for all k in $\mathcal{K}$, M satisfies k.*

To make examples easy to read we introduce the following piece of syntax.

Definition 2.4.4 *Let R and S be two relations. Let $\bar{X}$ and $\bar{Y}$ be vectors of variables. Let A and B be indices of the variables $\bar{X}$ and $\bar{Y}$. Then, we denote the foreign key $f = R[A] \subseteq R[B]$ by the following statement*

$$R(\_, \bar{X}, \_) \textit{ references } S(\_, \bar{Y}, \_). \tag{2.18}$$

In this statement, we explicitly mention the variables that are used in the foreign keys. By the sign "_", we denote the variables that are not used in the foreign key. This syntax allows us to impose foreign key constraints by referring to variables' names instead of indices.

Assumptions and Complexity

We are going to talk about the acyclicity assumption and complexity of completeness reasoning with foreign keys. We mainly refer to the foundational work of Johnson, Klug [10] on the complexity of query containment; we also refer to the complexity correspondence between TC-QC reasoning and query containment in Razniewski, Nutt [17].

The technique we use to reason with foreign keys is the chase procedure [5]. The chase procedure adds necessary facts to the model to satisfy constraints. Once it reaches a fix point when it cannot add any new fact, it stops.

We are going to justify our acyclicity assumption. The chase procedure in the presence of cycles makes reasoning unpractical since it may introduce infinite models; we also show why the chase procedure under the acyclicity assumption does not introduce infinite models.

Why cyclicity introduces infinite models We illustrate it by means of an example. Assume, we have a schema

$$A(\underline{X}, Y).$$
$$B(\underline{Y}, X).$$

Suppose, there are two foreign key constraints

$$A(\_, Y) \text{ references } B(Y, \_). \tag{2.19}$$
$$B(\_, X) \text{ references } A(X, \_). \tag{2.20}$$

Assume, Q is a query

$$Q(X) \leftarrow A(X, Y). \tag{2.21}$$

Assume, x is in the answer to Q, then for some constant y there must be an atom

$$A(x, y) \in D^i. \tag{2.22}$$

Due to the first foreign key 2.19, there also must be

$$B(y, f_B(y)) \in D^i. \tag{2.23}$$

where $f_B(y)$ is a Skolem term that depends only on the constant y. Due to the foreign key 2.20, there is also

$$A(f_B(y), f_A(f_B(y))) \in D^i. \tag{2.24}$$

Then, application of both 2.19 and 2.20 would introduce an infinite amount of atoms in D^i that are needed to construct a model that satisfies these constraints.

Why acyclicity does not introduce infinite models In this paragraph, we explain why the chase procedure does not introduce infinite models, if the foreign keys are acyclic. Observe, if all FK are acyclic, then there is a set of FK not referenced by any other FK. Let us denote this set as F_0.

To construct a model for the query, we need to satisfy F_0. We introduce new facts in the model. Having done so, all constraints in F_0 are satisfied. They can be ignored in the further reasoning. We drop F_0 from our set of FK constraints. We obtain a new set F_1 of FK, none of which is referred by any other FK. We repeat our procedure: introduce new facts, drop F_1. We know that the set $\mathcal{K}$ is finite and every step removes at least one constraint. After a finite number of step, we satisfy all FK and introduce only finite amount of facts in the database.

Summing up, we have constructed a finite model satisfying all FK.

CHAPTER 3

Completeness Reasoning

Overview

In this chapter, we are going to discuss completeness reasoning in the relational case. We assume, there are no constraints in the schema, and no built-in predicates in the query and table completeness statements.

First of all, we show how the reasoning procedure works in a typical case. We do it by means of an example. It works differently under set and bag semantics. The example is shown under set semantics.

Then, we introduce a characterization of TC-QC entailment under set semantics. It is a purely declarative method to check whether a query is complete with respect to a set of table completeness statements. The method constructs a special prototypical database and a boolean test query out of a query and a set of TC statements. If the boolean test query evaluates to true over the prototypical database, then TC-QC entailment holds.

In the encoding section, to make the characterization under set semantics feasible, we present a way to encode the evaluation a test query over a prototypical database into the problem of existence of an answer set of a datalog program.

In the same spirit, we introduce a characterization and an encoding under bag semantics. We start the section with an example showing the difference in reasoning between two semantics and justifying introduction of new concepts.

3.1 Example

Let us start with an example that shows how completeness reasoning works. The schema is the same as before:

$$Employee(\underline{Name}, DeptName, Birthday). \tag{3.1}$$

$$Department(\underline{DeptName}, Location). \tag{3.2}$$

Let Q be a conjunctive query

$$\begin{aligned} Q(\mathit{Name}) \quad &\leftarrow \quad \mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}), \\ &\qquad \mathit{Department}(\mathit{DeptName}, \mathit{Location}). \end{aligned}$$

The query Q asks for all names of employees working in some department.
Assume, we have two completeness statements, the first of which is:

"*All departments are in the database*".

In terms of the ideal and available databases, we can reformulated this:

"*If there is a department in the ideal database, then it must be in the available database*".

We formulate this as a table completeness statement

$$\mathit{Compl(Department(Name,Location)}; \top). \tag{3.3}$$

This completeness statement can be mimicked by a formal datalog rule. It connects a part of the ideal database with a part of the available database. Namely, it forces some tuples in the ideal database to be in the available database as well.

We generally refer to relations in the ideal database by using the upper index $\bullet^i$ and to relations in the available database by the upper index $\bullet^a$. We formulate this TC statement as a datalog rule

$$\mathit{Department}^a(\mathit{Name}, \mathit{Location}) \leftarrow \mathit{Department}^i(\mathit{Name}, \mathit{Location}).$$

The second statement says:

"Every employee in a department is in the database".

We formalize it as a datalog rule:

$$\mathit{Employee}^a(\mathit{Name}, D, B) \leftarrow \mathit{Employee}^i(\mathit{Name}, D, B), \mathit{Department}^i(D, L).$$

We have a conjunctive query Q and two table completeness statements, which we have expressed as constraints over an arbitrary partial database. These constraints enforce some tuples from D^i to be in D^a. We illustrate this by a completeness reasoning example.

By means of this example, we indicate the most important steps of the reasoning procedure. They also illustrate the general ideas behind the method.

According to the definition 2.2.2, a query is complete if the answers to the query over the ideal and the available databases are the same. That is why we start looking at the tuples that must be in the D^i to answer the query Q.

1. Determine what tuples have to be in D^i

Assume, there is a tuple *name* in the answer

$$name \in Q(D^i). \tag{3.4}$$

The query Q is a conjunctive: if there is a tuple in the answer, then there is a corresponding atom in the database

$$Employee(name, deptName, bday) \in D^i. \tag{3.5}$$

For the same reason, there must be a fact

$$Department(deptName, location) \in D^i. \tag{3.6}$$

According to the definition 3.4 both of the atoms must be in D^i

$$Employee(name, deptName, bday) \in D^i, \tag{3.7}$$
$$Department(deptName, location) \in D^i. \tag{3.8}$$

Note 3.1.1 *All arguments (name, bday etc.) are arbitrary but fixed.*

2. Determine what tuples have to be in D^a

We apply our table completeness rules (in datalog syntax) to the tuples in D^i. The meaning of TC statements is transferring tuples from D^i to D^a. Once we have established what tuples must be in D^i, we check which of them can pass to D^a through TC statements (they play the role of copying rules).

From previous calculations, we have a ground fact

$$Department(deptName, location) \in D^i. \tag{3.9}$$

We apply our TC rule to it

$$Department^a(Name, Location) \leftarrow Department^i(Name, Location). \tag{3.10}$$

As a result, we have the following atom

$$Department(deptName, location) \in D^a. \tag{3.11}$$

In the same way we conclude

$$Employee(name, deptName, bday) \in D^a. \tag{3.12}$$

Therefore both atoms are in D^a.

3. Verify completeness

According to the definition 2.2.2, a query Q is complete with respect to a partial database (D^i, D^a) iff

$$Q(D^i) = Q(D^a). \tag{3.13}$$

In our case, Q returns *name* as an answer tuple over D^a because both instantiated body atoms *Employee*(*name*, *deptName*, *bday*) and *Department*(*deptName*, *location*) are in D^a.

Since the constants are chosen arbitrarily, for any tuple $\bar{t}$ it holds that if $\bar{t}$ in the answer over the D^i, then it is in the answer over D^a. As a result, the query is complete.

3.2 Characterization

In this section, we characterize TC-QC entailment under set semantics. It allows us to check whether TC-QC entailment holds.

To design a decision procedure we use the same ideas as in the example from the previous section. We generalize the example to a characterization. That is the reason why we repeat many steps from the previous section:

1. We construct the most general answer to the query over the ideal database – the prototypical database.
2. We infer what tuples must be in the available database to satisfy TC statements.
3. We evaluate a special test query over the tuples ensured to be in the available database.

This procedure is inspired by the idea of the universal model. It has a property: if something holds in this model, it holds in any other model. As in the example, even though on every step we operate with fixed values, we can infer completeness in general.

Step 1: prototypical database

We start with an introduction of the prototypical database. It is the ideal database corresponding to a given conjunctive query.

Assume, Q is a conjunctive query:

$$Q(X_1, \ldots, X_n) = R_1(T_1^1, \ldots, T_m^1), \ldots, R_n(T_1^n, \ldots, T_k^n). \tag{3.14}$$

where R_i is a relation and T_i^j is a term for every i, j. The prototypical database D_Q^i for the query Q is

$$D_Q^i = \{R_1(t_1^1, \ldots, t_k^1), \ldots, R_n(t_1^n, \ldots, t_k^n)\}. \tag{3.15}$$

where every t_j^i is obtained from T_j^i by the "freezing" rule:

- if T_j^i is a constant, then $t_j^i = T_j^i$;
- if T_j^i is a variable, then $t_j^i = c_{T_j^i}$ such that constant $c_{T_j^i}$ does not occur anywhere else. We call them "fresh", because they have not been introduced anywhere else.

Note 3.2.1 (Intuition for the notation for a prototypical database) *We denote the prototypical database as D^i_Q because it is a **D**atabase that is needed to obtain an answer for the query **Q** over some hypothetical **I**deal database.*

We illustrate it with an example. Assume, we are given a conjunctive query:

$$\begin{aligned} Q(\mathit{Name}) \quad \leftarrow \quad & \mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}), \\ & \mathit{Department}(\mathit{DeptName}, \mathit{Location}). \end{aligned}$$

We need to construct the most general answer. We "freeze" all variables and treat them as "fresh" constants. Having applied "freezing" to Q, we obtain a set of ground atoms

$$\begin{aligned} \{&\mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}), \\ &\mathit{Department}(\mathit{deptName}, \mathit{location})\}. \end{aligned}$$

Given this database, we obtain the constant *name* as an answer tuple to Q. The constant *name* has an arbitrary but fixed value. That is why we refer to it as the most general answer. If we substitute *name* by "John Smith" everywhere in D^i_Q, we obtain "John Smith" as an answer tuple.

Step 2: complete parts of the database

A tuple must be in every available database only if it is forced by a table completeness statement. We introduce a function selecting all tuples that must be in every available database satisfying TC-statements.

Definition 3.2.1 (The Function f_C) *Assume $C = \mathit{Compl}(R(\bar{s}); G)$ is a table completeness statement, then function $f_C(D)$ maps database instances to R-facts*

$$f_C(D) = \{R(\bar{t}) \mid \bar{t} \in Q_{R(\bar{s}),G}(D)\}.$$

We have defined the function f_C with respect to one table completeness statement. However, we need a function for a set of table completeness statements. We introduce a new function $f_{\mathcal{C}}$

$$f_{\mathcal{C}}(D) = \bigcup_{C \in \mathcal{C}} f_C(D). \tag{3.16}$$

If D^i is an ideal database, then $f_{\mathcal{C}}(D^i)$ is the set of R-facts that must be in D^a, if (D^i, D^a) is to satisfy $\mathcal{C}$.

We continue our analogy with the previous example: if D^i_Q is the most general answer to Q representing an ideal database, then $f_{\mathcal{C}}(D^i_Q)$ is a part of the available database ensured to be complete.

We also need some properties of $f_{\mathcal{C}}(D^i_Q)$ for the proofs. We present Lemma 7 from the paper [17] with relevant properties of $f_{\mathcal{C}}(D^i_Q)$.

Lemma 3.2.2 (Lemma 7) *Let $\mathcal{C}$ be a set of TC statements. Then,*

- *$f_{\mathcal{C}}(D) \subseteq D$ for all database instances D;*
- *$(D^i, D^a) \models \mathcal{C}$ iff $f_{\mathcal{C}}(D^i) \subseteq D^a$ for all $D^a \subseteq D^i$.*

Step 3: boolean test query

In the example from the previous section, we did not introduce any boolean test queries. However, we did check that *name* had been returned as an answer tuple over both D^i and D^a. It is a binary question. We generalize it to the question: is the most general answer tuple $\bar{t}$ over D^i also an answer tuple over D^a?

Simply speaking, we encode this check into the test query. It is the reason why the test query is boolean.

As an example, the test query for the previous example is

$$\begin{aligned} Q_s() \quad \leftarrow \quad & \textit{Employee}(\textit{name}, \textit{DeptName}, \textit{Birthday}), \\ & \textit{Department}(\textit{DeptName}, \textit{Location}). \end{aligned}$$

The distinguished variable *Name* is mapped to the constant *name*. The empty tuple belongs to the answer iff the body is evaluated to true over the complete parts of the available database and the distinguished variables are matched to the general answer tuple over the ideal database.

In general, assume we are given a conjunctive query

$$Q(X_1, \ldots, X_n) \leftarrow R_1(T_1^1, \ldots, T_k^1), \ldots, R_h(T_1^h, \ldots, T_m^h). \tag{3.17}$$

Then, the boolean test query (the subscript s indicates set-semantics) is defined as

$$Q_s() \leftarrow R_1(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h(\theta T_1^h, \ldots, \theta T_m^h). \tag{3.18}$$

where θ is a mapping that enforces the distinguished variables to map to the corresponding values in the general answer to the query – D_Q^i. It substitutes by the rule:

- if $T_j^i \notin \{X_1, \ldots, X_n\}$, then $\theta :: T_j^i \mapsto T_j^i$;
- if $T_j^i \in \{X_1, \ldots, X_n\}$, then $\theta :: T_j^i \mapsto c_{T_j^i}$, where $c_{T_j^i}$ is the corresponding frozen constant from D_Q^i.

Characterization

The TC-QC characterization under set semantics shows how to check TC-QC entailment. It is meant to give an idea on how to design an encoding. We regard the characterization as a specification of the problem. If the specification works in one way, then a program must work in this way.

A characterization allows us to design a program that checks TC-QC entailment. We make an argument that the program we create has the same semantics as the characterization we have presented. That is why it is important to introduce a characterization as a specification that regulates how different parts must interact and how reasoning should work.

Having defined all necessary concepts, we present the characterization theorem. It is a computationally efficient way to check TC-QC entailment. The prototypical database D_Q^i is computed in linear time in the size of Q by applying the freezing substitution. Computation of $f_{\mathcal{C}}(D_Q^i)$ can be done by an application of TC rules. An evaluation of the test query $Q_s()$ is an evaluation of a conjunctive query. That is why, it is reasonable to work with characterizations instead of definitions directly.

Theorem 3.2.3 (Characterization under Set Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, then*

$$\mathcal{C} \models \mathit{Compl}^s(Q) \iff Q_s(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset. \tag{3.19}$$

Proof In the following θ will denote a freezing assignment of the variables in Q.

($\Rightarrow$) Let us assume $\mathcal{C} \models \mathit{Compl}^s(Q)$. We want to show that $Q_s(f_{\mathcal{C}}(D^i_Q))$ returns an empty tuple as an answer. To do so we construct a partial database $\mathcal{D} := (D^i_Q, f_{\mathcal{C}}(D^i_Q))$. According to Lemma 3.2.2, for a partial database $\mathcal{D} = (D^i, D^a)$ it holds that $\mathcal{D} \models \mathcal{C}$ iff $f_{\mathcal{C}}(D^i) \subseteq D^a$. It follows that $Q(D^i_Q) = Q(f_{\mathcal{C}}(D^i_Q))$. On the other hand, D^i_Q is a frozen version of Q where output variables $\bar{X}$ are bound to $\theta\bar{X}$. Then, the tuple $\theta\bar{X}$ is in $Q(D^i_Q)$ and the $\theta\bar{X}$ is in $Q(f_{\mathcal{C}}(D^i_Q))$. Finally, we conclude $Q_s(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset$

($\Leftarrow$) Now we assume that $Q_s(f_{\mathcal{C}}(D^i_Q))) \neq \emptyset$ and we want to show that $\mathcal{C} \models Compl^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database such that $\mathcal{D} \models \mathcal{C}$ where $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Let α be an assignment from Q into D^i that maps $\bar{X}$ to $\bar{c}$. Then, $\alpha B \subseteq D^i$ where B is the body of Q. Due to Lemma 3.2.2, $f_{\mathcal{C}}$ is a monotone function over the database instances it follows that $f_{\mathcal{C}}(\alpha B) \subseteq f_{\mathcal{C}}(D^i)$. Again from Lemma 3.2.2 we have that $f_{\mathcal{C}}(D^i) \subseteq D^a$ when $(D^i, D^a) \models \mathcal{C}$. We conclude that $f_{\mathcal{C}}(\alpha B) \subseteq D^a$.

Now let β be an assignments for which $Q_s(f_{\mathcal{C}}(D^i_Q))$ returns an empty tuple. Then, a composition $\alpha\theta^{-1}\beta$ is a proper assignment over the variables from Q into the database instance $\alpha\theta^{-1}f_{\mathcal{C}}(\theta B)$ for which Q returns an empty tuple as well. On the other hand, it is not hard to check that $\alpha\theta^{-1}f_{\mathcal{C}}(\theta B) \subseteq f_{\mathcal{C}}(\alpha B)$. Consequently, it holds that $Q_s(f_{\mathcal{C}}(\alpha B)) \neq \emptyset$.

Now $\alpha\theta^{-1}\beta$ maps $\bar{X}$ to $\bar{c}$. Then, evaluating $Q(X)$ against $f_{\mathcal{C}}(\alpha B)$ we obtain $\bar{c}$. Finally, considering that $f_{\mathcal{C}}(\alpha B) \subseteq D^a$ we conclude that $\bar{c} \in Q(D^a)$. ∎

We have introduced an effective check of TC-QC entailment. It explicitly demonstrates interaction between different parts of the problem and it can be effectively computed. It is an important step towards development of a decision procedure that can reason about query completeness. In this case, it can only perform reasoning without constraints in the schema. The above theorem will guide us in constructing an answer set program that encodes TC-QC entailment.

3.3 Encoding

To establish TC-QC entailment, we reduce the problem of TC-QC entailment into the problem of existence of an answer set (AS) of a datalog program. We construct a corresponding answer set program. It has an answer set if and only if entailment holds.

We define this program as several independent modules. The constructed program is going to mimic all the steps described in the characterization section:

1. Construct a prototypical database D^i_Q– a representation of the minimal information in a database needed to return an answer to Q.

2. Establish what facts must be in every available database satisfying TC-statements.

3. Evaluate a boolean test query $Q_s()$ over the parts ensured to be in every available database.

4. Check whether the boolean test query returns a non-empty answer, otherwise, reject the answer set.

Step 1: prototypical database in ASP

In answer set programming, there is a concept called EDB (extensional database) which is a set of ground atoms (facts) used to infer other facts. The IDB (intensional database) is the set of inferred facts.

Given a conjunctive query Q over the signature Σ^i (the same query and the same signature as in the characterization), we define the EDB of the corresponding answer set program to be syntactically equal to the prototypical database D^i_Q from 3.15.

Step 2: two signatures and $f_{\mathcal{C}}(D^i_Q)$

It is important to emphasize, there are two different signatures in the answer set programming encoding while there is only one in the characterization.

It has been introduced due to technical reasons mainly. However, it is very important to keep it in mind. When we talk about $f_{\mathcal{C}}$, we have a mathematical function. It takes as an argument a set of facts over the Σ^i signature and returns a subset of the same signature.

It is significantly different in the case of answer set programming. We cannot refer to the result of the $f_{\mathcal{C}}$ function in answer set programming without a syntactic distinction between inferred and given facts.

In the encoding, we refer to two signatures Σ^i and Σ^a (in the characterization we refer only to the signature Σ). At the beginning, we start with a prototypical database D^i_Q. It contains a set of facts over only Σ^i and an empty set over the Σ^a signature.

This distinction allows us to talk about an isomorphism between the facts over the Σ^a signature in the answer set and $f_{\mathcal{C}}(D^i_Q)$. Even though the image of $f_{\mathcal{C}}$ is a set of fact over the Σ^i, signature, we can always match facts over Σ^a in the answer set with the facts in $f_{\mathcal{C}}(D^i_Q)$. To match them, we will ignore the $\bullet^a$ index in comparison.

Let us fix the notation. Let $r^a(t_1, \ldots, t_n)$ be a fact, then we refer

- to r as its head;
- to $\bullet^a$ as its (upper) index;
- to $(t_1, \ldots, t_n)$ as its body.

We say two facts are the same except of the (upper) index, if they have the same heads and the same bodies.

Taking into account the existence of two signatures, for every table completeness statement $C = \mathit{Compl}(R(\bar{s}); G)$, we introduce a datalog rule r_C:

$$R^a(\bar{s}) \leftarrow R(\bar{s}), G. \tag{3.20}$$

Definition 3.3.1 (Encoding of TC statements) *Let $\mathcal{C}$ be a set of TC statements. Then, we define the program $P_{\mathcal{C}}$ as*

$$P_{\mathcal{C}} = \{r_C \mid C \in \mathcal{C}\}. \tag{3.21}$$

Lemma 3.3.1 *Let D_Q^i be the prototypical database. Then, for any fact $R(\bar{t})$ it holds that $R(\bar{t})$ is in $f_{\mathcal{C}}(D_Q^i)$ iff $R^a(\bar{t})$ is in the answer set of $P_{\mathcal{C}} \cup D_Q^i$, where both facts are the same except of the index.*

Proof The proof appeals to the semantic of $P_{\mathcal{C}}$ and $f_{\mathcal{C}}(D_Q^i)$. By construction, $P_{\mathcal{C}}$ is a positive datalog program. It always has a unique answer set. Shortly, we denote this answer set of $P_{\mathcal{C}} \cup D_Q^i$ as $\mathcal{A}$.

Assume, there is a fact $R^a(\bar{t})$ in $\mathcal{A}$, we will show that there is a fact $R(\bar{t})$ in $f_{\mathcal{C}}(D_Q^i)$.

There is no fact with the Σ^a signature in the EDB, therefore, according to answer set semantics, there must be an instantiated ground rule r_C supporting this fact:

$$R^a(\bar{t}) \leftarrow R(\bar{t}), G. \tag{3.22}$$

Consequently, there must be $R(\bar{t})$ and G in D_Q^i. We know that $f_{\mathcal{C}}(D_Q^i)$ is the smallest database D such that (D_Q^i, D) satisfies $\mathcal{C}$. Then, the fact $R(\bar{t})$ must be in $f_{\mathcal{C}}(D_Q^i)$. It is so, since the rule r_C forces it to be there due to the fact that atoms $R(\bar{t})$ and G are in D_Q^i. It must be there to satisfy completeness statement that correspond to the rule r_C.

Assume, there is a fact $R(\bar{t})$ in $f_{\mathcal{C}}(D_Q^i)$, we will show that there is a corresponding fact $R^a(\bar{t})$ in $\mathcal{A}$.

If a fact is in $f_{\mathcal{C}}(D_Q^i)$, then there must be a completeness statement $C = \mathit{Compl}(R(\bar{s}); G)$ such that $\bar{s}$ is unified with $\bar{t}$ of $R(\bar{t}) \in f_{\mathcal{C}}(D_Q^i)$ by α. Then, $R(\bar{t})$ and αG are in D_Q^i.

We apply α to the corresponding datalog rule r_C of the completeness statement C and obtain the rule αr_C

$$R^a(\bar{t}) \leftarrow R(\bar{t}), \alpha G. \tag{3.23}$$

The body is satisfied due to condition above. This rule forces the fact $R^a(\bar{t})$ to be in $\mathcal{A}$. ∎

As an abuse of notation we use the phrase "$P_{\mathcal{C}}$ computes the function $f_{\mathcal{C}}$". In a sense $P_{\mathcal{C}}$ forces an isomorphic set of facts over the Σ^a signature to be present in the answer set. Even though $P_{\mathcal{C}}$ does not compute anything and it is not a function, we use it to indicate that corresponding facts must be in the database to satisfy TC-statements in $\mathcal{C}$. In case of answer set programming, the only way to do it is to use a syntactic distinction between the domain (the EDB and D_Q^i) and the image (Σ^a and $f_{\mathcal{C}}(D_Q^i)$) that we have done by using the upper index $\bullet^a$.

Step 3: boolean test query

The boolean test query $Q_s()$ is defined using freezing of the distinguished variables. They are substituted by corresponding constants from D_Q^i. The query is evaluate over $f_{\mathcal{C}}(D_Q^i)$. However, in the answer set encoding we introduced the second signature Σ^a and we cannot directly evaluate the test query over $f_{\mathcal{C}}(D_Q^i)$. We make a syntactic change in the body of $Q_s()$ from the characterization chapter.

Assume, we are given a test query $Q_s()$ from Definition 3.18:

$$Q_s() \leftarrow R_1(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h(\theta T_1^h, \ldots, \theta T_m^h). \tag{3.24}$$

We define a boolean test query in the encoding by introducing the upper index $\bullet^a$ to every atom in the body of $Q_s()$:

Definition 3.3.2 (Encoding of the boolean test query)

$$Q_s \leftarrow R_1^a(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h^a(\theta T_1^h, \ldots, \theta T_m^h). \tag{3.25}$$

It allows us to evaluate the test query only over facts with Σ^a signature corresponding to $f_{\mathcal{C}}(D_Q^i)$.

Note 3.3.2 (The difference between $Q_s()$ and Q_s) *We refer to the boolean test query in the characterization as $Q_s()$ and to the corresponding test rule in the encoding as Q_s.*

Step 4: check result of the test query

We would like to make a statement: TC-QC entailment holds iff a program P has an answer set. The entailment holds if the test query returns the empty tuple as an answer. In case of answer set programming, we derive the fact Q_s instead of the empty tuple. That is why we would like to reject an answer set if it does not contain Q_s.

We introduce a filtering rule that rejects an answer set if it does not contain Q_s:

$$\bot \leftarrow \text{ not } Q_s. \tag{3.26}$$

We have introduced two types of rules in the encoding, the ones that depend on Q and the ones that depend on the set of table completeness statements $\mathcal{C}$.

We denote the first group as P_Q^s: D_Q^i, Q_s and the "filtering" rule (the upper index $\bullet^s$ indicates set semantics). The second was already defined as $P_{\mathcal{C}}$.

Let us motivate the encoding theorem in general. The encoding theorem is meant to be a translation from one formalism to another. It uses as a specification the characterization theorem and it connects this specification with a program. It shows how we can restate the problem in some other formalism that has a support of reasoning engines. Instead of developing a reasoning engine for query completeness, we reduce the initial problem into a problem for which there are effective reasoners. Then, our argument about soundness, completeness and effectiveness appeals to the characterization instead of definitions directly.

Theorem 3.3.3 (Encoding under Set Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, then*

$$\mathcal{C} \models \mathit{Compl}^s(Q) \iff P_Q^s \cup P_{\mathcal{C}} \text{ has an answer set.} \tag{3.27}$$

Proof Instead of showing the definition directly, we show the correspondence between the characterization theorem and our encoding into answer set programming. The theorem above follows from it.

Correspondence between test query and the ASP encoding

$$P = P^s_Q \cup P_{\mathcal{C}} \text{ has an answer set iff } Q_s(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset. \tag{3.28}$$

The proof is technical, that is why we briefly explain the main points of the proof. It has three logical levels. Firstly, we establish a correspondence between ground facts in the encoding and the prototypical database in the characterization. Then, we show a correspondence between the sets we derive from the prototypical database and ground facts, namely between the set of facts over the Σ^a signature and $f_{\mathcal{C}}(D^i_Q)$, by means of Lemma 3.3.1. Finally, by means of Lemma 3.25, we show that both queries evaluate to true . The proof follows these steps in both directions.

($\Rightarrow$) Assume, P has an answer set $\mathcal{A}$. Then, it has the fact Q_s in $\mathcal{A}$ due to the filtering rule. Consequently, there is a mapping β such that it maps the body B^a of Q_s to the facts in the answer set $\mathcal{A}$ over Σ^a signature.

The body B of the test query $Q_s()$ is the same set of atoms as the body B^a of the encoding test test Q_s except of the upper index $\bullet^a$, see Definition 3.25. For every atom $R^a(\bar{t})$ in the body B^a, there is an atom $R(\bar{t})$ in the body B such that they are the same except of the index. We know from the previous paragraph that $\beta R^a(\bar{t})$ is in $\mathcal{A}$, consequently $\beta R(\bar{t})$ is in $f_{\mathcal{C}}(D^i_Q)$. Due to Lemma 3.3.1, for every fact the over Σ^a signature, there is a corresponding fact in $f_{\mathcal{C}}(D^i_Q)$ the over Σ^i signature. By definition of the test query, $Q_s()$ is a boolean conjunctive query and every atom of the body B is mapped to $f_{\mathcal{C}}(D^i_Q)$ by β. Thus, the boolean test query returns the empty tuple.

($\Leftarrow$) Assume, $Q_s(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset$. There is a mapping β that maps every atom $R(\bar{t})$ in the body of $Q_s()$ to $f_{\mathcal{C}}(D^i_Q)$. Due to Definition 3.25 of the test query, for every atom $R(\bar{t})$ in the body of test query $Q_s()$, there is a corresponding atom $R^a(\bar{t})$ in the body of the test rule Q_s. Due to Lemma 3.3.1, for every atom $R(\bar{t})$ in $f_{\mathcal{C}}(D^i_Q)$, there is an atom $R^a(\bar{t})$ in $\mathcal{A}$. Consequently, for every atom $R^a(\bar{t})$ in the body of the test rule, it follows that $\beta R^a(\bar{t})$ in $\mathcal{A}$. Then, Q_s in $\mathcal{A}$ because every atom of its body mapped by β to facts over the Σ^a signature in $\mathcal{A}$. ∎

Having introduced the Encoding Theorem, we can design an effective (from a complexity point of view) reasoner for the plain TC-QC completeness problem. By design, this procedure follows the characterization that exactly captures the query completeness reasoning.

3.4 Bag Semantics

Characterization

In this section, we discuss how completeness reasoning changes, if we evaluate queries under bag semantics (i.e. if we take into account tuples together with their cardinalities).

Let us illustrate this idea by an example. Assume, we are given a relational schema with a relation $R/2$, a database $D = \{R(a, b), R(a, c)\}$ and a query $Q(X) \leftarrow R(X, Y)$.

If we evaluate this query under set semantics, the answer is $Q(D) = \{a\}$. Under bag semantics the answer is $Q^b(D) = \{\{a, a\}\}$ (we denote query evaluation under bag semantics by the upper index $\bullet^b$; double curly brackets indicate bag semantics). Answer tuple a appears twice. We keep tuples together with their cardinality.

We will revise our definition of an answer tuple in order to capture cardinalities.

Definition 3.4.1 (Answer Tuple under Bag Semantics) *Let $Q(\bar{X}) \leftarrow B$ be a query, D be a database. Then, $\bar{c}$ is an* answer tuple *with cardinality n iff there exists a set $\mathcal{A}$ of n different assignments s.t. for every α in $\mathcal{A}$ it holds that*

- $\bar{c} = \alpha\bar{X}$;
- $D, \alpha \models B$.

In the example the answer tuple a has cardinality two, because there are two mappings ($\alpha_1 = \{X \mapsto a, Y \mapsto b\}, \alpha_2 = \{X \mapsto a, Y \mapsto c\}$) satisfying the conditions in Definition 3.4.1. We also need to revise our definition of TC-QC entailment to capture evaluation under bag semantics. We propose a formal definition of TC-QC entailment under bag semantics and we also introduce a piece of syntax denoting it.

Definition 3.4.2 (TC-QC entailment under Bag Semantics) *Let Q be a query and $\mathcal{C}$ be a set of TC statements. Then, $\mathcal{C}$ entails completeness of Q under bag semantics, written $\mathcal{C} \models Compl^b(Q)$, iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \implies Q^b(D^i) = Q^b(D^a).$$

Note, that we explicitly mention bag semantics by the upper index $\bullet^b$ in the word *Compl*.

Enforcing bag semantics by freezing

It turns out that we need to make only one change in the characterization to capture evaluation under bag semantics. We need to introduce a new bag test query.
Assume, we are given a conjunctive query

$$Q(X_1, \ldots, X_n) \leftarrow R_1(T_1^1, \ldots, T_k^1), \ldots, R_h(T_1^h, \ldots, T_m^h). \tag{3.29}$$

The bag test query is defined as

$$Q_b() \leftarrow R_1(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h(\theta T_1^h, \ldots, \theta T_m^h). \tag{3.30}$$

where θ performs substitution by the rule:

- if T_j^i is a constant, then $\theta :: T_j^i \mapsto T_j^i$;
- if T_j^i is a variable, then $\theta :: T_j^i \mapsto c_{T_j^i}$, where $c_{T_j^i}$ is the corresponding frozen constant from D_Q^i.

If we freeze all variables, we require exactly the same set of variables as in D_Q^i to be derived, then all tuples will be different, due to the fact that databases are sets. As a result, we have the tuples with their cardinalities.

It forces every instantiation of the variables to be different even though there might be a projection in the query. It is exactly the definition of evaluation of the query under bag semantics.

Simply speaking, if we the body of bag test query matches to all variables in the prototypical database, we force not only answer tuples to match but also assignments to match.

In general, the difference between reasoning strategies is as follows:

- for set semantics, we show that every answer tuple that Q retrieves over D^i is also retrieved over D^a;
- for bag semantics, we show that every satisfying assignment for Q over D^i, is also a satisfying assignment for Q over D^a.

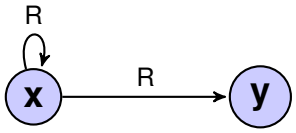
Example of freezing to enforce bag-semantics

Assume, we have a query asking for the graph pattern:

$$Q(X) \leftarrow R(X,Y),\ R(X,X). \tag{3.31}$$

where X is a distinguished variable (output variable) and Y is an existential variable. This query can be formulated as

"*All the nodes X such that there is an edge from X and X has a self-loop.*"



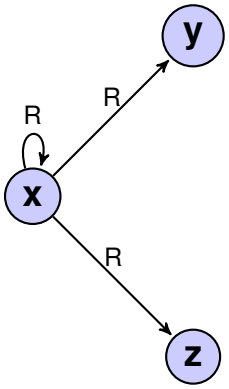
Assume, we have a TC statement:

$$R^a(X,X) \leftarrow R(X,X). \tag{3.32}$$

It can be interpreted as:

"*If there is a node X with a self-loop in the ideal database, then X is also in the available database.*"

Under set semantics Y can be easily mapped to X, because every node with a self-loop has an outgoing edge. The query is complete under set semantics. However, the situation is different under bag semantics. Consider a database:



According to the query evaluation under bag semantics, the answer tuple x has cardinality 3. Under set semantics it appears only once. The query under bag semantics is not complete with respect to the statement 3.32.

We need a TC statement to enforce counting every possible tuple satisfying the pattern differently. In this case, it can be done by introducing a statement

$$R^a(X,Y) \leftarrow R(X,X), R(X,Y). \tag{3.33}$$

It transfers necessary edges and nodes to D^a to count right cardinality by counting every tuple differently.

Characterization under bag semantics

In this section, we revise the characterization to work with bag semantics. The theorem below shows that all we need to do, is to replace the test query Q_s in Theorem 3.2.3 by the test query Q_b.

Theorem 3.4.1 (Characterization under Bag Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, then*

$$\mathcal{C} \models \mathit{Compl}^b(Q) \iff Q_b(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset. \tag{3.34}$$

Proof ($\Rightarrow$) From query completeness, it follows that Q has the same number of satisfying assignments over D^i_Q and $f_{\mathcal{C}}(D^i_Q)$. Since $f_{\mathcal{C}}(D^i_Q)$ is a subset of D^i_Q, there cannot be more satisfying assignments for Q over $f_{\mathcal{C}}(D^i_Q)$ than there are over D^i_Q. There can only be the same number of satisfying assignments, if every satisfying assignment over D^i_Q is also one over $f_{\mathcal{C}}(D^i_Q)$. Let $\bar{Y}$ be the tuple of all variables of Q. Then, α that maps $\bar{Y}$ to $\theta\bar{Y}$ is a satisfying assignment for Q over D^i_Q. Then, it is also a satisfying assignment over $f_{\mathcal{C}}(D^i_Q)$.

($\Leftarrow$) Assume, there are k different assignments α_i. Each assignment α_i maps the body B of Q to D^i and distinguished variables $\bar{X}$ to c (α_i is a general satisfying assignment, which has the property to map the distinguished variables to $\bar{c}$). Then $\alpha_i B \subseteq D^i$ where B is the body of Q. Considering that $f_{\mathcal{C}}$ is a monotone function over the database instances, it follows that $f_{\mathcal{C}}(\alpha_i B) \subseteq f_{\mathcal{C}}(D^i)$. Again from Lemma 3.2.2, we have that $f_{\mathcal{C}}(D^i) \subseteq D^a$ when $(D^i, D^a) \models \mathcal{C}$, then, we conclude that $f_{\mathcal{C}}(\alpha_i B) \subseteq D^a$.

Then the composition $\alpha_i\theta^{-1}$ is a proper assignment over the variables from Q to database instance $\alpha_i\theta^{-1} f_{\mathcal{C}}(\theta B)$ for which Q returns an empty tuple as well. On the other hand, it is not hard to check that $\alpha_i\theta^{-1} f_{\mathcal{C}}(\theta B) \subseteq f_{\mathcal{C}}(\alpha_i B)$. Consequently, we deduce that $Q_s(f_{\mathcal{C}}(\alpha_i B)) \neq \emptyset$.

Now $\alpha_i\theta^{-1}$ maps $\bar{X}$ to $\bar{c}$. Then evaluating $Q(\bar{X})$ against $f_{\mathcal{C}}(\alpha_i B)$ we obtain $\bar{c}$. Finally, considering that $f_{\mathcal{C}}(\alpha_i B) \subseteq D^a$ we conclude that $\bar{c} \in Q(D^a)$. Now we show that for any indices i, j it holds that

$$\alpha_i \neq \alpha_j \implies \alpha_i\theta^{-1} \neq \alpha_j\theta^{-1} \tag{3.35}$$

Since θ is a bijective function, its inverse function is a bijective function as well. Composition with a bijective function preserves distinctiveness of mappings. Thus, we have k different mappings $\alpha_i\theta^{-1}$ for the tuple $\bar{c}$. ∎

Encoding under bag semantics

To capture characterization under bag semantics, we have changed the test query. For the same reason, we need to change the test query in the encoding.

$$Q_b() \leftarrow R_1(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h(\theta T_1^h, \ldots, \theta T_m^h). \tag{3.36}$$

We define the encoding of the boolean bag test query $Q_b()$ by introducing the upper index $\bullet^a$ to every atom in the body:

Definition 3.4.3 (Encoding of the bag test query) *Let $Q_b()$ the bag test query from Definition 3.30, then the bag test rule is*

$$Q_b \leftarrow R_1^a(\theta T_1^1, \ldots, \theta T_k^1), \ldots, R_h^a(\theta T_1^h, \ldots, \theta T_m^h) \tag{3.37}$$

where θ is the freezing assignment from Definition 3.15.

Accordingly, we change the bag filtering rule

$$\bot \leftarrow \textit{ not } Q_b. \tag{3.38}$$

As P_Q^b we denote D_Q^i, Q_b and the bag filtering rule.

Let us motivate the encoding theorem in general. The encoding theorem is meant to be a translation from one formalism to another. It uses as a specification the characterization theorem and it connects this specification with a program. It shows how we can restate the problem in some other formalism that has a support of reasoning engines. Instead of developing a reasoning engine for query completeness from scratch, we reduce the initial problem into a problem that has efficient reasoners. Then, our argument about soundness, completeness and effectiveness appeals to the characterization instead of definitions directly. It is also important to mention that regular SQL queries are evaluated under bag semantics, that is why this theorem is important from a practical point of view.

Theorem 3.4.2 (TC-QC Encoding under Bag Semantics) *Let Q be a query and $\mathcal{C}$ be a set of TC statements. Then,*

$$\mathcal{C} \models \textit{Compl}^b(Q) \iff P_Q^b \cup P_C \textit{ has an answer set.} \tag{3.39}$$

Proof Instead of showing the definition directly, we prove a correspondence between the characterization theorem and our encoding into answer set programming. The theorem above follows from it.

Encoding and Test Query Correspondence

$$P = P_Q^b \cup P_{\mathcal{C}} \text{ has an answer set iff } Q_b(f_{\mathcal{C}}(D_Q^i)) \neq \emptyset. \tag{3.40}$$

($\Rightarrow$) Assume, P has an answer set $\mathcal{A}$. Due to Filtering Rule 3.38, the atom Q_b is in $\mathcal{A}$. All atoms of the body B^a of the test-rule Q_b must be in $\mathcal{A}$ because it is the only rule to support the atom Q_b. All atoms in B^a are over the Σ^a signature. Due to Lemma 3.3.1, for every atom $R^a(\bar{t})$

in B^a, a corresponding atom $R(\bar{t})$ is in $f_{\mathcal{C}}(D^i_Q)$. As a result, the body of the test query $Q_b()$ is satisfied and it holds that $Q_b(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset$.

($\Leftarrow$) Assume, $Q_b(f_{\mathcal{C}}(D^i_Q)) \neq \emptyset$. The body B of the test query $Q_b()$ is satisfied and it holds that $B \subseteq f_{\mathcal{C}}(D^i_Q)$. Due to Lemma 3.3.1, for every fact $R(\bar{t})$ in $f_{\mathcal{C}}(D^i_Q)$, a corresponding fact $R^a(\bar{t})$ is in the answer set $\mathcal{A}$. Consequently, for every fact $R(\bar{t})$ in the body B, a corresponding fact $R^a(\bar{t})$ is in the answer set $\mathcal{A}$. Then, the body of the test-rule Q_b is satisfied. Finally, Q_b is in $\mathcal{A}$. ∎

Having introduced the encoding theorem, we can design an efficient (from a complexity point of view) reasoner for the plain TC-QC completeness problem under bag semantics. By design, this procedure follows the characterization that exactly captures query completeness reasoning. It is an important theorem in the development of a practical completeness reasoner, since, regular SQL queries are evaluated under bag semantics.

CHAPTER 4

Reasoning with Finite Domain Constraints

Overview

In this chapter, we are going to discuss TC-QC reasoning in the presence of finite domain constrains. We start with an example showing how the reasoning procedure works with additional constraints in the schema. We generalize this example, deduce the main properties and discuss a formalization.

When we introduce the semantics of finite domain constraints in the context of completeness reasoning, we propose a characterization of TC-QC entailment in the presence of finite domain constraints. It takes into account additional possibilities to infer query completeness.

We also extend the encoding to reflect these additional inference possibilities.

4.1 Example

Let us start with an example. The schema is the same as before.

$$\mathit{Employee}(\underline{\mathit{Name}}, \mathit{DeptName}, \mathit{Birthday}). \tag{4.1}$$

$$\mathit{Department}(\underline{\mathit{DeptName}}, \mathit{Location}). \tag{4.2}$$

In addition, we impose a constraint on the *Location* column of the relation *Department*: it can be either "*Bolzano*" or "*Vienna*". Let us give an intuition about it, if for any tuple $(\mathit{name}, \mathit{location})$, the atom $\mathit{Department}(\mathit{name}, \mathit{location})$ is in the database, then *location* must be either "*Bolzano*" or "*Vienna*".

We extend our definition of a *schema* by adding a set of constraints to it. In this chapter, we consider a special type of constraints called finite domain constraints (FDC). Informally speaking, we say that a finite domain constraint consists of three components: an index i, a relation R and a finite set of constants M. This type of constraints has the name "finite domain

constraint" because it restricts the values that the i-th argument of R can take. Namely, the i-th argument of R can take only values from the set of constants M.

As in the previous chapter, we show how the reasoning procedure works in a typical case by means of an informal example.

Assume, we have a query Q

$$Q(\mathit{Name}) \leftarrow \mathit{Department}(\mathit{Name}, \mathit{Location}).$$

We also have a constraint:

"*Departments located in Bolzano and in Vienna are in the database.*"

In this statement "*Bolzano*" and "*Vienna*" are constants. We formalize this statement as two datalog rules:

$$\mathit{Department}^a(\mathit{Name}, \text{“}\mathit{Bolzano}\text{”}) \leftarrow \mathit{Department}^i(\mathit{Name}, \text{“}\mathit{Bolzano}\text{”}). \tag{4.3}$$

$$\mathit{Department}^a(\mathit{Name}, \text{“}\mathit{Vienna}\text{”}) \leftarrow \mathit{Department}^i(\mathit{Name}, \text{“}\mathit{Vienna}\text{”}). \tag{4.4}$$

If we just repeat all steps from the relational procedure, we cannot establish completeness. That is why we make use of the additional constraint.

Assume, there is a constant *name* in the answer to Q. The query is conjunctive and there must be a corresponding body atom

$$\mathit{Department}(\mathit{name}, \mathit{location}) \in D^i. \tag{4.5}$$

If we do not have an additional constraint, then none of the rules can be applied. Then, we cannot establish completeness. In this example, we also have to take into account the constraint on the *Location* in reasoning. We know *location* can be either "*Bolzano*" or "*Vienna*". Assume, it is "*Bolzano*". Then, we have

$$\mathit{Department}(\mathit{name}, \text{“}\mathit{Bolzano}\text{”}) \in D^i. \tag{4.6}$$

We apply the rule 4.3 to it and obtain the atom

$$\mathit{Department}(\mathit{name}, \text{“}\mathit{Bolzano}\text{”}) \in D^a. \tag{4.7}$$

The query Q returns *name* as the answer over both databases D^i and D^a. According to the definition 2.2.2, we infer completeness in this case.

If we assume, *location* to be "*Vienna*", we obtain the same results (due to the symmetry of the rules). For both values we can deduce completeness. We also have the constraint saying "*Vienna*" and "*Bolzano*" are the only values it can take. We can deduce completeness in general, since for any possible value of *location* we can prove completeness.

In this example, we have performed a case analysis. As we see later, this is going to be our main technique to reason with finite domain constraints.

TC-QC in the presence of finite domain constraints

Constraints on the database filter models. For example, TC statements reduce the number of models we consider. It seems natural to impose finite domain constraints as constraints on the database in the same manner as TC statements[1]

Definition 4.1.1 (TC-QC Entailment with FDCs) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{F}$ be a set of FDC. Then, $\mathcal{C}$ and $\mathcal{F}$ entail completeness of Q written $\mathcal{C}, \mathcal{F} \models Compl(Q)$ iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \text{ and } \mathcal{D} \models \mathcal{F} \implies Q(D^i) = Q(D^a). \tag{4.8}$$

It can be read as *for every partial database satisfying TC statements and finite domain constraints the query is complete*.

Note, the definition does not show interaction between the query and finite domain constraints explicitly. In extreme cases, we can even decide query completeness based only on the query and the set of FDC statements. It happens because now the query might get unsatisfiable with respect to the set of FDC. We are going to see an example later.

4.2 Properties

In this section, we are going to investigate properties of finite domain constraints. We exploit them to design a decision procedure working with finite domain constraints in the schema.

We start with an example indicating what features need to be reflected in the formalization. By means of examples, we show important properties of finite domain constraints in the context of completeness reasoning. We also argue that the formalization must reflect these properties and make use of them.

Interaction between query and FDC

Let us discuss general issues of completeness reasoning with additional constraints. We are going to point out how they affect reasoning and what are the consequences of imposing them.

To illustrate this idea, we use an example. Consider a query

$$Q() \leftarrow A(X),\ B(X),\ C(Y). \tag{4.9}$$

Assume, there is no TC statement but there are two FDC constraints[2]

$$F_1 = Dom(A, 1, \{a\}). \tag{4.10}$$
$$F_2 = Dom(B, 1, \{b\}). \tag{4.11}$$

There are infinitely many databases satisfying these constraints; however, the query is unsatisfiable with respect to the constraints. As a result, we can immediately conclude its completeness. If we do not take into account the query, we cannot infer completeness.

[1] for FDC satisfaction, see 2.14.

[2] for semantics and syntax of FDC, see Section 2.4.

Let us have a closer look at these finite domain constraints and the query. There are two constraints saying that the argument of the relation A can be only a and of the relation B only b. Informally speaking, we see that the variable X in the query is bound by F_1 because it constrains the same position of A where X occurs. For the same reason, F_2 constrains X. On the one hand, X can take only the value a; on the other hand, it can take only the value b. There is no assignment to the variables in the query that can possible satisfy these constraints.

The definition does not indicate explicitly interaction between the query and FDC. That is why we are going to design an effective check – a characterization that takes into account this interaction and exploit properties of the query and constraints.

Independence of bound variables

In this subsection, we investigate one of the most important properties of FDC – independence of bound variables. Informally speaking, values of a bound variable do not depend on the values of other variables.

Assume, the query is the same

$$Q() \leftarrow A(X),\ B(X),\ C(Y). \tag{4.12}$$

There is a finite domain constraint

$$F_C = \mathit{Dom}(C, 1, \{c_1, c_2\}). \tag{4.13}$$

It allows us to rewrite the query to

$$Q() \leftarrow A(X),\ B(X),\ (C(c_1) \vee C(c_2)). \tag{4.14}$$

This is syntactic sugar for the union of two queries

$$Q() \leftarrow A(X),\ B(X),\ C(c_1). \tag{4.15}$$
$$Q() \leftarrow A(X),\ B(X),\ C(c_2). \tag{4.16}$$

In this case, we are interested in two concrete queries instead of one general.

Assume, there is also a finite domain constraint

$$F_A = \mathit{Dom}(A, 1, \{a_1, a_2\}). \tag{4.17}$$

We rewrite the query again

$$Q() \leftarrow A(a_1),\ B(a_1),\ C(c_1). \tag{4.18}$$
$$Q() \leftarrow A(a_1),\ B(a_1),\ C(c_2). \tag{4.19}$$
$$Q() \leftarrow A(a_2),\ B(a_2),\ C(c_1). \tag{4.20}$$
$$Q() \leftarrow A(a_2),\ B(a_2),\ C(c_2). \tag{4.21}$$

There are four cases – ways to map bound variables. In general, we say that a case is a mapping of bound variables in the query to a set of constants. In this particular example, the cases are

1. $X \mapsto a_1;\ Y \mapsto c_1;$

2. $X \mapsto a_1;\ Y \mapsto c_2;$

3. $X \mapsto a_2;\ Y \mapsto c_1;$

4. $X \mapsto a_2;\ Y \mapsto c_2.$

If we recall our definition of a finite domain constraint, we see that every assignment of a variable is independent from the other variables. If a variable is bound by a finite domain constraint, then its value does not depend on the other variables in the query. There are two cases for X: a_1 and a_2; there are two cases for Y: c_1 and c_2. We have to take into account all possible combinations of cases.

It gives us an intuition about case analysis. If values of bound variables are independent, then we can order variables and values. It allows us to make a stepwise case analysis. We can iterate over all assignments to investigate each case separately.

Interaction of finite domain constraints

From the previous section, we know that bound variables are independent. But the very first example shows that finite domain constraints can interact over a bound variable even if they are imposed over different relations.

Assume, the query is the same

$$Q() \leftarrow A(X),\ B(X),\ C(Y). \tag{4.22}$$

Suppose, there are two FDC

$$F_A = \mathit{Dom}(A, 1, \{a, b, c\}). \tag{4.23}$$
$$F_B = \mathit{Dom}(B, 1, \{b, c, d\}). \tag{4.24}$$

They both bind the same variable X in the query, even though they are declared over different relations.

What values can X take? Assume, the value of the variable X is x. Then, x must satisfy both F_A and F_B

$$x \in \{a, b, c\} \text{ and } x \in \{b, c, d\}. \tag{4.25}$$

We simplify it to

$$x \in \{a, b, c\} \cap \{b, c, d\}. \tag{4.26}$$

This example can be generalized to an arbitrary number of finite domain constraints.

We have obtained a set of possible values of the variable X. For a variable X, we call the set of possible values M_X. If there is a bound variable X and a set of FDC $\mathcal{F}_X$ constraining it, the set of possible values M_X is intersection of all domains of $\mathcal{F}_X$.

Overall number of cases

The set of possible values of a variable X_1 do not depend on the values of X_2. Assume, M_{X_1} is the set of possible values of X_1 and M_{X_2} is the set of possible values of X_2. If we pick any value x_1 from M_{X_1} for X_1, then we still can pick any value from M_{X_2}. There are $|M_{X_1}|$ options to pick a value for X_1, for each of which there are M_{X_2} options. Then, the total number is

$$|M_{X_1}| * |M_{X_2}|. \tag{4.27}$$

By induction, this approach can be generalized to any number of bound variables. However, what is missing in this reasoning is a way to compute M_{X_1}. To compute M_{X_1}, we use our second statement about interaction of finite domain constraints.

We define a set of possible values as a domain intersection of finite domain constraints constraining it. Assume, finite domain constraints f_1 and f_2 constrain X_1; f_2 and f_3 constrain X_2, then the total number of cases is

$$|M_{f_1} \cap M_{f_2}| * |M_{f_2} \cap M_{f_3}|. \tag{4.28}$$

Where M_{f_i} is the set of constants of finite domain constraints f_i.

Summary

In this section, by means of examples, we have indicated three main properties of finite domain constraints:

1. Independence of bound variables.
2. Interaction of finite domain constraints.
3. The overall number of cases.

The first statement shows how to reduce the case analysis over the whole query to the case analysis over one bound variable. The second statement shows how to use a set of finite domain constraints to perform a cases analysis over one bound variable. The third statement shows how the combination of the query and the set of finite domain constraints affects the number of cases we have to consider.

4.3 Formalization

To formalize the completeness reasoning problem in the presence of finite domain constraints, we generalize the examples from the previous section. Having established the main properties of finite domain constraints, we make use of them by introducing a characterization working with finite domain constraints in the schema. We have already indicated the main ideas and properties, now we are going to formalize them rigorously.

We are going to revise the examples step by step and generalize them.

Independence of bound variables

So far we have just introduced an informal definition of a bound variable. In this subsection, we fix the notation of it.

Definition 4.3.1 (Bound Variable) *In a query Q, a variable v is a bound by a finite domain constraint $Dom(R, i, M)$, if v occurs in the i-th position of a body atom with the relation R in Q.*

In the example 4.9, the variable X is bound by both FDC statements 4.10, 4.11 and Y is not bound by any FDC.

Note 4.3.1 *Throughout the section, we assume the query to be the same for all definitions. We refer to assignments and bound variables defined with respect to a query. We assume, the query is the same for all of them.*

Assume, there is an assignment α consistent with the set of FDC $\mathcal{F}$. Assume, there are at least two variables X and Y bound by some finite domain constraint and they can take at least two values $\{a, b\}$.

Assume, α maps X to a and Y to b. Now we define an assignment β as $\alpha[X \mapsto b]$. That is, the assignment β coincides with α everywhere but on X and it is consistent with finite domain constraints; furthermore, we can define β' consistent with α and change Y to a. Consequently, for any assignment we can independently switch the value of a bound variable and obtain an assignment consistent with FDC.

Finally, we conclude an independence statement about finite domain constraints.

Proposition 4.3.2 (Independence of finite domain contraints) *Let $\mathcal{F}$ be a set of finite domain constraints, B be a condition, and α be an assignment for the variables in B. If α is consistent with $\mathcal{F}$, then also $\alpha[X/c]$ is consistent with $\mathcal{F}$ for any $c \in M_X$.*

Interaction of finite domain constraints

In this subsection, we generalize the corresponding examples from the previous chapter. Let $\mathcal{F}_X$ be the set of FDC constraining a variable X. Assume, the variable X has a value x consistent with the set $\mathcal{F}_X$. Then for any $Dom(R, i, M) \in \mathcal{F}_X$ it holds that

$$x \in M. \tag{4.29}$$

In other words, x must be in every domain of $\mathcal{F}_X$. By a contrapositive argument, it must take values only from the intersection of all domains in $\mathcal{F}_X$.

Definition 4.3.2 (Set of Possible Values) *Let X be a variable, $\mathcal{F}_X$ be a set of FDC constraining X, then we define a set of possible values written M_X as*

$$M_X = \bigcap_{Dom(R,i,M) \in \mathcal{F}_X} M. \tag{4.30}$$

Case analysis

In this subsection, we are going to formalize the definition of a case. Let Q be a query, $\mathcal{F}$ be a set of finite domain constraints, then *a case* is a mapping γ such that

- if a variable X is bound, then it γ maps X to a value in M_X;
- otherwise, γ maps X to itself.

As we have seen in the previous chapter, there is an exact number of cases. We denote the set of all cases as $\Gamma_{\mathcal{F},\mathcal{Q}}$.

Correspondence between assignments and cases

In this subsection, we are going to investigate properties of cases with respect to general assignments. We also exploit these properties to establish correspondence between general assignments and cases. A case maps only bound variables. The sets of bound and non-bound variables are trivially disjoint. Then, any assignment α can be represented as

$$\alpha = \beta\gamma. \tag{4.31}$$

Where γ maps only bound variables and β maps only non-bound variables. Note, that in this case α and β are commutative.

We propose a stronger statement showing how assignments and cases are related.

Lemma 4.3.3 (Representation Lemma) *Let α be an assignment consistent with $\mathcal{F}$, then α can be represented as*

$$\alpha = \alpha'\gamma. \tag{4.32}$$

where α' is an assignment (might be general) and $\gamma \in \Gamma_{\mathcal{F},\mathcal{Q}}$.

Proof We have to prove two properties that α can be represented by two assignments α' and γ and that γ is in $\Gamma_{\mathcal{F},\mathcal{Q}}$.

First statement trivially follows from the argument above (in case if α' is a general mapping, α' and γ are not commutative).

Second statement follows from a contrapositive argument. Assume, $\gamma \notin \Gamma_{\mathcal{F},\mathcal{Q}}$. Then, $\gamma \neq \gamma_i$, for all $\gamma_i \in \Gamma_{\mathcal{F},\mathcal{Q}}$. Then, for any $\gamma_i \in \Gamma_{\mathcal{F},\mathcal{Q}}$, γ must disagree with γ_i at least on one bound variable. Assume, it is X. Then, γ maps it to a constant x such that $x \notin M_X$. Consequently, γ violates $\mathcal{F}$. Contradiction. ∎

Interaction between the query and cases

Assume, a query Q with distinguished variables $\bar{X}$ returned $\bar{t}$ as an answer over D^i. We know that $\Gamma_{\mathcal{F},\mathcal{Q}}$ contains all consistent with $\mathcal{F}$ mappings of bound variables. Then, due to the representation lemma $\bar{t}$ must be of the form

$$\bar{t} = \theta\gamma\bar{X}, \tag{4.33}$$

where θ is the freezing assignment from 3.18. Instead of a general tuple with frozen constants, we have a general tuple where all bound variables are mapped by γ. The general tuple $\bar{t}$ has this form since it must be consistent with finite domain constraints and all the other variable must be frozen. It is always safe to map bound variables and then to apply freezing since all bound variables have already turned into constants and they are not affected by θ. Then, for a case γ the general answer to a query Q is $\theta\gamma\bar{X}$. Variables that are bound and distinguished must be mapped by the case γ. This connection between the current case and distinguished variables must be reflected in the formalization.

In the definition of a test query 3.18 we mapped all distinguished variables of the test query to the corresponding frozen constants by means of θ. We have to change this mapping to capture the new general answer tuple.

Definition 4.3.3 (Test Query $Q_s^\gamma()$) *Let Q be a query from the definition 3.18, γ be a case in $\Gamma_{\mathcal{F},\mathcal{Q}}$, then the gamma test query $Q_s^\gamma()$ is*

$$Q_s^\gamma() \leftarrow R_1(\theta' T^1, \ldots, \theta' T_k^1), \ldots, R_h(\theta' T_1^h, \ldots, \theta' T_m^h). \tag{4.34}$$

where θ' performs substitution by the rule:

- *if $T_j^i \in \bar{X}$ and T_j^i is a bound variable, then $\theta' T_j^i \mapsto \gamma T_j^i$;*
- *else if $T_j^i \in \bar{X}$, then $\theta' T_j^i \mapsto c_{T_j^i}$, where $c_{T_j^i}$ is the corresponding frozen constant from D_Q^i;*
- *otherwise, $\theta' T_j^i \mapsto T_j^i$.*

Intuition behind it is that the query evaluates to true, if distinguished bound variables can be mapped to the corresponding constants in the case.

Interaction between the prototypical database and cases

Cases force bound variables to take particular values from domains of finite domain constraints instead of taking arbitrary values. As in the example from the previous section, the prototypical database changes because of the general answer to the query changes. This interaction between cases and the prototypical database must be taken into account in the formalization. We need to substitute general frozen values with particular constants from the considered case.

Let us give an intuition about this. We substitute the bound variables in the prototypical database by applying the corresponding γ. Then, we define the prototypical database with respect to a particular case as follows.

Definition 4.3.4 (Prototypical Database γD_Q^i) *Let Q be a query with a body B and γ be a mapping of bound variables. Then, the prototypical database γD_Q^i is*

$$\gamma D_Q^i = \theta\gamma B. \tag{4.35}$$

where θ is freezing assignment from 3.15.

4.4 Characterization

In this section, we present an efficient check of TC-QC entailment. It explicitly shows and uses the interaction between all parts of the problem. It also indicates how the original problem is affected by the introduction of finite domain constraints. Simply speaking, it takes into account how many cases it needs to consider and how a particular case changes the reasoning problem.

It is also an important step in the development of a practical decision procedure that can be programmed and used as a way to automatically check whether TC-QC entailment holds. Formally, we present it in the following theorem.

Theorem 4.4.1 (TC-QC Characterization with FDCs) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{F}$ be a set of FDC. Then,*

$$\mathcal{C}, \mathcal{F} \models \mathit{Compl}^s(Q) \iff Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i)) \neq \emptyset \quad \textit{for every } \gamma \textit{ in } \Gamma_{\mathcal{F},\mathcal{Q}}. \tag{4.36}$$

Proof In the following θ will denote a freezing assignment of the variables in Q.

($\Rightarrow$) Let's assume $\mathcal{C}, \mathcal{F} \models \mathit{Compl}^s(Q)$. We want to show that $Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i))$ evaluates to true (it returns the empty tuple). To do so we construct a partial database $\mathcal{D} := (\gamma D_Q^i, f_\mathcal{C}(\gamma D_Q^i))$, for an arbitrary but fixed γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. According to Lemma 3.2.2, for a partial database $\mathcal{D} = (D^i, D^a)$ it holds that $D \models \mathcal{C}$ iff $f_\mathcal{C}(D^i) \subseteq D^a$. Thus, by construction the partial database $\mathcal{D}$ satisfies $\mathcal{C}$. It follows that $Q(\gamma D_Q^i) = Q(f_\mathcal{C}(\gamma D_Q^i))$. On the other hand, γD_Q^i is a frozen version of Q where output variables $\bar{X}$ are bound to $\theta\gamma\bar{X}$. Then $\theta\gamma\bar{X} \in Q(\gamma D_Q^i)$ and so $\theta\gamma\bar{X} \in Q(f_\mathcal{C}(\gamma D_Q^i))$. Finally we conclude $Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i)) \neq \emptyset$.

($\Leftarrow$) Now we assume that $Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i))) \neq \emptyset$ and we want to show that $\mathcal{C}, \mathcal{F} \models \mathit{Compl}^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database such that $\mathcal{D} \models \mathcal{C}$ where $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Let α be an assignment from Q into D^i that maps $\bar{X}$ to $\bar{c}$. Due the assumption $\mathcal{D} \models \mathcal{F}$, we apply Representation Lemma 4.3.3 to α. Then, $\alpha = \alpha'\gamma$ for some γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Then $\alpha'\gamma B \subseteq D^i$ where B is the body of Q. Considering that $f_\mathcal{C}$ is a monotone function (due to Lemma 3.2.2) over database instances it follows that $f_\mathcal{C}(\alpha'\gamma B) \subseteq f_\mathcal{C}(D^i)$. Again from Lemma 3.2.2 we have that $f_\mathcal{C}(D^i) \subseteq D^a$ when $(D^i, D^a) \models \mathcal{C}$, then we conclude $f_\mathcal{C}(\alpha'\gamma B) \subseteq D^a$.

Now let β be an assignment such that $Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i))$ evaluates to true for every γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Then the composition $\alpha'\gamma\theta^{-1}\beta$ is a proper assignment over the variables from Q to the database instance $\alpha'\gamma\theta^{-1} f_\mathcal{C}(\gamma D_Q^i)$ for which Q evaluates to true as well. On the other hand, it is not hard to check that $\alpha'\gamma\theta^{-1}\beta f_\mathcal{C}(\gamma D_Q^i) \subseteq f_\mathcal{C}(\alpha'\gamma B)$. Consequently $Q_s^\gamma(f_\mathcal{C}(\gamma D_Q^i)) \neq \emptyset$.

Now $\alpha'\gamma\theta^{-1}\beta$ maps $\bar{X}$ to $\bar{c}$. Then evaluating $Q(X)$ against $f_\mathcal{C}(\alpha'\gamma B)$ we obtain $\bar{c}$. Finally, considering that $f_\mathcal{C}(\alpha'\gamma B) \subseteq D^a$ we conclude that $\bar{c} \in Q(D^a)$. ∎

We have presented the characterization theorem that we are going to use as a mathematical formulation of what should be done by a program. It is essentially a specification of the problem. Later, we are going to develop a procedure that will follow this specification. As a result, we will obtain a decision procedure. We appeal to the characterization to show soundness and completeness of the decision procedure.

4.5 Encoding

In this section, we are going to discuss how the encoding changes in the presence of finite domain constraints. In the preceding chapter, we showed the correspondence between the relational characterization and the encoding. In this section, we are going to follow the same idea. We have already seen how the characterization changes. It indicates how to start modifications of the encoding.

In the TC-QC characterization with finite domain constraints, we changed the prototypical database and the test query by applying an assignment to them. In general, the encoding stepwise mirrors reasoning in the characterization. That is why we start looking at these two parts and modify them accordingly.

The section has the following structure:

- we show how to encode the cases γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$;
- we revise Q_s, D^i_Q in P_Q;
- we analyze how $P_\mathcal{C}$ changes in the presence of new rules.

Encoding of cases

We defined a case γ as an assignment of bound variables. An assignment is a set of pairs. It is reasonable to encode a set of pairs as a binary predicate. We call it *val*, which stands for *value*. The set of *val*-atoms in the answer set corresponds to γ. A fact $\mathit{val}(x, a)$ is in the answer set iff γ maps the variable X to the constant a. We formalize this property later as a lemma.

First of all, we have to impose on *val* a functionality constraint. Any assignment must satisfy the functionality constraint. Then, the assignment γ must satisfy the functionality constraint. The predicate *val* represents γ and must satisfy the functionality constraint as well. As a first attempt, we try to define the functionality constraint as

$$\bot \leftarrow \mathit{val}(X, Y),\ \mathit{val}(X, Z),\ Y \neq Z. \tag{4.37}$$

However, we do not know in advance what variables are going to be bound in the query and we would like to make the generation of γ independent from the other parts of the program. Therefore, for every frozen variable x in the query Q, we introduce a default value $\mathit{val}(x, x)$. It immediately violates Functionality Constraint 4.37.

Instead of Functionality Constraint 4.37, we define a weak functionality constraint r_{fun} as

$$\bot \leftarrow \mathit{val}(X, Y),\ \mathit{val}(X, Z),\ X \neq Y,\ X \neq Z,\ Y \neq Z. \tag{4.38}$$

That allows to have default values of the frozen variables and the values from γ.

To introduce $\Gamma_{\mathcal{F},\mathcal{Q}}$ in the encoding, we mimic the steps we took in the analysis of finite domain constraints. Let $f = \mathit{Dom}(R, i, M)$ be a finite domain constraint, then we define the rule r_f as

$$\mathit{val}(X_i, a_1) \vee \cdots \vee \mathit{val}(X_i, a_n) \leftarrow R(X_1, \ldots, X_i, \ldots, X_n). \tag{4.39}$$

where each a_i is in M.

We define a program $P_{\mathcal{F}}$ as

$$P_{\mathcal{F}} = \{r_f \mid f \in \mathcal{F}\} \cup \{r_{\mathit{fun}}\}. \tag{4.40}$$

Encoding of the prototypical database γD^i_Q

In the encoding of γD^i_Q, we assume that a case γ is given as a set of *val*-facts. Assigning values to bound variables in γD^i_Q is done implicitly by presence of γ in the answer set as *val*-facts. That is why we only add to D^i_Q a set of default values.

Let V be a set of all variables in Q and D^i_Q be a prototypical database. Then, D^u_Q is defined as

$$D^u_Q = D^i_Q \cup \{\mathit{val}(\theta v, \theta v) \mid v \in V\}. \tag{4.41}$$

Encoding of variable unfolding

In the encoding of Q^γ_s, we need to take into account two facts

1. *values* of the distinguished variables must be mapped to *values* of the corresponding "frozen" constants in γD^i_Q;

2. *values* of the other variables must be mapped to some *values*.

The first statement follows from 4.33, we know that all bound variables must be mapped according to γ. Then, if a distinguished variable is not bound by a finite domain constraint, then its value is the same as its name. If it is bound, then it must be mapped to the value in γ. Observe that for a predicate A, a variable X and a constant c, it holds that

$$A(c) \iff A(X) \wedge X = c. \tag{4.42}$$

In our encoding, the role of "$=$" is played by the *val*-predicate. The first argument of *val* is a variable and the second is its value. Then in our context, we encode this statement as a formula.

$$A(c) \iff A(X), \mathit{val}(X, c). \tag{4.43}$$

If we assign a value to a variable X, in a predicate $A(X)$, we introduce a "fresh" variable V_X instead of the constant c

$$A(X) \iff A(V_X), \mathit{val}(V_X, X). \tag{4.44}$$

An atom $A(X)$ holds if there is an atom $A(V_X)$ in D^u_Q and V_X has the value of X. In case, if X is not a bound variable, then it has the default value of itself. Then, $A(X)$ holds iff $A(X) \in D^u_Q$ ($\mathit{val}(X, X)$ holds always due to 4.41).

Rule 4.44 suffices for non-distinguished variables. For distinguished variables we have to extend it. Assume, we have an atom $A(x)$, where x is a "frozen" variable. The atom $A(x)$ with a value v of x holds if the atom $A(v)$ holds. Applying Rule 4.42, we obtain $A(v)$ holds iff there is an atom $A(y)$ and val(y, v).

Summing up, we say an atom $A(x)$ with $x = v$ holds iff there exists a constant y such that the atom $A(y)$ in D^u_Q and $y = v$. We encode this statement as

$$A(x) \iff A(Y), \mathit{val}(x, V), \mathit{val}(Y, V). \tag{4.45}$$

where variables Y and V are fresh. The right hand side of 4.45 is satisfied if Y is bound to an argument of A, such that the value of this argument is the same as the value of x.

Encoding of the test query Q^γ_s

In this subsection, we are going to apply the unfolding rules to the test query. As a result, we obtain the unfolded test rule that takes into account values of frozen variables. In the previous chapter, the test rule has the form of $Q(\bar{X}) \leftarrow B$, where B is a conjunction of atoms. Now we have to into account values of variables. We do it by means of unfolding of variables. Then, the unfolded test rule has the form of $Q(\bar{X}) \leftarrow B, U_Q$ where U_Q is a conjunction of *val*-atoms.

Here is the algorithm that unfolds the test rule. Assume, the *val*-set U_Q is empty. Let Q_s be a test rule with body B, γ be a set of *val*-atoms. Then, we unfold the body of the test rule, by the following rule: for every term T_i in every atom $R(T_1, \ldots, T_n) \in B$

- if T_i is a distinguished frozen variable, replace T_i with Y; add $\mathit{val}(Y, V), \mathit{val}(T_i, V)$ to the *val*-set U_Q, according to 4.45.
- otherwise, replace T_i with V; add $\mathit{val}(V, T_i)$ to the *val*-set U_Q, according to 4.44.

where V and Y are fresh variables. We denote the unfolded version of B as B^u. Note, for different occurrences of T_i, we introduce different variables V.

Definition 4.5.1 *Let Q_s be a test rule, γ be a set of val-atoms. Then, we define Q^u is*

$$Q^u \leftarrow B^u, U_Q. \tag{4.46}$$

where B^u is the unfolded body of Q_s and U_Q is the val-set from the unfolding algorithm.

Let us illustrate this definition with an example. Assume, a query Q is

$$Q(\mathit{Name}) \leftarrow \mathit{Department}(\mathit{Name}, \mathit{Location}). \tag{4.47}$$

Then, Q^u is

$$Q^u \leftarrow \mathit{Department}^a(V, V_{\mathit{Location}}), \mathit{val}(V, V_{\mathit{Name}}), \mathit{val}(\mathit{name}, V_{\mathit{Name}}), \mathit{val}(V_{\mathit{Location}}, \mathit{Location}).$$

This rule fires over a *Department*-atom in the available database such that its first argument has the same value with a frozen variable *name* and its second argument can take any value.

We define the program P^u_Q.

$$P^u_Q = \{Q^u\} \cup D^u_Q. \tag{4.48}$$

where we identify Q^u with the test rule defining Q^u.

Revision of the $P_{\mathcal{C}}$ encoding

We encode γ as a set *val*-atom, that is why we need to unfold variables in the encoding of TC statements to take into account their *values* of the frozen variables. We have already seen a way to do it 4.44. We need to let the variables pass through TC statements judging by their values.

Assume, we have a rule r_C that encodes a TC statement C

$$R^a(\bar{t}) \leftarrow R(\bar{t}), G. \tag{4.49}$$

For every atom $A(T_1, \ldots, T_n)$ in the body of r_C, we apply Rule 4.44. As a result, we obtain an unfolded body of the rule r_C: $R^u(\bar{t}'), G^u, U_C$. We define the unfolded rule r_C^u as

$$R^a(\bar{t}') \leftarrow R^u(\bar{t}'), G^u, U_C. \tag{4.50}$$

We define the encoding of the TC rules $P_{\mathcal{C}}^u$ as

$$P_{\mathcal{C}}^u = \{r_C^u \mid C \in \mathcal{C}\}. \tag{4.51}$$

Let us illustrate this definition with an example. Assume we have a TC statement r_C

$$\textit{Department}^a(\textit{Name}, \textit{Location}) \leftarrow \textit{Department}^i(\textit{Name}, \textit{Location}). \tag{4.52}$$

Then, the unfolded rule r_C^u is

$$\begin{aligned} \textit{Department}^a(V_{\textit{Name}}, V_{\textit{Location}}) \leftarrow \quad & \textit{Department}^i(V_{\textit{Name}}, V_{\textit{Location}}), \\ & \textit{val}(V_{\textit{Name}}, \textit{Name}), \textit{val}(V_{\textit{Location}}, \textit{Location}). \end{aligned}$$

Unfolding a query makes more sense, if we have a join variable; however, it tends to be hard to read. It illustrates that the most important feature of unfolding is that we judge every variable by its value stored in the *val*-predicate. We delegate the assignment of variables to the external predicate *val* in an answer set. We let the names of the variables go through TC statements since we store all the values separately.

Properties of the encoding

We have established how the introduction of finite domain constraints has changed the definitions. Here we are going to investigate properties of parts of the encoding formally. This subsection has the following structure:

1. correspondence between a case γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$ and the *val*-set in the encoding;
2. correspondence between atoms in the characterization and atoms in the encoding;
3. correspondence between the test query and the test rule.

We start the discussion with a correspondence between γ and the *val*-set. They both represent a case – an assignment of bound variables consistent with finite domain constraints. We would like to make a connection between the atoms in the *val*-set of the answer set and γ from $\Gamma_{\mathcal{F},\mathcal{Q}}$. We introduce a piece of syntax to make the statement more readable. Given a case γ, we denote the encoding of γ as a *val*-set as $\textit{val}(\gamma)$. To formally introduce it we present a lemma.

Lemma 4.5.1 *Let Q be a query and γ be a case in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Let P be the program $P_{\mathcal{F}} \cup P^u_{\mathcal{C}} \cup P^u_Q$. Then,*

- *for every γ, there is exactly one answer set $\mathcal{A}$ of P such that $val(\gamma) \subset \mathcal{A}$;*
- *for every answer set $\mathcal{A}$ of P, there is exactly one γ such that $val(\gamma) \subset \mathcal{A}$.*

Proof We prove both of the statements by contrapositive argument.

Assume, for some γ there are two answer sets $\mathcal{A}_1$ and $\mathcal{A}_2$. The only disjunctive rules are in $P_{\mathcal{F}}$. Then, $\mathcal{A}_1$ and $\mathcal{A}_2$ differ only in *val*-atoms. Due to minimality there are only *val*-atoms for bound variables. Then, for some bound variable x, there must be two atoms $val(x, a) \in \mathcal{A}_1$ and $val(x, b) \in \mathcal{A}_2$. Due to the functionality constraint, x has the only one value in an answer set. Then, we obtain contradiction since to satisfy both inclusions γ must violate the functionality constraint. Contradiction.

Assume, for some answer set $\mathcal{A}$ there are two gammas γ_1 and γ_2. They differ at least in one variable; assume it is X. Assume, γ_1 maps it to a and γ_2 maps it to b. By initial assumption, $val(\gamma_1) \subset \mathcal{A}$ and $val(\gamma_2) \subset \mathcal{A}$. Then, $\mathcal{A}$ contains $val(x, a)$ and $val(x, b)$. It violates the functionality constraint. Contradiction. ∎

Let us illustrate this lemma by an example. Assume, there is a query Q

$$Q(\mathit{Name}) \leftarrow \mathit{Department}(\mathit{Name}, \mathit{Location}). \tag{4.53}$$

Assume, there is a finite domain constraint

$$val(\mathit{Name}, \text{“}\mathit{Bolzano}\text{”}) \vee val(\mathit{Name}, \text{“}\mathit{Vienna}\text{”}) \leftarrow \mathit{Department}(\mathit{Name}, \mathit{Location}). \tag{4.54}$$

There are two cases. When a case γ_1 maps *Name* to “*Bolzano*” and when a case γ_2 maps *Name* to “*Vienna*”. Due to Finite Domain Constraint 4.54, there are two answer sets: an answer set $\mathcal{A}_1$ that contains the atom $val(\mathit{Name}, \text{“}\mathit{Bolzano}\text{”})$ and an answer set $\mathcal{A}_2$ that contains the atom $val(\mathit{Name}, \text{“}\mathit{Vienna}\text{”})$. Then, $\mathcal{A}_1$ corresponds to γ_1 and $\mathcal{A}_2$ corresponds to γ_2.

Secondly, we are going to make a formal connection between the modified test query Q^γ_s and the test rule in the encoding Q^u. Let D be a database over the Σ^i signature, then we denote the same set of facts as in D but with the upper index $\bullet^a$ as $(D)^a$.

Lemma 4.5.2 *Let $Q^\gamma_s()$ be the test query, γD^i_Q be the prototypical database and Q^u be the test rule corresponding to the test query. Then, the following are equivalent*

- *$Q^\gamma_s(\gamma D^i_Q) \neq \emptyset$;*
- *the rule Q^u is activated over the set of facts $(D^u_Q)^a \cup val(\gamma)$.*

Proof The test query holds iff the body is satisfied. The rule is activated iff the body is satisfied. Then, we need to show that both bodies are satisfied over the corresponding sets.

Assume, the test query $Q^\gamma_s()$ is satisfied. Then, there is a mapping α that maps B to D such that distinguished variables are mapped by γ. Due to Lemma 4.5.1, γ is encoded in the *val*-set. For any distinguished variable in a body atom $R^a(\bar{t})$ in Q^u, we apply Rule 4.45. A

non-distinguished variable X α maps to a constant c. Assume an atom $A(c)$ holds, then atom $A(x)$ is in γD^i_Q and due to Lemma 4.5.1, it holds that $val(x, c) \in val(\gamma)$. Then, body of the rule Q^u is satisfied.

Assume, the rule Q^u is activated. Then, due to Lemma 4.5.1, $val(\gamma)$ encodes γ. Due to Rule 4.45, all distinguished variables have values assigned by γ. Due to Rule 4.44, all other variables have some values, they are assigned by some α. Then, we construct the assignment $\alpha\gamma$ such that, it satisfies body of the test query $Q^\gamma_s()$. ∎

Note 4.5.3 *Lemma 4.5.2 shows, that it is possible to separate the binding information from the core representation of the rules and the query.*

Finite domain characterization and encoding correspondence

To automate the work of the decision procedure, we reduce the characterization to evaluation of an answer set program. It allows us to check TC-QC entailment by using existing ASP reasoning engines. Note, this solution is optimal from a complexity point of view. The complexity of the reduced problem is the same as the complexity of the problem we reduce into.

We would like to formally establish a correspondence between the characterization we have presented in this chapter and answer set programming.

Theorem 4.5.4 (Encoding with FDCs) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{F}$ be a set of FDC. Then,*

$$\mathcal{C}, \mathcal{F} \models Compl(Q) \iff Q \text{ is in every answer set of } P^u_Q \cup P^u_{\mathcal{C}} \cup P_{\mathcal{F}}. \tag{4.55}$$

Proof The proof is technical, that is why we briefly explain the main points of the proof. The proof appeals to the characterization presented before. It has four logical levels. Firstly, we establish a correspondence between ground facts in the encoding and the prototypical database in the characterization. Secondly, by means of Lemma 4.5.1, we show a correspondence between a considered case γ and the *val*-set in the encoding. Then, we show a correspondence between the sets we derive from the prototypical database and ground facts, namely between the set of facts over the Σ^a signature and $f_{\mathcal{C}}(D^i_Q)$, by means of Lemma 3.3.1. Finally, by means of Lemma 4.5.2, we show that both queries evaluate to true . The proof follows these steps in both directions.

Let $\mathcal{A}$ be an answer set of the program.

Assume, $Q^\gamma_s(f_{\mathcal{C}}(\gamma D^i_Q)) \neq \emptyset$ for an arbitrary but fixed γ. Then, due to Lemma 4.5.1, $val(\gamma)$ must be in $\mathcal{A}$. We know that D^u_Q must be in EDB as a set of ground facts. Due to Lemma 3.3.1, for every fact $R(\bar{t})$ in $f_{\mathcal{C}}(\gamma D^i_Q)$, there is a fact $R^a(\bar{t})$ in $\mathcal{A}$. Taking them both into account, we apply Lemma 4.5.2 to the test-rule Q^u. We took γ to be arbitrary but fixed; then, the proof holds for any γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$.

Assume, Q holds in every answer set of the program. We take an arbitrary but fixed answer set $\mathcal{A}$. Due to Lemma 4.5.1, there must be a corresponding γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Then, the prototypical database is γD^i_Q. According to Lemma 3.3.1, we establish correspondence between the facts over the Σ^a signature and the facts in $f_{\mathcal{C}}(\gamma D^i_Q)$. As a result, we deduce that for every fact $R^a(\bar{t})$

in $\mathcal{A}$, there is a fact $R(\bar{t})$ in $f_{\mathcal{C}}(\gamma D^i_Q)$. Due to Lemma 4.5.2, we conclude that test query Q^γ_s evaluates to true. ∎

Having introduced the encoding theorem, we can design an effective (from a complexity point of view) reasoner for the TC-QC completeness problem in the presence of finite domain constraints. By design, this procedure follows the characterization that exactly captures query completeness reasoning. It is an important theorem in development of a practical completeness reasoner.

Having connected two formalisms, we have made a feasible decision procedure for TC-QC reasoning. It can be used in practice provided there is an efficient ASP reasoner available.

CHAPTER 5

Reasoning with Foreign Key Constraints

Overview

In this chapter, we discuss completeness reasoning in the presence of foreign keys. We start with an example illustrating how completeness reasoning works in a typical case. We show how foreign keys affect TC-QC reasoning in general.

We discuss two semantics and formalizations of foreign keys in the context of completeness reasoning. Assuming acyclicity of foreign keys, we characterize query completeness in the presence of foreign keys. We also show how to modify the encoding to reflect the changes in the characterization.

At the end, we combine finite domain and foreign key constraints. We show how they interact with each other. Finally, we introduce the characterization and the encoding working with the both types of constraints.

5.1 Example

In this section, by means of an informal example, we show how the reasoning procedure works in a typical case. We revise the example from the third chapter and incorporate foreign keys[1] into the reasoning procedure.

There are two ways how to interpret foreign keys in the context of completeness reasoning: to impose them on the ideal database only, or to impose them on both databases. The example shows the first approach. Let us give an intuition about these two approaches. Foreign keys over the ideal database represent knowledge about the domain i.e., the real world. Foreign keys over both represent knowledge about the real state of the available database. The main difference in the inference techniques is explained later.

[1] For basic definitions, syntax and semantics of foreign key constraints, The section 2.4.

Assume, the schema Σ is the same as before

$$\textit{Employee}(\underline{\textit{Name}}, \textit{DeptName}, \textit{Birthday}). \tag{5.1}$$
$$\textit{Department}(\underline{\textit{DeptName}}, \textit{Location}). \tag{5.2}$$

In addition, there is a foreign key constraint in Σ. Assume, the foreign key is[2]

$$\textit{Employee}(\_, \textit{DeptName}, \_) \text{ references } \textit{Department}(\textit{DeptName}, \_). \tag{5.3}$$

Simply speaking, this constraint says

"*Every department name in the Employee relation must be in the Department relation.*"

Suppose, the query is

$$Q(\textit{Name}) \leftarrow \textit{Employee}(\textit{Name}, \textit{DeptName}, \textit{Birthday}). \tag{5.4}$$

We also have the TC statement

$$\begin{aligned} \textit{Employee}^a\textit{(Name,DeptName,Bday)} \leftarrow \quad & \textit{Employee}^i\textit{(Name,DeptName,Bday)}, \\ & \textit{Department}^i\textit{(DeptName,Location)}. \end{aligned} \tag{5.5}$$

Below, we are going to revise the reasoning procedure we presented in the example in the third chapter. We explain it step by step showing interaction between the original procedure and new constraints.

Let us note that if we just repeat all steps from the relational procedure, then our procedure will not return a positive answer. In fact, the query is complete (since completeness is logically entailed by the TC statements), but the procedure is unable to find this out. The TC rule needs an *Employee*-atom and a *Department*-atom to fire, however, the prototypical database contains only one atom. Intuitively, the foreign key constraint forces the second atom to be in the prototypical database. We have to modify the procedure to reflect the new constraints in the schema. In general, foreign keys give us additional possibilities to establish completeness. Even if the query is not ensured to be complete by the relational procedure, it might be complete provided there are foreign keys in the schema. See next paragraph for technical details on it.

1. Determine what tuples have to be in D^i

Assume, *name* is in the answer to Q over D^i, that is,

$$\textit{name} \in Q(D^i). \tag{5.6}$$

Since Q is a conjunctive query, there must be a corresponding body atom in D^i

$$\textit{Employee}(\textit{name}, \textit{deptName}, \textit{bday}) \in D^i. \tag{5.7}$$

[2]For the foreign key syntax, see Definition 2.4.4

If we proceed according to the relational procedure, then the TC statement does not apply and we cannot establish query completeness.

Due to the FK constraint, for every department name in the *Employee* relation there must be a corresponding tuple in the *Department* relation. Assume, *deptName* is the value of the *DeptName* column in the *Employee* relation; for some value *location* there must be a tuple $(\mathit{deptName}, \mathit{location})$ in the *Department* relation

$$\mathit{Department}(\mathit{deptName}, \mathit{location}) \in D^i. \tag{5.8}$$

We have established that the following facts must be in D^i

$$\begin{aligned} \mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}) &\in D^i, \\ \mathit{Department}(\mathit{deptName}, \mathit{location}) &\in D^i. \end{aligned} \tag{5.9}$$

2. Determine what tuples have to be in D^a

In this example, we have only one TC statement 5.5 that we can apply to D^i. As a result, we obtain an atom

$$\mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}) \in D^a. \tag{5.10}$$

This is the only fact established to be in D^a.

3. Verify completeness

We assumed *name* to be in the answer to Q over D^i. Now we evaluate Q over D^a and obtain *name* as an answer tuple. According to Definition 2.2.2, we found out that Q is complete.

Visualization of reasoning procedure

As before, we used the query to establish what atoms must be present in D^i_Q to produce the query answer. However, in our example these atoms are not sufficient to apply the TC rule. Still, we can establish more atoms by applying the FK statements, and the TC rule may fire over the extended ideal database.

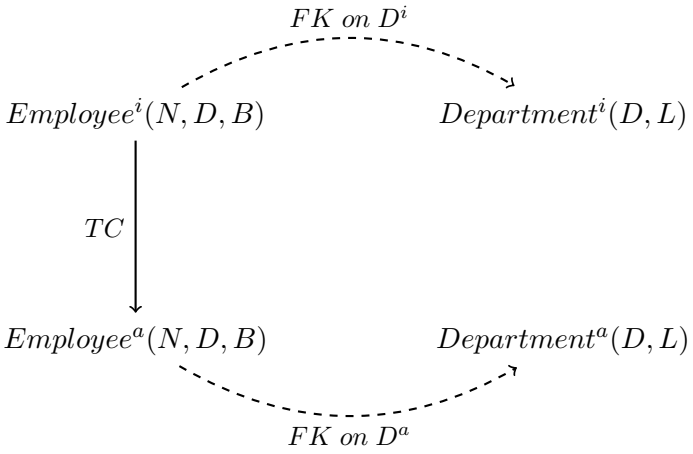
This idea can be nicely depicted, see Figure 5.1. Normally, atoms go from D^i to D^a by means of TC statements. Having established facts in D^i, foreign keys give us an additional way to infer facts. As we can see later, if we impose foreign keys on D^a as well, we can use the same idea to infer even more facts by using foreign keys.

5.2 Formalization

In this section, we are going to discuss semantics of foreign keys. To formalize reasoning with foreign keys, we investigate their properties and features in the context of query completeness.

Inclusion dependencies filter models. They specify what tuple must present in the database to satisfy constraints. In this sense, they are very close to the TC statements; a TC statement

Figure 5.1: Visualization of reasoning example with FK



rejects a model, if a certain fact in the ideal database and it is not in the available database. A foreign key rejects a model, if a certain fact in the database and the fact that it references is not.

Foreign keys as well as TC statements determine a lower bound on the facts in a partial database. They enforce some tuples to be present under certain conditions. For comparison, finite domain constraints determine an upper bound on the facts in a partial database. They enforce only certain facts to be present in the database and prohibit the others.

We say that TC-QC entailment in the presence of foreign keys holds, if for any partial database satisfying the TC statements and the foreign key constraints, the query is complete. To define it formally – it is an important question to answer: which database the ideal or both must satisfy foreign keys? In this section, we start with foreign key constraints imposed on the ideal database. We call it TC-QC entailment under ideal semantics. We define it formally as

Definition 5.2.1 (TC-QC Entailment under Ideal Semanticss) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{K}$ be a set of FK. Then, $\mathcal{C}$ and $\mathcal{K}$ entail completeness of Q under ideal semantics written $\mathcal{C}, \mathcal{K} \models^i Compl(Q)$ iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \text{ and } D^i \models \mathcal{K} \implies Q(D^i) = Q(D^a). \tag{5.11}$$

There is also an option to impose foreign keys on both databases. In this case, we have different semantics for TC-QC entailment that we call enforced semantics. It does not make sense to impose FK on only the available database since it is always a subset of the ideal database.

To fix the notation, we say that a partial database $\mathcal{D} = (D^i, D^a)$ satisfies a set of FK $\mathcal{K}$, written as $\mathcal{D} \models \mathcal{K}$ iff

$$D^i \models \mathcal{K} \text{ and } D^a \models \mathcal{K}. \tag{5.12}$$

Now we define formally TC-QC entailment with foreign keys under enforced semantics.

Definition 5.2.2 (TC-QC Entailment under Enforced Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{K}$ be a set of FK. Then, $\mathcal{C}$ and $\mathcal{K}$ entail completeness of Q under enforced*

semantics written $\mathcal{C}, \mathcal{K} \models \mathit{Compl}(Q)$ *iff for any partial database* $\mathcal{D} = (D^i, D^a)$ *it holds that*

$$\mathcal{D} \models \mathcal{C} \text{ and } \mathcal{D} \models \mathcal{K} \implies Q(D^i) = Q(D^a). \tag{5.13}$$

5.3 Characterization

Characterization under ideal semantics

In this section, we are going to discuss TC-QC reasoning with foreign keys. Under the assumption that foreign keys are acyclic[3], we present a TC-QC characterization that captures reasoning with foreign keys. We assume foreign key constraints to be acyclic throughout the whole work.

We need to revise the relational procedure to capture additional constraints in the schema. In this section, we consider only foreign keys imposed only on the ideal database, that is why we call it ideal semantics.

We are going to generalize the example from the first section. There we applied foreign keys as rules to enforce some tuples to be present in the database. We have already seen this idea when we talked about the function $f_{\mathcal{C}}$. It maps facts from D^i to facts in D^a. In our case, our new procedure will introduce some new facts in D^i based on the other facts in D^i.

It give us an idea how to apply this function in the characterization – it should apply to D^i before $f_{\mathcal{C}}$ to allow it to infer more facts. Following the ideas from Cali et al. [2], we define this function by means of a chase procedure[4]. We denote it as $\mathcal{K}^i$, see Algorithm 5.1.

Require: a database D
Require: a set of foreign keys $\mathcal{K}$
Ensure: the database D satisfying $\mathcal{K}$
{The expression of the form $R[X]$ is a syntactic sugar for $\pi_X(R)$}
1: **repeat**
2: $D' := D$
3: **if** there is a fact $R(t)$ in D and an FK $R[X] \subseteq S[Y]$ in $\mathcal{K}$ s.t. there exists no fact $S(t')$ in D where $t'[Y] = t[X]$ **then**
4: create a fact $S(t'')$ such that $t''[Y] = t[X]$ and create fresh values in t'' on the remaining positions
5: $D \leftarrow D \cup \{S(t'')\}$
6: **end if**
7: **until** $D' = D$
8:
9: **return** D

Algorithm 5.1: Co-chase procedure: $\mathcal{K}^i$.

Lemma 5.3.1 *Let* $\mathcal{K}$ *be a set of FK. Then,* $\mathcal{K}^i(D)$ *is a minimal extension of* D *(under set inclusion) satisfying* $\mathcal{K}$*.*

[3] For details about the acyclicity assumption, see Section 2.4

[4] To be precise, it is a co-chase procedure, according to the classification of Alin Deutsch [5]

Proof Let us prove both properties of $\mathcal{K}^i(D)$ – satisfaction and minimality.

Assume, $\mathcal{K}^i(D)$ does not satisfy $\mathcal{K}$. Then, there is a tuple t in $R(D)$, an FK $R[X] \subseteq S[Y]$ and there is no $t' \in S(D)$ such that $t'[Y] = t[X]$. Due to the acyclicity assumption, $\mathcal{K}^i$ terminates, then the rule in Algorithm 5.1 in Line 4 is not applicable. Consequently, for the tuple t, the rule creates a tuple t'' such that $t''[Y] = t[X]$ and $S(t'') \in D$. On the one hand, there is no tuple $t' \in S(D)$ s.t. $t'[Y] = t[X]$; on the other hand, there is $t'' \in S(D)$ such that $t''[Y] = t[X]$. We derived contradiction and by contraposition, $\mathcal{K}^i(D)$ satisfies $\mathcal{K}$.

Assume, $\mathcal{K}^i(D)$ is not a minimal extension of D. Then there is a smaller extension D' of D satisfying $\mathcal{K}$. There is at least one tuple t in the difference $\mathcal{K}^i(D) \backslash D'$. The tuple t as well as all tuples in $\mathcal{K}^i(D) \backslash D$ are added by the rule the line 4 of Chase Procedure 5.1. Then, $\mathcal{K}^i(D) \backslash \{t\}$ does not satisfy $\mathcal{K}$, because there is an FK $R[X] \subseteq S[Y]$ in $\mathcal{K}$ and there is no tuple $t' \in S(D)$ s.t. $t'[Y] = t[X]$. Contradiction. ∎

The new TC-QC characterization with foreign keys under ideal semantics illustrates and shows how completeness reasoning changes due to introduction of constraints in the schema. It is meant to show and give an idea how to change the encoding. We regard the characterization as a specification of the problem. If we change the specification, it means we have to reflect these changes in the program as well.

Theorem 5.3.2 (Characterization with FK under Ideal Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements and $\mathcal{K}$ be a set of FK. Then,*

$$\mathcal{C}, \mathcal{K} \models^i \mathit{Compl}(Q) \iff Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q))) \neq \emptyset. \tag{5.14}$$

Proof ($\Rightarrow$) Assume, $\mathcal{C}, \mathcal{K} \models^i \mathit{Compl}(Q)$. Consequently, we consider only D^i that satisfies $\mathcal{K}$. Then, $\mathcal{K}^i(D^i) = D^i$. Due to completeness of Q, it holds that $Q(D^i) = Q(D^a)$. Due to Lemma 3.2.2, it holds that $Q(D^a) = Q(f_{\mathcal{C}}(D^i))$. By substitution of equals, we obtain $Q(D^a) = Q(f_{\mathcal{C}}(\mathcal{K}^i(D^i)))$.

Assume, a tuple $\bar{c} = \alpha\bar{X}$ is in the answer to Q over D^i. Taking into account that D^i_Q is the frozen by θ version of the body B of the query Q with distinguished variables $\bar{X}$. Then, $\alpha B \subseteq D^i$ and $\alpha\theta^{-1}$ maps D^i_Q to D^i. Due to Lemma 5.3.1, $\mathcal{K}^i$ is monotone function, then the assignment $\alpha\theta^{-1}$ maps $\mathcal{K}^i(D^i_Q)$ to D^i. Consequently, $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q))) \neq \emptyset$.

($\Leftarrow$) Assume, $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q))) \neq \emptyset$. We want to show that $\mathcal{C}, \mathcal{K} \models^i \mathit{Compl}^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database such that $\mathcal{D} \models \mathcal{C}$ and $D^i \models \mathcal{K}$ where $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Let α be an assignment from Q into D^i that maps $\bar{X}$ to $\bar{c}$. Then $\alpha B \subseteq D^i$ where B is the body of Q. Considering that $f_{\mathcal{C}}$ is a monotone function (due to Lemma 3.2.2) over database instances it follows that $f_{\mathcal{C}}(\alpha B) \subseteq f_{\mathcal{C}}(D^i)$. By assumption it holds that $D^i \models \mathcal{K}$ and $\mathcal{K}^i(D^i) = D^i$. By definition, $\mathcal{K}^i$ is a monotone function, consequently $f_{\mathcal{C}}(\mathcal{K}^i(\alpha B)) \subseteq f_{\mathcal{C}}(\mathcal{K}^i(D^i))$. Again from Lemma 3.2.2 we have that $f_{\mathcal{C}}(D^i) \subseteq D^a$ when $(D^i, D^a) \models \mathcal{C}$. Then, $f_{\mathcal{C}}(\mathcal{K}^i(\alpha B)) \subseteq D^a$.

Now let β be an assignments for which $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q)))$ returns an empty tuple. Then a composition $\alpha\theta^{-1}\beta$ is a proper assignment over the variables from Q into the database instance $\alpha\theta^{-1} f_{\mathcal{C}}(\mathcal{K}^i((\theta B)))$ for which Q returns an empty tuple as well. On the other hand, it is not hard to check that $\alpha\theta^{-1} f_{\mathcal{C}}(\mathcal{K}^i(\theta B)) \subseteq f_{\mathcal{C}}(\mathcal{K}^i(\alpha B))$. Consequently $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(\alpha B))) \neq \emptyset$.

Now $\alpha\theta^{-1}\beta$ maps $\bar{X}$ to $\bar{c}$. Then evaluating $Q(X)$ against $f_{\mathcal{C}}(\mathcal{K}^i(\alpha B))$ we obtain $\bar{c}$. Finally, considering that $f_{\mathcal{C}}(\mathcal{K}^i(\alpha B)) \subseteq D^a$ we conclude that $\bar{c} \in Q(D^a)$. ∎

The characterization allows us to design a program that checks TC-QC entailment. We make an argument that the program we create has the same semantics as the characterization we have presented. That is why it is important to introduce a characterization as a specification that regulates how different parts must interact and how reasoning should work.

Characterization under enforced semantics

In this section, we discuss enforced semantics for TC-QC reasoning. Simply speaking, this semantics imposes foreign keys on the available database as well as on the ideal database.

We define a new function working with the available database. For a given ideal database satisfying foreign keys it extends the available database to satisfy foreign keys as well. We denote this function as $\mathcal{K}^a$. We define it by means of a chase procedure, see the algorithm 5.2.

Require: a partial database $\mathcal{D} = (D^i, D^a)$, where D^i satisfies foreign keys
Require: a set of foreign keys $\mathcal{K}$
Ensure: the available database D^a) satisfying foreign keys
1: **repeat**
2: $D' := D^a$
3: **if** there is a fact $R(t)$ in D^a and an FK $f = R[X] \subseteq S[Y]$ in $\mathcal{K}$ s.t. there is no fact $S(t')$ in D^a s.t. $t'[Y] = t[X]$ **then**
4: find the fact $R(t)$ in D^i and the fact $S(t'')$ in D^i corresponding by f
5: $D^a \leftarrow D^a \cup \{S(t'')\}$
6: **end if**
7: **until** $D' = D^a$
8:
9: **return** D'

Algorithm 5.2: Co-chase procedure: $\mathcal{K}^a$.

Lemma 5.3.3 *Let $\mathcal{K}$ be a set of FK. Let $\mathcal{D} = (D^i, D^a)$ be a partial database, where D^i satisfies $\mathcal{K}$. Then, $\mathcal{K}^a(\mathcal{D})$ is the minimal extension of D^a (under set inclusion) satisfying $\mathcal{K}$.*

Proof Let us prove both properties: satisfaction and minimality.

Assume, $\mathcal{K}^a(\mathcal{D})$ does not satisfy $\mathcal{K}$. Then, there is a tuple $t \in R(D^a)$, an FK $f = R[X] \subseteq S[Y] \in \mathcal{K}$ and there is no $t' \in S(D^a)$ s.t. $t'[Y] = t[X]$. By the initial assumption, D^i satisfies $\mathcal{K}$– there is $t' \in S(D^i)$ s.t. $t'[Y] = t[X]$. In Algorithm 5.2, the condition in Line 3 is satisfied by t and the rule in Line 3 finds t' in D^i. On the one hand, we assumed there is no t' s.t. $t'[Y] = t[X]$; on the other hand, the rule in Line 5 has to add it to the database. Contradiction.

Assume, $\mathcal{K}^a(\mathcal{D})$ is not a minimal extension. Then, there is an extension D' of D^a satisfying $\mathcal{K}$ s.t. $D' \subset \mathcal{K}^a(\mathcal{D})$. There is at least one tuple t in the difference $\mathcal{K}^a(\mathcal{D})\backslash D'$. This tuple as well as all tuples added in Procedure 5.2 due to the condition in Line 5. Then, $\mathcal{K}^a(\mathcal{D})\backslash\{t\}$ does

not satisfy $\mathcal{K}$. Thus, t must be referenced by some FK $f = R[X] \subseteq S[Y]$, then there is a tuple $t' \in R(D^a)$ such that there is no $t \in S(D^a)$ and $t[Y] = t[X]$. Contradiction. ∎

Having defined the function $\mathcal{K}^a$, we revise the TC-QC characterization to capture the new semantics. The new TC-QC characterization with foreign keys under enforced semantics illustrates and shows how completeness reasoning changes due to the introduction of constraints in the schema. We regard the characterization as a specification of the problem. If we change the specification, it means we have to reflect these changes in the encoding as well.

The operator $\mathcal{K}_\mathcal{C}$ The operator $\mathcal{K}^a$ is defined over partial databases. In the characterization in the previous chapter, we have seen that the ideal database and the available databases are strongly related. It would be redundant to write twice similar constructions. That is why we introduce the new operator $\mathcal{K}_\mathcal{C}$. It takes as an argument a database instance and it is parameterized by the set of foreign keys $\mathcal{K}$ and by the set of TC statements $\mathcal{C}$. The operator $\mathcal{K}_\mathcal{C}$ is defined by the following equation

$$\mathcal{K}_\mathcal{C}(D) = \mathcal{K}^a(\mathcal{K}^i(D), f_\mathcal{C}(\mathcal{K}^i(D))). \tag{5.15}$$

It passes to the operator $\mathcal{K}^a$ two arguments: the ideal database $\mathcal{K}^i(D)$ that satisfies the set of foreign keys $\mathcal{K}$, and the available database $f_\mathcal{C}(\mathcal{K}^i(D))$.

Now we can introduce the characterization in a readable way.

Theorem 5.3.4 (Characterization with FK under Enforced Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{K}$ be a set of FK, then*

$$\mathcal{C}, \mathcal{K} \models \mathit{Compl}(Q) \iff Q_s(\mathcal{K}_\mathcal{C}(D^i_Q)) \neq \emptyset. \tag{5.16}$$

Proof ($\Rightarrow$) Assume that $\mathcal{C}, \mathcal{K} \models \mathit{Compl}(Q)$. Then, we consider only D^i and D^a that satisfy foreign keys: $D^i \models \mathcal{K}$ and $D^a \models \mathcal{K}$. Then, $\mathcal{K}^i(D^i) = D^i$ and $\mathcal{K}^a(\mathcal{D}) = D^a$. Equality holds $Q(D^i) = Q(D^a)$, due to the fact that $D \models \mathcal{C}$. From Lemma 3.2.2 it follows that $Q(D^a) = Q(f_\mathcal{C}(D^i))$. By substitution of equals, we obtain $Q(D^a) = Q(\mathcal{K}_\mathcal{C}(D^i))$.

By definition, D^i_Q is the frozen by θ version of query Q with the body B and distinguished variables $\bar{X}$. Assume, a tuple $\bar{c} = \alpha\bar{X}$ is in the answer to Q over D^i. Then, $\alpha B \subseteq D^i$ and $\alpha\theta^{-1}$ maps D^i_Q to D^i. Due to Lemma 5.3.1, $\mathcal{K}^i$ is monotone function, then $\alpha\theta^{-1}\mathcal{K}^i(D^i_Q) \subseteq D^i$. Due to Lemma 5.3.3, $\mathcal{K}^a$ is a monotone function, then $\mathcal{K}_\mathcal{C}(D^i) \subseteq D^a$. Consequently, $Q_s(\mathcal{K}_\mathcal{C}(D^i_Q)) \neq \emptyset$.

($\Leftarrow$) Assume, $Q_s(\mathcal{K}_\mathcal{C}(D^i_Q)) \neq \emptyset$. We want to show that $\mathcal{C}, \mathcal{K} \models \mathit{Compl}^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database such that $\mathcal{D} \models \mathcal{C}$ where $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Let α be an assignment from Q into D^i that maps $\bar{X}$ to $\bar{c}$. Then $\alpha B \subseteq D^i$ where B is the body of Q. Considering that $f_\mathcal{C}$ is a monotone function (due to Lemma 3.2.2) over database instances it follows that $f_\mathcal{C}(\alpha B) \subseteq f_\mathcal{C}(D^i)$. By assumption $D^i \models \mathcal{K}$, then due to Lemma 5.3.1 $\mathcal{K}^i$ is a monotone function and $\mathcal{K}^i(D^i) = D^i$, it follows that $f_\mathcal{C}(\mathcal{K}^i(\alpha B)) \subseteq f_\mathcal{C}(\mathcal{K}^i(D^i))$. Again from Lemma 3.2.2 we have that $f_\mathcal{C}(D^i) \subseteq D^a$ when $(D^i, D^a) \models \mathcal{C}$, so we conclude that $f_\mathcal{C}(\alpha B) \subseteq D^a$. Then, it holds that $f_\mathcal{C}(\mathcal{K}^i(\alpha B)) \subseteq D^a$. Due to the initial assumption, $D^a \models \mathcal{K}$,

so it holds that $\mathcal{K}^a(\mathcal{D}) = D^a$. Due to Lemma 5.3.3, $\mathcal{K}^a$ is a monotone function and we conclude that $\mathcal{K}_\mathcal{C}(\alpha B) \subseteq D^a$.

Now let β be an assignments for which $Q_s(\mathcal{K}_\mathcal{C}(D^i_Q))$ returns an empty tuple. Then a composition $\alpha\theta^{-1}\beta$ is a proper assignment over the variables from Q into database instance $\alpha\theta^{-1}\mathcal{K}_\mathcal{C}(\theta B)$ for which Q returns an empty tuple as well. On the other hand, it is not hard to check that $\alpha\theta^{-1}\mathcal{K}_\mathcal{C}(\theta B)) \subseteq \mathcal{K}_\mathcal{C}(\alpha B)$. Consequently $Q_s(\mathcal{K}_\mathcal{C}(\alpha B)) \neq \emptyset$.

Now $\alpha\theta^{-1}\beta$ maps $\bar{X}$ to $\bar{c}$. Then evaluating $Q(X)$ against $\mathcal{K}_\mathcal{C}(\alpha B)$ we obtain $\bar{c}$. Finally, considering that $\mathcal{K}_\mathcal{C}(\alpha B) \subseteq D^a$ we conclude that $\bar{c} \in Q(D^a)$. ∎

Introduction of the characterization allows us to design a program that checks TC-QC entailment. We make an argument that the program we create has the same semantics as the characterization we have presented. That is why it is important to introduce a characterization as a specification how different parts must interact and how reasoning should work.

A note on complexity In this paragraph, we refer to the Klug's work (IND – Inclusion Dependencies) [10]:

"The containment problem for conjunctive queries remains in *NP* for any fixed bound on the maximum width of an IND (the width of an IND is the number of attributes occurring on either of its sides)."

At the beginning of the section, we defined a foreign key as a combination of an inclusion dependency and a functional dependency. We have our schema fixed and all FK are fixed. The list of attributes of every FK is fixed on both sides. Consequently, the problem of TC-QC reasoning with FK is in *NP*.

5.4 Encoding

Foreig keys encoding under ideal semantics

In this section, we are going to present an encoding of the TC-QC entailment with foreign keys under ideal semantics. It is an extension of the relation encoding with an additional module $P^i_\mathcal{K}$. It captures additional constraints imposed over the ideal database.

In the characterization we have seen the only thing that changes is the prototypical database. That is why we only need to encode the application of $\mathcal{K}^i$ to the prototypical database D^i_Q. In the encoding, we have two signatures Σ^i and Σ^a. We have to apply the new constraints to Σ^i, since they correspond to D^i_Q.

Wlog we assume that all primary key columns are at the beginning of the relations and all referencing columns of all foreign keys are at the end. For every foreign key $f = R[A] \subseteq S[B]$, we define two rules r^1_f and r^2_f. In these rules, $\bar{B}$ is a tuple of distinct variables in the position of the referencing attributes of R and $\bar{C}$ is a tuple of distinct variables in the remaining positions of R. In S, $\bar{B}$ occurs in the positions of the key attributes. The tuple $\bar{A}$ does not explicitly occur in the rules because we join the primary key arguments of S with the referencing arguments of R

and denote them using only one vector of variables $\bar{B}$.

$$\begin{aligned} S(\bar{B}, f_S(\bar{B})) &\leftarrow R(\bar{C}, \bar{B}),\ \text{not aux}S(\bar{B}). \\ \text{aux}S(\bar{B}) &\leftarrow S(\bar{B}, \bar{D}),\ \textit{original}(\bar{D}). \end{aligned} \tag{5.17}$$

If we look at the semantics of these rules, we can see why they encode the co-chase procedure $\mathcal{K}^i$. Observe that the second rule projects the arguments of S on the positions in $\bar{B}$, it corresponds to the "such that" part in the condition of Algorithm 5.1 in Line 4. Then, for every R-atom the first rule checks, if there is a corresponding S-atom. We can see that if there is no corresponding S-atom, then the rule creates it and adds to the answer set.

We make use of Skolem functions to guarantee uniqueness of the fresh constants that we introduce. By means of the predicate *original* we make sure that added facts do not violate the rule r_f^1. If we omit this atom in the *aux*-predicates, then any added fact would be violate r_f^1 (see the example below for clarifications).

For the sake of readability and compactness, we have written $\textit{original}(\bar{D})$. Strictly speaking, the predicate *original* is unary. This construction $\textit{original}(\bar{D})$ should be read as a conjunction of *original*-atoms for every variable d in $\bar{D}$. It is an abbreviation for the following statement

$$\textit{original}(\bar{D}) \iff \textit{original}(d_1), \ldots, \textit{original}(d_n). \tag{5.18}$$

where each d_i is in $\bar{D}$.

We generalize these rules by considering $\bar{A}, \bar{B}$ as lists of indices. Then, we join variables according to the foreign keys. Let us illustrate this by an example. Assume, there is a foreign key[5]

$$\textit{Employee}(\_, \textit{DeptName}, \_) \text{ references } \textit{Department}(\textit{DeptName}, \_).$$

Then, the rules are

$$\begin{gathered} \textit{Department}(D, f_D(D)) \leftarrow \textit{Employee}(N, D, B),\ \text{not aux}\textit{Department}(D). \\ \text{aux}\textit{Department}(D) \leftarrow \textit{Department}(D, L), \textit{original}(L). \end{gathered}$$

Let us show, what happens if we omit the predicate *original*. Assume, there is an atom in the ideal database

$$\textit{Employee}(\textit{name}, \textit{deptName}, \textit{bday}) \in D^i.$$

and there is no corresponding *Department*-atom in D^i. Then, the rule r_f^1 fires and adds the atom to D^i

$$\textit{Department}(\textit{deptName}, f_{\textit{Department}}(\textit{deptName})) \in D^i.$$

It triggers the second rule r_f^2 and it adds the fact to D^i.

$$\text{aux}\textit{Department}(\textit{deptName}) \in D^i.$$

On the one hand, the atom $\textit{Department}(\textit{deptName}, f_{\textit{Department}}(\textit{deptName}))$ must be added to the answer set due to the rule r_f^1; on the other hand, the rule r_f^1 must be removed since its body

[5]For the foreign key syntax, see Definition 2.4.4

contradicts to the answer set. In answer set programming this kind of rules is called "killer" rules. In this example, there is no answer set that contains an *Employee*-atom, because it would be rejected by the "killer" rules r_f^1 and r_f^2.

To avoid this effect, we have introduced the predicate *original*. If we add it to the bodies of the "aux"-rules, then newly added facts do not trigger r_f^2 and there is no "killer" rule.

We denote the pair r_f^1 and r_f^2 as r_f

$$r_f = \{r_f^1,\ r_f^2\}.$$

We also need to introduce all original constants and frozen variables into the encoding. Let us denote the set of all terms that occur in a query Q as *terms*(Q). Then, we define the program $P_{\mathcal{K}}^i$ as

$$P_{\mathcal{K}}^i = \bigcup_{f \in \mathcal{K}} r_f \cup \{\textit{original}(t) \mid t \in \textit{terms}(Q)\}.$$

We state a lemma to make a connection between these constraints and the $\mathcal{K}^i$ operator that we have introduced in the characterization section.

Lemma 5.4.1 (Correspondence: $\mathcal{K}^i$ and $P_{\mathcal{K}}^i$) *Let D be a database. Then, a fact $S(\bar{t})$ is in $\mathcal{K}^i(D)$ iff $S(\bar{t})$ is in the answer set of $P_{\mathcal{K}}^i \cup D$.*

Proof Note that under the acyclicity assumption, $P_{\mathcal{K}}^i$ has a unique answer set (each pair of rules is consistent due to the conjunction of *original*-atoms). We denote the answer set of $P_{\mathcal{K}}^i \cup D$ as $\mathcal{A}$.

($\Rightarrow$) Assume, there is a fact $S(\bar{t})$ in $\mathcal{K}^i(D)$. If it is in D, then it is trivially in $\mathcal{A}$. If it is not, then it was added by the rule of Algorithm 5.1 in Line 4. Assume, $S(\bar{t})$ is not in $\mathcal{A}$. If it was added by the rule in the chase procedure, then there is $f = R[A] \subseteq S[B] \in \mathcal{K}$ such that $R[A] \subseteq S[B]$. Consequently, there is a fact $R(\bar{A}, \bar{C}) \in D$. Then the rule $r_{\mathcal{K}}^1$ must fire: "not aux$S(\bar{B})$" is satisfied because we assumed that $S(\bar{t})$ is not in $\mathcal{A}$, and the conjunction of *original*-atoms is not satisfied since the fact $S(\bar{t})$ is not in D. The $R(\bar{A}, \bar{C})$ is in D by assumption. We obtain contradiction; on the one hand, there must be $S(\bar{t})$ due to the rule $r_{\mathcal{K}}^1$; on the other hand, we assumed it is not in $\mathcal{A}$.

($\Leftarrow$) Assume, there is a fact $S(\bar{t})$ in $\mathcal{A}$. If it is in D, then it is trivially in $\mathcal{K}^i(D)$. If it is not, then it was added by r_f^1 where $f = R[A] \subseteq S[B]$ is an FK in $\mathcal{K}$. Then there is a fact $R(\bar{A}, \bar{C}) \in D$, and there is no $S(\bar{B}, \bar{D}) \in D$. Assume, there is no $S(\bar{B}, \bar{D}) \in \mathcal{K}^i(D)$. There is a contradiction between the rule of Algorithm 5.1 in Line 4 and this assumption. On the one hand, the fact must be added to $\mathcal{K}^i(D)$; on the other hand, we assumed it is not in $\mathcal{K}^i(D)$. ∎

Theorem 5.4.2 (Encoding with FK under Ideal Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements and $\mathcal{K}$ be a set of FK. Then,*

$$\mathcal{C}, \mathcal{K} \models^i \textit{Compl}(Q) \iff P_Q \cup P_{\mathcal{C}} \cup P_{\mathcal{K}}^i \textit{ has an answer set.}$$

Proof Instead of showing the correspondence between the encoding and the definition of query completeness, we show the correspondence between the encoding and the characterization.

$$Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D_Q^i))) \neq \emptyset \iff P_Q \cup P_{\mathcal{C}} \cup P_{\mathcal{K}}^i \text{ has an answer set.}$$

The proof is technical, that is why we briefly explain the main points of the proof. The proof appeals to the characterization presented before. It has four logical levels. Firstly, we establish a correspondence between ground facts in the encoding and the prototypical database in the characterization. Secondly, by means of Lemma 5.4.1, we show a correspondence between the prototypical database extended by $\mathcal{K}^i$ and the set of facts over the Σ^i signature extended by $P^i_{\mathcal{K}}$. Then, we show a correspondence between the sets we derive from the prototypical database extended by $\mathcal{K}^i$ and facts over the Σ^i signature, namely between the set of facts over the Σ^a signature and $f_{\mathcal{C}}(D^i_Q)$, by means of Lemma 3.3.1. Finally, by means of Lemma 4.5.2, we show that both queries evaluate to true. The proof follows these steps in both directions.

The difference between the proof in the relational case and this proof is in application of $\mathcal{K}^i$ in the characterization and $P^i_{\mathcal{K}}$ in the encoding. The procedure $\mathcal{K}^i$ only affects D^i_Q and $P^i_{\mathcal{K}}$ only affects EDB – the set of facts over the Σ^i signature. Then, we need to show correspondence between the set of facts in the answer set $\mathcal{A}$ over Σ^i signature and $\mathcal{K}^i(D^i_Q)$. Let D^i_Q be a database instance over Σ^i signature, then $\mathcal{K}^i(D^i_Q)$ and the set of facts over Σ^i signature in the answer set are equal, due to Lemma 5.4.1.

Then, the proof is the same as the proof of Theorem 3.3.3, since both of the inclusion arguments refer to D^i_Q and EDB, which in our case are proven to be equal – as well as they are in Theorem 3.3.3. Let us to denote the program $P_Q \cup P_{\mathcal{C}} \cup P^i_{\mathcal{K}}$ as P.

($\Rightarrow$) Assume, P has an answer set $\mathcal{A}$. Then it has Q_s in $\mathcal{A}$ due to the filtering rule. Consequently, there is a mapping β such that it maps the body B^a of Q_s to the facts in $\mathcal{A}$ over the Σ^a signature.

The body B of the test query $Q_s()$ is the same set of atoms as the body B^a of the test-rule Q_s except of the upper index $\bullet^a$ 3.25. For every atom $R^a(\bar{t}) \in B^a$, there is an atom $R(\bar{t}) \in B$ such that they are the same except of the index. We know from previous paragraph that $\beta R^a(\bar{t}) \in \mathcal{A}$, consequently $\beta R(\bar{t}) \in f_{\mathcal{C}}(D^i_Q)$. Due to Lemma 3.3.1 for every fact over Σ^a signature, there is corresponding fact in $f_{\mathcal{C}}(D^i_Q)$ over Σ^i signature. $Q_s()$ is a boolean conjunctive query and every atom of the body(B) is mapped to $f_{\mathcal{C}}(D^i_Q)$ by β. Ergo, it returns the empty tuple.

($\Leftarrow$) Assume, $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q))) \neq \emptyset$. There is a mapping β that maps every atom $R(\bar{t})$ in the body of $Q_s()$ to $f_{\mathcal{C}}(D^i_Q)$. Due to 3.25 for every atom $R(\bar{t})$ in the body of test query $Q_s()$, there is corresponding atom $R^a(\bar{t})$ in the body of encoding of the test-query Q_s. Due to Lemma 3.3.1 for every atom $R(\bar{t}) \in f_{\mathcal{C}}(D^i_Q)$, there is an atom $R^a(\bar{t}) \in \mathcal{A}$. Consequently, for every atom $R^a(\bar{t})$ in the body of encoding of the test query, it follows that $\beta R^a(\bar{t}) \in \mathcal{A}$. Then, $Q_s \in \mathcal{A}$ because every atom of its body mapped by β to fact over Σ^a signature in $\mathcal{A}$. ∎

Foreign keys encoding under enforced semantics

In this section, we are going to present an encoding of TC-QC entailment with FK under enforced semantics. It is an extension of the encoding under ideal semantics. It captures additional constraints imposed over the ideal and available databases.

In the characterization we have seen the only two applications of $\mathcal{K}^i$ and $\mathcal{K}^a$. That is why we only need to encode the application of $\mathcal{K}^i$ to the prototypical database D^i_Q and the application of $\mathcal{K}^a$ to $f_{\mathcal{C}}(D^i_Q)$. In the encoding, we have two signatures Σ^i and Σ^a. We have to apply new constraints to Σ^a, since they correspond to $f_{\mathcal{C}}(D^i_Q)$.

Assume, primary key constraints are always at the beginning of relations and referencing columns are always at the end. Then, for a foreign key $f = R[A] \subseteq S[B]$ we define a rule r_f^a. In the rule, $\bar{B}$ is a tuple of distinct variables in the position of the referencing attributes of R, $\bar{C}$ is a tuple of distinct variables in the remaining positions of R, $\bar{D}$ is a tuple of distinct variables in the remaining positions of S. In S, $\bar{B}$ occurs in the positions of the key attributes. Note, $\bar{A}$ does not explicitly occur in the rules because we join the primary key arguments of S with the referencing arguments of R and denote them using only one vector of variables $\bar{B}$.

$$S^a(\bar{B}, \bar{D}) \leftarrow S(\bar{B}, \bar{D}), R^a(\bar{C}, \bar{B}). \tag{5.19}$$

If we look at the semantics of the rule, we see why it encodes the co-chase procedure $\mathcal{K}^a$. It adds a fact only if the corresponding R-fact is already in the available database and a corresponding S-atom is only in the ideal database. It corresponds to the rule 3 in the procedure 5.2.

We generalize this rule accordingly to the previous rule r_f. We define the program $P_{\mathcal{K}}$ as

$$P_{\mathcal{K}} = P_{\mathcal{K}}^i \cup \{r_f^a \mid f \in \mathcal{K}\}. \tag{5.20}$$

We define a lemma to make a connection between these constraints and the $\mathcal{K}^a$ operator.

Lemma 5.4.3 (Correspondence: $\mathcal{K}^a$ and $P_{\mathcal{K}}$) *Let $\mathcal{D} = (D^i, D^a)$ be a partial database where D^i satisfies $\mathcal{K}$. Then, a fact $S(\bar{t})$ is in $\mathcal{K}^a(\mathcal{D})$ iff $S^a(\bar{t})$ is in the answer set of $P_{\mathcal{K}} \cup D^i \cup D^a$. In this union, we assume D^a to be the set of facts over the Σ^a signature.*

Proof Note that under the acyclicity assumption, $P_{\mathcal{K}}$ is a stratified program with a unique answer set. We denote it as $\mathcal{A}$.

Assume, a fact $S(\bar{t})$ is in $\mathcal{K}^a(\mathcal{D})$ but it is not in $\mathcal{A}$. If the fact is in D^a, then we trivially obtain contradiction. Assume, it is not. Then, $S(\bar{t})$ was added by the rule in Line 3 of Algorithm 5.2. Then, there must be a corresponding by some FK f fact $R(\bar{s})$ in D^a. We know that D^i satisfies FK, then both facts are in D^i as well. Consequently, we obtain contradiction between facts $R^a(\bar{t})$ in D^a, $S(\bar{s})$ in D^i and the rule r_f^a that forces $S^a(\bar{s})$ to present in $\mathcal{A}$.

Assume, a fact $S^a(\bar{t})$ is in $\mathcal{A}$ but $S(\bar{t})$ is not in $\mathcal{K}^a(\mathcal{D})$. If the fact is in D^a, then we trivially obtain contradiction. Assume, it is not. Then, it was added by some rule r_f^a. Then, there is a corresponding atom $R(\bar{s})$ in D^a. Since D^i satisfies $\mathcal{K}$, both $R(\bar{s})$ and $S(\bar{t})$ are in D^i. Then, we obtain contradiction between the rule in Line 3 of Algorithm 5.2 and the fact that $S(\bar{t})$ is not in $\mathcal{K}^a(\mathcal{D})$. ∎

Theorem 5.4.4 (Encoding with FK under Enforced Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements and $\mathcal{K}$ be a set of FK. Then,*

$$\mathcal{C}, \mathcal{K} \models \mathit{Compl}(Q) \iff P_Q \cup P_{\mathcal{C}} \cup P_{\mathcal{K}} \text{ has an answer set.} \tag{5.21}$$

Proof Instead of showing the correspondence between the encoding and the definition of query completeness, we show the correspondence between the encoding and the characterization.

$$Q_s(\mathcal{K}_{\mathcal{C}}(D_Q^i)) \neq \emptyset \iff P_Q \cup P_{\mathcal{C}} \cup P_{\mathcal{K}} \text{ has an answer set.} \tag{5.22}$$

The proof is technical, that is why we briefly explain the main points of the proof. The proof appeals to the characterization presented before. It has five logical levels. Firstly, we establish correspondence between ground facts in the encoding and the prototypical database in the characterization. Secondly, by means of Lemma 5.4.1, we show correspondence between the prototypical database extended by $\mathcal{K}^i$ and the set of facts over the Σ^i signature extended by $P^i_{\mathcal{K}}$ (which is a part of $P_{\mathcal{K}}$, see Definition 5.20). Then, by means of Lemma 3.3.1, we show correspondence between the sets we derive from them, namely between the set of facts over the Σ^a signature and the set $f_{\mathcal{C}}(D^i_Q)$. According to Lemma 5.4.3, we show correspondence between the set of fact over Σ^a signature extended by $P_{\mathcal{K}}$ and the $f_{\mathcal{C}}(D^i_Q)$ set extended by $\mathcal{K}^a$. Finally, by means of Lemma 4.5.2, we show that both queries evaluate to true. The proof follows these steps in both directions.

The procedure $\mathcal{K}^i$ only affects D^i_Q; the $P^i_{\mathcal{K}}$ part of $P_{\mathcal{K}}$ only affects EDB – the set of facts over Σ^i signature. Then, we need to show correspondence between the set of facts in the answer set over the Σ^i signature and the set $\mathcal{K}^i(D^i_Q)$. Let D^i_Q be a database instance over the Σ^i signature, then $\mathcal{K}^i(D^i_Q)$ and the set of facts over the Σ^i signature in the answer set are equal, due to Lemma 5.4.1. It means Lemma 3.3.1 holds for $\mathcal{K}^i(D^i_Q)$ and the set of facts over Σ^i signature in the answer set.

($\Rightarrow$) Assume, P has an answer set $\mathcal{A}$. Then it has Q_s in $\mathcal{A}$ due to filtering rule. Consequently, there is a mapping β such that it maps body (B^a) of Q_s to the facts in $\mathcal{A}$ over Σ^a signature.

The body B of the test query $Q_s()$ is the same set of atoms as the body B^a of the test-rule Q_s except of the upper index $\bullet^a$, according to Definition 3.25. For every atom $R^a(\bar{t}) \in B^a$, there is an atom $R(\bar{t}) \in B$ such that they are the same except of the index. We know from previous paragraph that $\beta R^a(\bar{t}) \in \mathcal{A}$, consequently we obtain that $\beta R(\bar{t}) \in f_{\mathcal{C}}(D^i_Q)$. Due to Lemmata 3.3.1 and 5.4.3, for every fact over the Σ^a signature, there is a corresponding fact in $f_{\mathcal{C}}(D^i_Q)$ over the Σ^i signature. Due to the definition of the test query, $Q_s()$ is a boolean conjunctive query and every atom of the body B is mapped to $f_{\mathcal{C}}(D^i_Q)$ by β. Ergo, it returns the empty tuple.

($\Leftarrow$) Assume, $Q_s(f_{\mathcal{C}}(\mathcal{K}^i(D^i_Q))) \neq \emptyset$. There is a mapping β that maps every atom $R(\bar{t})$ in the body of $Q_s()$ to $f_{\mathcal{C}}(D^i_Q)$. Due to Definition 3.25, for every atom $R(\bar{t})$ in the body of the test query $Q_s()$, there is a corresponding atom $R^a(\bar{t})$ in the body of encoding of the test-rule Q_s. Due to Lemmata 3.3.1 and 5.4.3, for every atom $R(\bar{t}) \in f_{\mathcal{C}}(D^i_Q)$, there is an atom $R^a(\bar{t}) \in \mathcal{A}$. Consequently, for every atom $R^a(\bar{t})$ in the body of the test-rule, it holds that $\beta R^a(\bar{t}) \in \mathcal{A}$. Then, $Q_s \in \mathcal{A}$ because every atom of its body mapped by β to fact over the Σ^a signature in $\mathcal{A}$. ∎

Encoding theorems with foreign keys Having introduced a reduction of the query completeness problem into a problem of an evaluation of an answer set program, we have made a feasible decision procedure for TC-QC reasoning. It can be used in practice provided there is an efficient ASP reasoner available.

It also allows to reason with foreign keys in two modes: under ideals semantics, and under available semantics. If we have only knowledge about the domain, we use the reasoning procedure under ideal semantics. If we know that foreign keys hold on the database as well, we use the reasoning procedure under enforced semantics.

5.5 Both Types of Constraints

Additional constraints on the schema and the database allow for more conclusions. If we impose foreign keys, we can discover new inference possibilities. As well, if we impose finite domain constraints, we can obtain more complentess results. If we combine both types of constraints, we obtain even more new inference possibilities.

We investigate how to combine both approaches we have seen so far. We start with an example indicating how constraints interact with each other. Then, we generalize this example and introduce a characterization with foreign keys and finite domain constraints. At the end, we change the encoding to take into account both types of constraints.

FK and FDC: example under ideal semantics

In this section, we want to develop a procedure that captures reasoning with both types of constraints – finite domain constraints and foreign keys. Let us start with an example illustrating what inferences are possible.

Assume, we work under ideal semantics in this example. We would like to demonstrate an interaction between different types of constraints. To do so, we take an appropriate schema where such an inference can naturally occur. Assume, the schema is

$$\begin{aligned} &\mathit{pupil}(\mathit{Name}, \mathit{Level}, \mathit{Code}).\\ &\mathit{class}(\mathit{Level}, \mathit{Code}, \mathit{Branch}). \end{aligned}$$

Assume, there are two constraints on it. There is a finite domain constraint

$$\mathit{Dom}(\mathit{class}, 2, \{a, b\}).$$

There is a foreign key constraint[6] that holds over D^i

$$\mathit{pupil}(\_, \mathit{Level}, \mathit{Code}) \text{ references } \mathit{class}(\mathit{Level}, \mathit{Code}, \_).$$

There are two TC statements

$$\begin{aligned} &\mathit{Compl}(\mathit{pupil}(N, 1, a); \top).\\ &\mathit{Compl}(\mathit{pupil}(N, 1, b); \top). \end{aligned}$$

There is a query Q

$$Q(N) \leftarrow \mathit{pupil}(N, 1, C).$$

In this example, we are going to follow the same pattern as in the 3rd chapter. We also show interaction between different constraints and parts of the reasoning procedure.

[6] For the foreign key syntax, see Definition 2.4.4

1. Determine what tuples have to be in D^i Assume, Q returns a constant *name* as an answer tuple. Then, there must be a corresponding body atom in D^i because Q is a conjunctive query

$$\textit{pupil}(\textit{name}, 1, \textit{code}) \in D^i. \tag{5.23}$$

Due to the foreign key constraint, for every *pupil*-atom, there must be a corresponding *class*-atom. For the atom $\textit{pupil}(\textit{name}, \textit{level}, \textit{code})$, there must be an atom $\textit{class}(\textit{level}, \textit{code}, \textit{branch})$. Then, we conclude that

$$\textit{class}(1, \textit{code}, \textit{branch}) \in D^i. \tag{5.24}$$

2. Determine what tuples have to be in D^a Due to the finite domain constraint, *code* in the atom $\textit{class}(1, \textit{code}, \textit{branch})$ can be only a or b. Assume, it is a. Then, we apply the TC statement $\textit{Compl}(\textit{pupil}(N, 1, a); \top)$ to the atom $\textit{pupil}(\textit{name}, 1, a)$. As a result, we establish this atom to be in the available database

$$\textit{pupil}(\textit{name}, 1, a) \in D^a. \tag{5.25}$$

3. Verify completeness Assuming *code* to be a, the query Q returns the constant *name* over both databases D^i and D^a. In this case, the query is complete.

If we assume *code* to be b, then the result stays the same due to the symmetry of the rules.

FK and FDC: example under enforced semantics

There are two possible semantics for foreign keys in the context of completeness reasoning. We have already shown an example with finite domain and foreign key constraints under ideal semantics. In this subsection, we present an example of reasoning with finite domains and foreign keys under enforced semantics. We would like to demonstrate additional possibilities to due establish completeness due to interaction between these two types of constraints in the reasoning procedure.

Assume, the schema, the TC statements, the finite domain constraint and the foreign key constraint are the same as in the previous example. The query Q is

$$Q(N) \leftarrow \textit{pupil}(N, 1, C), \textit{class}(1, C, B).$$

The difference with the previous example is also in semantics of foreign keys. In this example, we assume enforced semantics which allows to reason over the available database by means of foreign keys. It gives additional possibilities to establish query completeness.

1. Determine what tuples have to be in D^i Assume, Q returns a constant *name* as an answer tuple. Then, there must be corresponding body atoms in D^i because Q is a conjunctive query

$$\textit{pupil}(\textit{name}, 1, \textit{code}) \in D^i.$$
$$\textit{class}(1, \textit{code}, \textit{branch}) \in D^i.$$

For every *pupil*-atom, there is a corresponding *class*-atom. For the atom $\textit{pupil}(\textit{name}, 1, \textit{code})$, there is the atom $\textit{class}(1, \textit{code}, \textit{branch})$. Then, the foreign key constraint over D^i is satisfied and we cannot infer any new facts in D^i.

2. Determine what tuples have to be in D^a Due to the finite domain constraint, the constant *code* in the atom $\mathit{class}(1, \mathit{code}, \mathit{branch})$ can be only a or b. Assume, it is a. Then, we apply the TC statement $\mathit{Compl}(\mathit{pupil}(N, 1, a); \top)$ to the atom $\mathit{pupil}(\mathit{name}, 1, a)$. As a result, we establish that this atom is in the available database

$$\mathit{pupil}(\mathit{name}, 1, a) \in D^a.$$

Due to the foreign key constraint over D^a, for every *pupil*-atom in the available database, there is a corresponding *class*-atom in the available database. For the atom $\mathit{pupil}(\mathit{name}, 1, \mathit{code})$, there must be the corresponding atom $\mathit{class}(1, \mathit{code}, \mathit{branch})$. As a result, we establish that this atom is in the available database.

$$\mathit{class}(1, \mathit{code}, \mathit{branch}) \in D^a.$$

3. Verify completeness Assuming *code* to be a, the query Q returns the constant *name* over both databases D^i and D^a. In this case, the query is complete.

If we assume *code* to be b, then the result stays the same due to the symmetry of the rules.

FK and FDC: semantics

Query completeness with foreign keys restricts models to satisfying foreign keys. Query completeness with finite domain constraints restricts models to satisfying finite domain constraints. Defining query completeness with both types of constraints, it is natural to restrict model to satisfying both of them.

We have introduced two semantics for foreign keys in the context of completeness reasoning. That is why we have two possible semantics for reasoning with both types of constraints. When we talk about foreign keys imposed only over the ideal database we explicitly mention ideal semantics in the titles. When we talk about foreign keys imposed over both databases, we normally do not mention semantics in the titles.

Let us formally introduce TC-QC entailment in the presence of both types of constraints. We start with foreign keys under ideal semantics.

Definition 5.5.1 (TC-QC Entailment with FK and FDC under Ideal Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{F}$ be a set of FDC and $\mathcal{K}$ be a set of FK. Then, $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{K}$ entail completeness of Q under ideal semantics written $\mathcal{C}, \mathcal{F}, \mathcal{K} \models^i \mathit{Compl}(Q)$ iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \text{ and } \mathcal{D} \models \mathcal{F} \text{ and } D^i \models \mathcal{K} \implies Q(D^i) = Q(D^a).$$

In the similar way we introduce TC-QC entailment with FK and FDC under enforced semantics.

Definition 5.5.2 (TC-QC Entailment with FK and FDC) *Let Q be a query, $\mathcal{C}$ be a set of TC-statements, $\mathcal{F}$ be a set of FDC and $\mathcal{K}$ be a set of FK. Then, $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{K}$ entail completeness of Q written $\mathcal{C}, \mathcal{F}, \mathcal{K} \models \mathit{Compl}(Q)$ iff for any partial database $\mathcal{D} = (D^i, D^a)$ it holds that*

$$\mathcal{D} \models \mathcal{C} \text{ and } \mathcal{D} \models \mathcal{F} \text{ and } \mathcal{D} \models \mathcal{K} \implies Q(D^i) = Q(D^a).$$

Characterization of FK and FDC

We have described reasoning with finite domain constraints separately from reasoning with foreign key constraints. In this section, we combine both types of constraints into the one characterization.

The difference between relational reasoning and reasoning with foreign keys is in application of the $\mathcal{K}^i$ and $\mathcal{K}^a$ operators. The difference between relational reasoning and reasoning with finite domain constraints is in application of cases (mappings of bound variables) to the query and the prototypical database. Finite domain and foreign key constraints apply to the different parts of the reasoning problem independently. However, as we have seen in the examples they do interact with each other. Namely, new facts added by the chase procedure $\mathcal{K}^i$ can trigger some finite domain constraints. That is why we have to extend the definition of a bound variable. Also, we have seen in the examples with fresh constants that these constants can trigger finite domain constraints, therefore, now we have to consider bound terms.

Definition 5.5.3 (Bound Term) *A term t is bound by a finite domain constraint $f = Dom(R, i, M)$ iff t occurs in the i-th position of an atom with relation R in $\mathcal{K}^i(D^i_Q)$.*

It implies that the atoms added because of foreign keys in the schema, can trigger finite domain constraints.

It also implies that the set of cases $\Gamma_{\mathcal{F},\mathcal{Q}}$ depends on the set of foreign keys $\mathcal{K}$. Let us denote this dependence of the set of cases as $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$. Each γ in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$ is a mapping of bound terms to a set of constant such that γ is consistent with the set of finite domain constants $\mathcal{F}$. Let us formally introduce $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$.

Definition 5.5.4 *Let Q be a query, $\mathcal{K}$ be a set of foreign keys, $\mathcal{F}$ be a set of finite domain constraints. Then, $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$ is the set of all mappings from the set of bound terms in Q to a set of constants such that each mapping is consistent with $\mathcal{F}$.*

However, what we need to change is application of γ to the set $\mathcal{K}^i(D^i_Q)$ instead of D^i_Q. We need to move γ out of the γD^i_Q. We have seen in the example the order of operator applications in the reasoning. Firstly, we compute $\mathcal{K}^i(D^i_Q)$, then we apply γ to it. Afterwards, we apply the function $f_\mathcal{C}$ to the set $\gamma\mathcal{K}^i(D^i_Q)$ and evaluate the test query over the result of $f_\mathcal{C}$ application.

We can see all these steps clearly in the example in the previous chapter. At the beginning, we computed D^i_Q in 5.23. Then, we applied the foreign key constraint to it 5.24. Afterwards, at the beginning of the second paragraph of Section 5.5 we applied the finite domain constraint to $\mathcal{K}^i(D^i_Q)$. Finally, in 5.25, we applied the TC statement to $\gamma\mathcal{K}^i(D^i_Q)$ and verified completeness in the third paragraph of Section 5.5.

Let us motivate the characterization with both types of constraints. As in the previous chapters, we introduce an effective check of TC-QC entailment. It explicitly shows interaction between different parts of the problem and it can be effectively computed. It is an important step towards development of an algorithm or a decision procedure that can reason with different types of constraints. In this case, the types are finite domain and foreign key constraints.

Theorem 5.5.1 (TC-QC Characterization with FDC and FK under Ideal Semantics) *Let Q be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{F}$ be a set of FDC, $\mathcal{K}$ be a set of FK. Then,*

$$\mathcal{C}, \mathcal{F}, \mathcal{K} \models^i \mathit{Compl}(Q) \iff Q_s^\gamma(f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i))) \text{ for every } \gamma \text{ in } \Gamma_{\mathcal{F},\mathcal{K},Q}.$$

Proof In the following, let θ be the freezing assignment of the variables in Q, B be the body of Q, $\bar{X}$ be the vector of distinguished variables of Q.

($\Rightarrow$) Assume that $\mathcal{C}, \mathcal{F}, \mathcal{K} \models^i \mathit{Compl}(Q)$. Consider a partial database $\mathcal{D} = (D^i, D^a)$ that satisfies $\mathcal{C}, \mathcal{F}$ and $D^i \models \mathcal{K}$. Since Q is complete, for any tuple $\bar{c}$ it holds that if $\bar{c}$ is in the answer to $Q(D^i)$, then it is in the answer to $\bar{c}$ in $Q(D^a)$. Let us take tuple $\bar{t} = \gamma\theta\bar{X}$. There must be a corresponding to $\bar{t}$ assignment α such that $\alpha B \subseteq D^i$. By assumption $\mathcal{D} \models \mathcal{F}$, Representation Lemma 4.3.3 applies to γ. Since $\bar{t}$ is the frozen tuple, α must be of the form $\alpha'\gamma\theta$, for some fixed γ in $\Gamma_{\mathcal{F},\mathcal{K},Q}$. Then, it holds that $\alpha'\gamma\theta$ maps B to D^i. By definition, D_Q^i is frozen by θ version of B. We conclude that $\alpha'\gamma$ maps D_Q^i to D^i. By definition $\mathcal{K}^i$ is a monotone operator, then $\alpha'\gamma$ maps $\mathcal{K}^i(D_Q^i)$ to $\mathcal{K}^i(D^i)$. Since $D^i \models \mathcal{K}$, we conclude that the assignment $\alpha'\gamma$ maps $\mathcal{K}^i(D_Q^i)$ to D^i. Since $\mathcal{D} \models \mathcal{C}$, it holds that $f_\mathcal{C}(D^i) \subseteq D^a$. Finally, we conclude that α' maps $f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i))$ to D^a. For the fixed assignment γ, it holds that $Q_s^\gamma(f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i))) \neq \emptyset$

($\Leftarrow$) Now we assume that $Q_s^\gamma(f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i))) \neq \emptyset$ for all γ in $\Gamma_{\mathcal{F},\mathcal{K},Q}$ and we want to show that $\mathcal{C}, \mathcal{F}, \mathcal{K} \models^i \mathit{Compl}^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database that satisfies $\mathcal{C}, \mathcal{F}$ and $D^i \models \mathcal{K}$. We assume that $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Assume there is α that maps Q to D^i such that $\bar{c} = \alpha\bar{X}$. Due to Representation Lemma 4.3.3, α must be of the form $\alpha'\gamma$. Then, it holds that $\alpha'\gamma B \subseteq D^i$. Since $D^i \models \mathcal{K}$ we conclude that $\mathcal{K}^i(\alpha'\gamma B) \subseteq D^i$. Since $\mathcal{D} \models \mathcal{C}$, we conclude that $f_\mathcal{C}(\mathcal{K}^i(\alpha'\gamma B)) \subseteq D^a$.

Let β be an assignments such that $Q_s^\gamma(f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i)))$ evaluates to true for every γ in $\Gamma_{\mathcal{F},\mathcal{K},Q}$. Due to Representation Lemma 4.3.3 and the fact that distinguished variables of Q_s^γ are frozen by θ, β must be of the form $\theta\gamma\beta'$, where β' maps only non-distinguished variables. Consequently, the tuple $\bar{t} = \theta\gamma\bar{X}$ must be in the answer to Q over $f_\mathcal{C}(\gamma\mathcal{K}^i(D_Q^i))$. Let us apply $\alpha'\theta^{-1}$ to $\bar{t}$. As result we obtain, $\bar{t} = \alpha'\gamma\bar{X}$, i.e. we obtain c. Then, c is an answer tuple to Q over $\alpha'\theta^{-1}f_\mathcal{C}(\gamma\mathcal{K}^i(\theta B))$.

Then, the composition $\alpha'\theta^{-1}$ maps $f_\mathcal{C}(\gamma\mathcal{K}^i(\theta B))$ to $f_\mathcal{C}(\mathcal{K}^i(\alpha'\gamma B))$, since Q_γ^s evaluated to true for any γ in $\Gamma_{\mathcal{F},\mathcal{K},Q}$ and both θ and θ^{-1} preserve distinguishness of variables. Due to the fact that $f_\mathcal{C}(\mathcal{K}^i(\alpha'\gamma B)) \subseteq D^a$, we conclude that c is in $Q(D^a)$.

It seems like the finite domain constraints are applied first in this definition. However, due to Definition 5.5.3, it is not the case. Let us illustrate it with an example.

Assume, the schema is the same as in Example 5.5. The query Q is

$$Q(N) \leftarrow \mathit{pupil}(N, L, C).$$

There is a foreign key constraint[7]

$$\mathit{pupil}(\_, L, C) \text{ references } \mathit{class}(L, C, \_). \tag{5.26}$$

[7]For the foreign key syntax, see Definition 2.4.4

There is a finite domain constraint[8]

$$\mathit{Dom}(\mathit{class}, 3, \{a, b\}) \tag{5.27}$$

There are two TC statements

$$\mathit{Compl}(\mathit{pupil}(N, L, C); \mathit{class}(L, C, a)). \tag{5.28}$$

$$\mathit{Compl}(\mathit{pupil}(N, L, C); \mathit{class}(L, C, b)). \tag{5.29}$$

Let us present calculations and an application of Theorem 5.5.1.

At the beginning we compute $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$. We start with Definition 5.5.3 that shows how to compute bound terms. According to Definition 5.5.3, we need to compute $\mathcal{K}^i(D^i_Q)$ to establish bound terms. In our case D^i_Q is $\{\mathit{pupil}(n, l, c)\}$. Then, due to FK 5.26, $\mathcal{K}^i(D^i_Q)$ is $\{\mathit{pupil}(n, l, c), \mathit{class}(l, c, f_{\mathit{class}}(l, c))\}$. Let us refer to the Skolem term $f_{\mathit{class}}(l, c)$ as f. Then, f is the only bound term. Due to FDC 5.27, f can be either a or b. Finally, there are two mappings in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$: $\gamma_1 = \{f \mapsto a\}$ and $\gamma_2 = \{f \mapsto b\}$.

Let us show all consequent computations for γ_1 (they are symmetric for γ_2). To compute γD^i_Q, we apply γ_1 to D^i_Q, as a result we obtain $\gamma D^i_Q = \{\mathit{pupil}(n, l, c), \mathit{class}(l, c, a)\}$. Then, we compute $f_{\mathcal{C}}(\gamma D^i_Q)$, by applying TC Statements 5.28, as a result, $f_{\mathcal{C}}(\gamma D^i_Q)$ is equal to $\{\mathit{pupil}(n, c, l)\}$. Finally, there is a mapping $\alpha = \{N \mapsto n, L \mapsto k, C \mapsto c\}$ such that the body $\mathit{pupil}(N, L, C)$ of the test query is satisfied by α over the part of the available database $f_{\mathcal{C}}(\gamma D^i_Q)$. According to Theorem 5.5.1, the query is complete (due to the symmetry of TC statements the calculations are the same for γ_2).

In the same manner we introduce a characterization under enforced semantics. It captures reasoning with both types of constraints when foreign keys are also imposed over the available database.

However, we have to make a change in our definition of $\mathcal{K}_{\mathcal{C}}$ operator. We need to introduce γ into it. Let us introduce a new version of this operator formally.

Definition 5.5.5 (The operator $\mathcal{K}^{\gamma}_{\mathcal{C}}$) *Let $\mathcal{K}$ be a set of foreign keys, $\mathcal{C}$ be a set of TC statements, γ be a case in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$. Then, we define the operator $\mathcal{K}^{\gamma}_{\mathcal{C}}$ over a database instance D by the following equation:*

$$\mathcal{K}^{\gamma}_{\mathcal{C}}(D) = \mathcal{K}^a(\gamma\mathcal{K}^i(D), f_{\mathcal{C}}(\gamma\mathcal{K}^i(D))). \tag{5.30}$$

Theorem 5.5.2 (TC-QC Characterization with FDC and FK) *Let $\mathcal{Q}$ be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{F}$ be a set of FDC, $\mathcal{K}$ be a set of FK. Then,*

$$\mathcal{C}, \mathcal{F}, \mathcal{K} \models \mathit{Compl}(Q) \iff Q^{\gamma}_s(\mathcal{K}^{\gamma}_{\mathcal{C}}(D^i_Q)) \text{ for every } \gamma \text{ in } \Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}.$$

Proof In the following, let θ be the freezing assignment of the variables in Q, B be the body of Q, $\bar{X}$ be the vector of distinguished variables of Q.

[8]For the finite domain constraint syntax, see Section 2.4

($\Rightarrow$) Assume that $\mathcal{C}, \mathcal{F}, \mathcal{K} \models \mathit{Compl}(Q)$. Consider a partial database $\mathcal{D} = (D^i, D^a)$ that satisfies $\mathcal{C}, \mathcal{F}, \mathcal{K}$. Since $\mathcal{D} \models \mathcal{C}$, for any tuple $\bar{c}$ it holds that if $\bar{c}$ is in the answer to $Q(D^i)$, then it is in the answer to $Q(D^a)$. Let us take the tuple $\bar{t} = \gamma\theta\bar{X}$. There must be a corresponding to $\bar{t}$ assignment α such that $\alpha B \subseteq D^i$. By assumption $\mathcal{D} \models \mathcal{F}$, Representation Lemma 4.3.3 applies to γ. Since t is the frozen tuple, α must be of the form $\alpha'\gamma\theta$, for some fixed γ in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$. Then, it holds that $\alpha'\gamma\theta$ maps B to D^i. By definition, D^i_Q is frozen by θ version of B. We conclude that $\alpha'\gamma$ maps D^i_Q to D^i. By definition $\mathcal{K}^i$ is a monotone operator, then $\alpha'\gamma$ maps $\mathcal{K}^i(D^i_Q)$ to $\mathcal{K}^i(D^i)$. Since $D^i \models \mathcal{K}$, we conclude that the assignment $\alpha'\gamma$ maps $\mathcal{K}^i(D^i_Q)$ to D^i. Since $\mathcal{D} \models \mathcal{C}$, it holds that $f_{\mathcal{C}}(D^i) \subseteq D^a$. Then, we conclude that α' maps $f_{\mathcal{C}}(\gamma\mathcal{K}^i(D^i_Q))$ to D^a. Finally, since $D^a \models \mathcal{K}$, α' maps $\mathcal{K}^\gamma_{\mathcal{C}}(D^i_Q)$ to D^a. For the fixed assignment γ, it holds that $Q^\gamma_s(\mathcal{K}^\gamma_{\mathcal{C}}(D^i_Q)) \neq \emptyset$

($\Leftarrow$) Now we assume that $Q^\gamma_s(\mathcal{K}^\gamma_{\mathcal{C}}(D^i_Q))) \neq \emptyset$ for all γ in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$ and we want to show that $\mathcal{C}, \mathcal{F}, \mathcal{K} \models \mathit{Compl}^s(Q)$. In order to do so we assume that $\mathcal{D} = (D^i, D^a)$ is a partial database that satisfies $\mathcal{C}, \mathcal{F}, \mathcal{K}$. We assume that $\bar{c}$ is an arbitrary tuple in $Q(D^i)$. We have to show that $\bar{c}$ is in $Q(D^a)$ as well.

Assume there is α that maps Q to D^i such that $\bar{c} = \alpha\bar{X}$. Due to Representation Lemma 4.3.3, α must be of the form $\alpha'\gamma$. Then, it holds that $\alpha'\gamma B \subseteq D^i$. Since $D^i \models \mathcal{K}$ we conclude that $\mathcal{K}^i(\alpha'\gamma B) \subseteq D^i$. Since $\mathcal{D} \models \mathcal{C}$, we conclude that $f_{\mathcal{C}}(\mathcal{K}^i(\alpha'\gamma B)) \subseteq D^a$. Since, $D^a \models \mathcal{K}$, $\mathcal{K}^\gamma_{\mathcal{C}}(D^i_Q) \subseteq D^a$

Let β be an assignments such that $Q^\gamma_s(\mathcal{K}^\gamma_{\mathcal{C}}(D^i_Q))$ evaluates to true for every γ in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$. Due to Representation Lemma 4.3.3 and the fact that distinguished variables of Q^γ_s are frozen by θ, β must be of the form $\theta\gamma\beta'$, where β' maps only non-distinguished variables. Consequently, the tuple $\bar{t} = \theta\gamma X$ must be in the answer to Q over $f_{\mathcal{C}}(\gamma\mathcal{K}^i(D^i_Q))$. Let us apply $\alpha'\theta^{-1}$ to $\bar{t}$. As result we obtain, $\bar{t} = \alpha'\gamma\bar{X}$, i.e. we obtain c. Then, c is an answer tuple to Q over $\alpha'\theta^{-1}\mathcal{K}^\gamma_{\mathcal{C}}(\theta B)$.

Then, the composition $\alpha'\theta^{-1}$ maps $\mathcal{K}^\gamma_{\mathcal{C}}(\theta B)$ to $\mathcal{K}^\gamma_{\mathcal{C}}(\alpha'\gamma B)$, since Q^s_γ evaluated to true for any γ in $\Gamma_{\mathcal{F},\mathcal{K},\mathcal{Q}}$ and both θ and θ^{-1} preserve distinguishness of variables. Due to the fact that $\mathcal{K}^\gamma_{\mathcal{C}}(\alpha'\gamma B) \subseteq D^a$, we conclude that c is in $Q(D^a)$.

Introduction of the characterization with finite domains and foreign keys allows us to design a program that checks TC-QC entailment in the presence of both types of constraints. We make an argument that the program we create has the same semantics as the characterization we have presented. That is why it is important to introduce the characterization as a specification that regulates how different parts must interact and how the reasoning procedure should work.

Encoding of FK and FDC

In this section, we introduce an encoding of the completeness reasoning problem in the presence of foreign keys and finite domain contains. We are going to follow the same strategy as before. We analyze the changes in the characterization. We modify the encoding accordingly.

Changes in the rules We start with the encoding under ideal semantics of foreign keys. In the example, we have seen that a finite domain constraint can be triggered by a fresh constant introduced by a foreign key constraint. It naturally follows that now all terms in $\mathcal{K}^i(D^i_Q)$ must

have values. By default, every frozen variable has a value that is equal to itself. Then, we add a rule for every foreign key that generates a default value for every added atom. Let r be a pair of rules associated with a foreign key $f = R[A] \subseteq S[B]$ in $\mathcal{K}$, then two rules in r are the following [9]

$$S(\bar{B}, f_S(\bar{B})) \leftarrow R(\bar{C}, \bar{B}),\ \text{not aux}S(\bar{B}). \tag{5.31}$$
$$\text{aux}S(\bar{B}) \leftarrow S(\bar{B}, \bar{D}),\ \textit{original}(\bar{D}). \tag{5.32}$$

We have also observed that $P_{\mathcal{C}}$ has changed by means of unfolding of the variables and the constants in the rules. The rules in $P_{\mathcal{K}}$ are similar to the rules in $P_{\mathcal{C}}$. Then, we modify the program $P_{\mathcal{K}}$ by unfolding the rules as we have done in Section 4.5. We modify the rules, according to Unfolding 4.50, by application of Rule 4.44 to every atom in the body of the rules.

Let us note here that the aux-predicate is meant to verify that the fact is already in the database. Then, it means we have to pass to the aux-predicate *values* of the variables instead of names of the variables. Let r_1^u and r_2^u be the unfolded version of a pair of rules r_1 and r_2. Having unfolded the rules, we change the head of the aux-rule and obtain.

$$S(\bar{B}, f_S(\bar{V}_B)) \leftarrow R(\bar{C}, \bar{B}),\ \text{not aux}S(\bar{V}_B), \textit{val}(\bar{B}, \bar{V}_B). \tag{5.33}$$
$$\text{aux}S(\bar{V}_B) \leftarrow S(\bar{B}, \bar{D}),\ \textit{original}(\bar{D}),\ \textit{val}(\bar{B}, \bar{V}_B). \tag{5.34}$$

Let us denote as r_{val} a rule corresponding to r that generates values for fresh constants

$$\textit{val}(f_S(\bar{V}_B), f_S(\bar{V}_B)) \leftarrow R(\bar{C}, \bar{B}),\ \text{not aux}S(\bar{V}_B),\ \textit{val}(\bar{B}, \bar{V}_B). \tag{5.35}$$

The rule generates values for fresh constants correctly for the following reasons, if the body of Rule 5.31 is satisfied, then the head is added to the answer set. It means under the same condition we must associate a value with it. Then, the body of Value Generation Rule 5.35 must be exactly the same as the body of Rule 5.31 that adds atoms. Let us observe that we use values of variables in the Skolem terms instead of names.

Let r_{val} be the *val*-generation rule for r, then we define the unfolded program $P^u_{\mathcal{K}^i}$ as

$$P^u_{\mathcal{K}^i} = \bigcup_{r \in P^i_{\mathcal{K}}} \{r_1^u, r_2^u, r_{val}\}.$$

In the encoding enforced semantics, we need to set up values in the same way. We also need to unfold the rules, but in this case we need to unfold three rules associated with a foreign key constraint.

Let r_1^u, r_2^u, r_a^u be the unfolded version of the triple of rules r. Let r_{val} be the *val*-generation rule for r. Then, we define the unfolded program $P^u_{\mathcal{K}}$ under enforced semantics as

$$P^u_{\mathcal{K}} = \bigcup_{r \in P_{\mathcal{K}}} \{r_1^u, r_2^u, r_a^u, r_{val}\}.$$

[9] for details about the rules' definition, see 5.17

Encoding Theorem

To automate the work of the decision procedure, we reduce the characterization to an evaluation of an answer set program. It allows us to check TC-QC entailment by using existing ASP reasoning engines. Note, this solutions is optimal from a complexity point of view. The complexity of the reduced problem is the same as the complexity of the problem we reduce into.

Let us introduce an encoding theorem that works with both types of constraints under ideal semantics.

Theorem 5.5.3 *Let Q be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{K}$ be a set of FK, $\mathcal{F}$ be a set of FDC, then*

$$\mathcal{C}, \mathcal{F}, \mathcal{K} \models^i \textit{Compl}(Q) \iff P^u_Q \cup P^u_{\mathcal{C}} \cup P^u_{\mathcal{K}^i} \cup P_{\mathcal{F}} \textit{ has } Q \textit{ in every answer set}.$$

Proof The proof is technical, that is why we briefly explain the main points of the proof. The proof appeals to the characterization presented before. It has five logical levels. Firstly, we establish correspondence between ground facts in the encoding and the prototypical database in the characterization. Secondly, by means of Lemma 4.5.1, we show correspondence between a considered case γ and the *val*-set in the encoding. Thirdly, by means of Lemma 5.4.1, we show correspondence between the prototypical database extended by $\mathcal{K}^i$ and the set of facts over the Σ^i signature extended by $P^i_{\mathcal{K}}$. Then, we show correspondence between the sets we derive from the prototypical database extended by $\mathcal{K}^i$ and facts over the Σ^i signature, namely between the set of facts over the Σ^a signature and $f_{\mathcal{C}}(D^i_Q)$, by means of Lemma 3.3.1. Finally, by means of Lemma 4.5.2, we show that both queries evaluate to true. The proof follows these steps in both directions.

Let $\mathcal{A}$ be an answer set of the program.

Assume, $Q^\gamma_s(f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q))) \neq \emptyset$ for an arbitrary but fixed γ. Then, due to Lemma 4.5.1, the set $\textit{val}(\gamma)$ must be in $\mathcal{A}$. We know that D^u_Q must be in EDB as a set of ground facts over Σ^i signature. Since Lemma 5.4.1 applies, the set of facts over Σ^i signature is equal to $\mathcal{K}^i(\gamma D^i_Q)$. Due to Lemma 3.3.1, for every fact $R(\bar{t})$ in $f_{\mathcal{C}}(\gamma D^i_Q)$, there is a fact $R^a(\bar{t})$ in $\mathcal{A}$. Then, we apply Lemma 4.5.2 to the test-rule Q^u. We took γ to be arbitrary but fixed; then, the proof holds for any γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$.

Assume, Q holds in every answer set of the program. We take an arbitrary but fixed answer set $\mathcal{A}$. Due to Lemma 4.5.1, there must be a corresponding γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Then, the set of facts in EDB is $\mathcal{K}^i(\gamma D^i_Q)$ because of Lemma 5.4.1. According to Lemma 3.3.1, the correspondence holds between the facts over Σ^a signature and the facts in $f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q))$, we deduce that for every fact $R^a(\bar{t})$ in $\mathcal{A}$, there is a fact $R(\bar{t})$ in $f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q))$. Due to Lemma 4.5.2, we conclude that the test query Q^γ_s evaluates to true. ∎

Having modified the foreign key encoding part under enforced part, we introduce an encoding theorem that captures reasoning with both types of constraints.

Theorem 5.5.4 *Let Q be a query, $\mathcal{C}$ be a set of TC statements, $\mathcal{K}$ be a set of FK, $\mathcal{F}$ be a set of FDC, then*

$$\mathcal{C}, \mathcal{F}, \mathcal{K} \models \mathit{Compl}(Q) \iff P^u_Q \cup P^u_{\mathcal{C}} \cup P^u_{\mathcal{K}} \cup P_{\mathcal{F}} \text{ has } Q \text{ in every answer set.}$$

Proof The proof is technical, that is why we briefly explain the main points of the proof. The proof appeals to the characterization presented before. It has six logical levels. Firstly, we establish correspondence between ground facts in the encoding and the prototypical database in the characterization. Secondly, by means of Lemma 4.5.1, we show correspondence between a considered case γ and the *val*-set in the encoding. Thirdly, by means of Lemma 5.4.1, we show correspondence between the prototypical database extended by $\mathcal{K}^i$ and the set of facts over the Σ^i signature extended by $P^i_{\mathcal{K}}$ (which is a part of $P_{\mathcal{K}}$, see Definition 5.20). Then, by means of Lemma 3.3.1, we show correspondence between the sets we derive from them, namely between the set of facts over the Σ^a signature and the set $f_{\mathcal{C}}(D^i_Q)$. According to the lemma 5.4.3, we show correspondence between the set of fact over Σ^a signature extended by $P_{\mathcal{K}}$ and the $f_{\mathcal{C}}(D^i_Q)$ set extended by $\mathcal{K}^a$. Finally, by means of the lemma 4.5.2, we show that both queries evaluate to true. The proof follows these steps in both directions.

Let $\mathcal{A}$ be an answer set of the program.

Assume, $Q^\gamma_s(\mathcal{K}^a(f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q)))) \neq \emptyset$ for an arbitrary but fixed γ. Then, due to Lemma 4.5.1, the set $\mathit{val}(\gamma)$ must be in $\mathcal{A}$. We know that D^u_Q must be in EDB as a set of ground facts over Σ^i signature. Since Lemma 5.4.1 applies, the set of facts over Σ^i signature is equal to $\mathcal{K}^i(\gamma D^i_Q)$. Due to Lemmata 3.3.1 and 5.4.3, for every fact $R(\bar{t})$ in $f_{\mathcal{C}}(\gamma D^i_Q)$, there is a fact $R^a(\bar{t})$ in $\mathcal{A}$. Then, we apply Lemma 4.5.2 to the test-rule Q^u. We took γ to be arbitrary but fixed; then, the proof holds for any γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$.

Assume, Q holds in every answer set of the program. We take an arbitrary but fixed answer set $\mathcal{A}$. Due to Lemma 4.5.1, there must be a corresponding γ in $\Gamma_{\mathcal{F},\mathcal{Q}}$. Then, the set of facts in EDB is $\mathcal{K}^i(\gamma D^i_Q)$ because of Lemma 5.4.1. According to Lemmata 3.3.1 and 5.4.3 the correspondence holds between the facts over Σ^a signature and the facts in $\mathcal{K}^a(f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q)))$, we deduce that for every fact $R^a(\bar{t})$ in $\mathcal{A}$, there is a fact $R(\bar{t})$ in $\mathcal{K}^a(f_{\mathcal{C}}(\mathcal{K}^i(\gamma D^i_Q)))$. Due to Lemma 4.5.2, we conclude that the test query Q^γ_s evaluates to true. ∎

Having introduced the encoding theorem, we can design an effective (from a complexity point of view) reasoner for the TC-QC completeness problem in the presence of finite domain and foreign key constraints. By design, this procedure follows the characterization that exactly captures query completeness reasoning. It is an important theorem in development of a practical completeness reasoner.

Having connected two formalisms, we have made a feasible decision procedure for TC-QC reasoning. It can be used in practice provided there is an efficient ASP reasoner available.

CHAPTER 6

Reasoning with Built-in Predicates

Overview

In this chapter, we propose an approach to completeness reasoning with built-in predicates. We start with examples showing how the reasoning procedure is affected by the presence of built-in predicates. We discuss a way to connect reasoning in the presence of built-in predicates with reasoning in the presence of finite domain predicates.

6.1 Example

Let us start with an example illustrating how the reasoning procedure works.

Assume, we have a query Q asking for all employee born after the year 1975

$$Q(\mathit{Name}) \leftarrow \mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}), \mathit{year}(\mathit{Birthday}) > 1975.$$

where the predicate $>$ has the standard semantics. Assume, we have a TC statement

$$\mathit{Compl}(\mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}); \mathit{year}(\mathit{Birthday}) > 1972). \tag{6.1}$$

Clearly, the query is complete. How would we reason in this particular case?

Assume there *name* is in the answer to Q. Then, there is a corresponding body atom

$$\mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}) \in D^i.$$

such that $\mathit{year}(\mathit{bday})$ is greater than 1975. Applying TC Statement 6.1, we obtain the atom

$$\mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}) \in D^a.$$

Consequently, we deduce that the query is complete. Here, we virtually stored the *value* of *bday* and evaluated the built-in predicate taking into account the value of *bday* in the application of the TC statement.

Let us illustrate another possibility to ensure completeness in the presence of the built-in predicates. Assume, we have a query Q_1

$$Q_1(\mathit{Name}) \leftarrow \mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}).$$

Assume, there are two TC statements

$$\mathit{Compl}(\mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}); \mathit{year}(\mathit{Birthday}) \leq 1974).$$
$$\mathit{Compl}(\mathit{Employee}(\mathit{Name}, \mathit{DeptName}, \mathit{Birthday}); \mathit{year}(\mathit{Birthday}) > 1974).$$

If we assume that there is a constant *name* in the answer to Q_1. Then, there must be a corresponding body atom in D^i

$$\mathit{Employee}(\mathit{name}, \mathit{deptName}, \mathit{bday}) \in D^i.$$

The value of *bday* is unrestricted in this case. If we apply the TC statements syntactically, then none of them apply and we cannot ensure completeness. However, we know that the year value of *bday* is either less than, equal to or greater than 1974. Assume, it is less than 1974. The first rules applies and completeness is ensured. In the same way, we establish completeness in the two other cases. As a result, we deduce completeness in general.

We have already seen this reasoning technique – case analysis. As we see later, we are going to propose it as the main technique to reason in the presence of built-in predicates.

6.2 Formalization

Queries with comparisons

In this section, we give a formalization of the entailment under comparisons.

First of all, we start with syntax and semantics of comparison atoms. Every comparison atom has the form

$$X \circ a,$$

where X is a variable, a is a positive integer, and $\circ \in \{<, \leq, >, \geq\}$. The semantics of $\circ$ is defined as classical comparison semantics over integers. We have made a restriction on the comparisons by restraining the form of the atoms.

Let B be a set of relational atoms, M be a set of built-in atoms, $\bar{X}$ be a list of variables, then we say that a *query with comparisons* Q is an expression of the form

$$Q(\bar{X}) \leftarrow B, M. \tag{6.2}$$

Accordingly, we change the definition of an answer tuple

Definition 6.2.1 (Answer Tuple) *A tuple $\bar{c}$ is an answer tuple to a query $Q(\bar{X}) = B, M$ over a database D iff there is a mapping α s.t.*

- $\bar{c} = \alpha\bar{X}$;

- $D, \alpha \models B$;
- $D, \alpha \models M$.

We defined TC statements by means of an associated query. Having defined queries with comparisons, we introduce TC statements with comparisons. Let $R(\bar{s})$ be a relational atom, G be a set of relational atoms, M be a set of comparisons such that the query $Q(\bar{s}) \leftarrow R(\bar{s}), G, M$ is safe.

Definition 6.2.2 (TC Statement with Comparisons) *A TC statement C with comparisons is an expression of the form Compl$(R(\bar{s}); G, M)$. It has the associated query $Q_C(\bar{s}) \leftarrow R(\bar{s}), G, M$. The statement is satisfied by $\mathcal{D} = (D^i, D^a)$, written $\mathcal{D} \models$ Compl$(R(\bar{s}); G, M)$, iff*

$$Q_C(D^i) \subseteq R(D^a). \tag{6.3}$$

We reformulate TC-QC entailment to take into account comparisons.

Definition 6.2.3 (TC-QC with Comparisons) *Let Q be a query with comparisons under set semantics and $\mathcal{C}$ be a set of TC-statements with comparisons. Then, $\mathcal{C}$ entails completeness of Q, written $\mathcal{C} \models$ Compl(Q), iff for any partial database $\mathcal{D} = (D^i, D^a)$ holds*

$$D \models \mathcal{C} \implies Q(D^i) = Q(D^a).$$

Properties

In this subsection, we are going to investigate properties of the queries and TC statements relevant for completeness reasoning with comparisons. We have already seen in the examples that comparisons can distinguish only certain classes of models. Let $\circ$ be any comparison predicate, then the formula

$$X \circ a,$$

can distinguish at most three classes of models:

1. $X \in (a, \infty)$;
2. $X \in (-\infty, a)$;
3. $X = a$.

It cannot distinguish if $X = a + 1$ or $X = a + 2$. Therefore, instead of considering the whole range of values $(-\infty, +\infty)$, we can select representatives from classes of models.

This is an important property because in the reasoning procedure we operate with the most general answer. If a query returns the most general answer over the ideal database and the available database, then the query is complete with respect to any partial database. In case of built-in predicates, TC statements can distinguish only finitely many general answers, namely every comparison can distinguish at most three classes. Then, it is reasonable to consider general answers that belong to every distinguishable class to establish completeness.

6.3 Reasoning Proposal

We have seen that comparisons can distinguish only several classes of models. The reasoning procedure checks completeness by means of the most general answer. In other words instead of checking every model, it checks the representative from the whole class of considered models. This gives us an idea how to reason with built-in predicates in general.

The standard interpretation for TC statements is the one as transfer rules of atoms from the ideal database to the available database. In the TC-QC problem we check whether enough atoms have been copied from the ideal database to the available to return the most general tuple. We know that comparisons in TC statements can distinguish only several classes of models. Then, we can consider a representative from each class of models. If we take the most general answer as a representative from each class of models, then we can establish completeness in general.

CHAPTER 7

Implementation

In this chapter, we discuss practical aspects of completeness reasoning and implemenation issues of the completeness reasoner. We explain the structure of the reasoner. We show how to use the reasoner to test examples from the work. We provide links and general information about the the reasoner.

7.1 Description

In this section we are going to talk about an implementation of the completeness reasoning system, called MAGIK. It is a web application. It implements all the algorithms described in the previous chapters. It can work with foreign key and finite domains constraints under set and bag semantics. It is fully available at

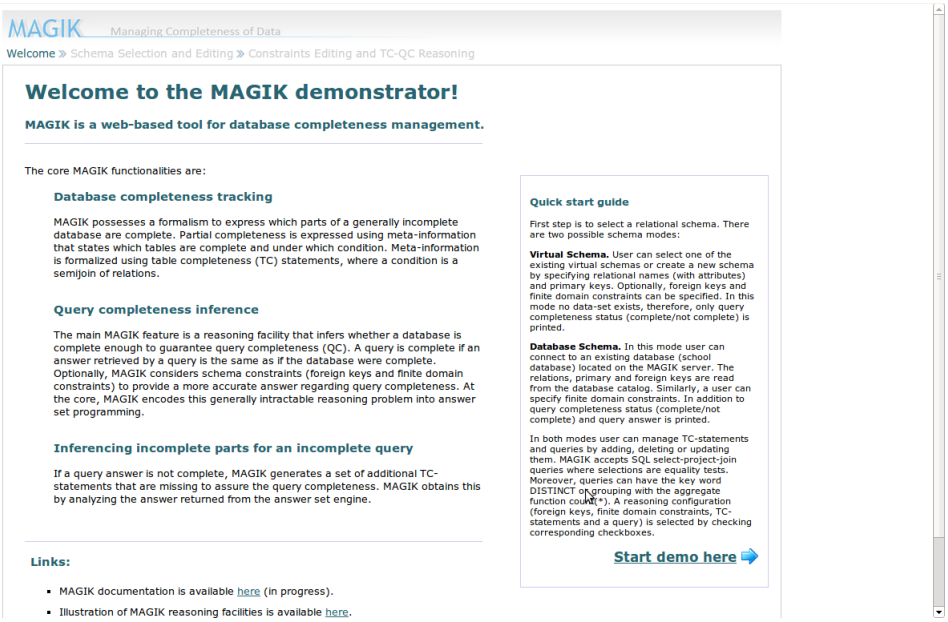
`http://magik-demo.inf.unibz.it`.

As an illustration and short description of MAGIK see Figure 7.1.

The project has a wiki page that describes different available features of the reasoning engine and the structure of the system as well as possible use-cases, see Figure 7.2 .

In general, this implementation is meant to be a proof of concept. It demonstrates the feasibility of the approach. It allows us to show in an interactive way practical aspects of the work. The system can be used as a prototype for data analysis. The functionality of the system goes beyond what has been described in the thesis. The system makes suggestions on TC statements that are needed for the query to be complete. In case of relational queries and in the presence of finite domain constraints, the suggestions are minimal. However, how to make minimal suggestions in the presence of foreign keys is still an open problem. We are not going into detail about suggestions, since this lies beyond the scope of this work.

Figure 7.1: MAGIK Welcome Page



7.2 Completeness Reasoning

Let us illustrate how the system works by an example. We make use of the example from the section 5.5. It shows the interaction between finite domain and foreign key constraints. Let us proceed to the reasoning page 7.3.

Let us to reconstruct the example's setting from the section 5.5. In the tab "Foreign Keys", we introduce the constraint $pupil[level, code] \rightarrow class[level, code]$. In the tab "Finite Domain Constraints", we introduce the constraint $class[code] \in \{a, b\}$. In the tab "Table Completeness Statements" we introduce two constraints:

$$Compl(pupil(Name, 1, b); \top); \tag{7.1}$$

$$Compl(pupil(Name, 1, a); \top). \tag{7.2}$$

In the tab "Queries", we introduce a query Q

```
SELECT p.Name
FROM pupil as p
WHERE p.Level = '1'
```

Then, we obtain the result:

Query is complete.

Figure 7.2: MAGIK Wiki Page

There is also the possibility to look at the generated ASP program by clicking right below the result title.

If we give up any of the constraints, we obtain a different result:

Query is not complete.

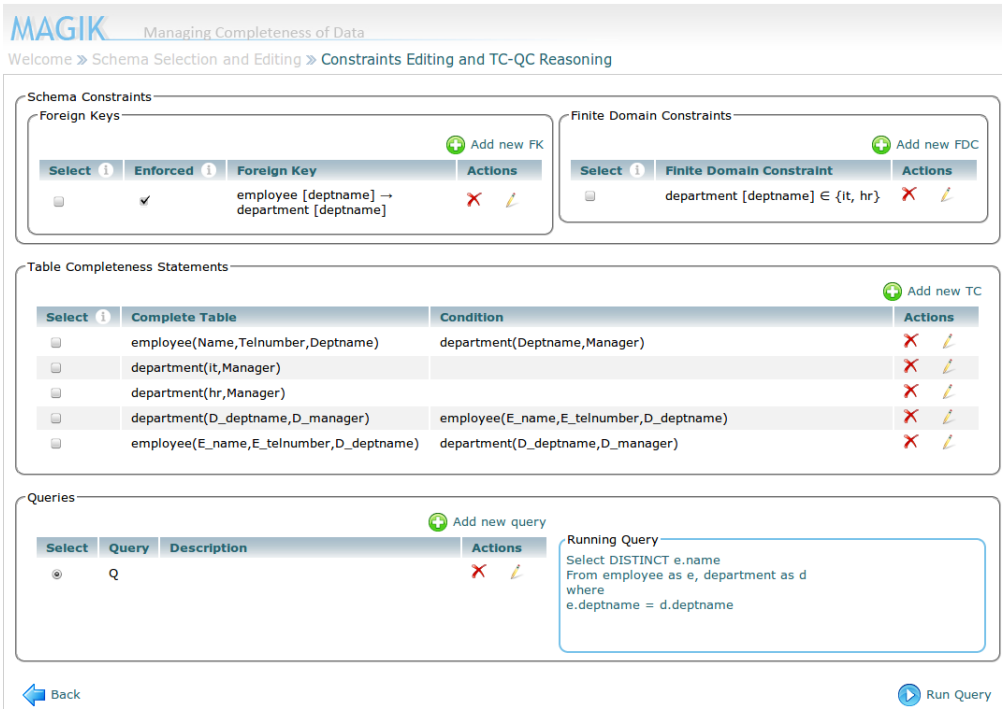
Note, the system suggests a minimal set of TC statements to ensure completeness.

7.3 General Implementation Issues

In this section we are going to discuss several issues with the current implementation of the completeness reasoner. Namely, we are going to talk about interface design, boolean reasoning, mixed notation in the system and reasoning with a database instance. Let us start with interface design.

Interface design It is important to develop a system that allows to state completeness and to reason about query completeness. However, it is not clear how to develop an intuitive interface for stating completeness information. The intended usage of the system was to allow people working with local information to state completeness in an intuitive way. The most straightforward way to implement the form for stating completeness knowledge is based on the logic programming notation or relational algebra. Let us show it with an example. Assume we need to state that every department where some employee work is present in the database. We formulate

Figure 7.3: MAGIK Reasoning Page



this expression as a TC statement.

$$Compl(Department(DeptName, Location); Employee(Name, DeptName, Birthday)). \quad (7.3)$$

To state this formally in the system a user needs at least basis knowledge of logic programming, e.g. a user needs to know that upper case letters stand for variables and letters with the same name are joined. It does not seem to be common knowledge among school teacher, managers and even database administrators. Therefore, it is important to design an interface that can be easily used by a wide class of users and does not require a lot of training.

Boolean reasoning Throughout the work, we have been talking only about boolean reasoning i.e. when the query is either complete or not. However, it does not fully reflect common sense. As we mentioned before, experts say that two percents of records in a customer file become obsolete in one month because customers die, divorce, marry, and move [6]. It means we have to take into account fuzziness to reflect common sense and more even more important to introduce a notation of time into the theory and into the reasoner to make it practical.

Mixture of notations As it is implemented now, when we introduce a TC statement like 7.3, we use logic programming notation. When we introduce a query into the system, we use SQL as

a query language. It might be quite confusing because a user must have at least basic understanding of both notations. Besides, a user needs to have some knowledge or the relational model to work fully with the reasoner. The variety of different notations might cause some confusion.

Reasoning with a database instance It is important to capture different aspects that contribute to the completeness decision procedure. Reasoning with a database can significantly contribute to the reasoning procedure and it is also an important psychological aspect to give a feasible argument about completeness based on the real data. There has been a lot of different studies where completeness with respect to a given instance was analyzed [4], [20], [16], [8], however, how to adopt and implement it in our environment is an open issue.

7.4 Practical Application

Even though the application is meant to be a proof of concept, it is not hard to see how it can applied in practice. Web base access allows local administrators to introduce completeness statements into the system without any special software. It also allows a database administrator to use the same application to investigate completeness properties.

The web based approach and only few forms required to operate the system make integration possible even in a big company. The system is scalable with respect the number of users, since every statement can be added by a local administrator independently via any browser.

It is worth mentioning that the current version of the system does not take into account data degradation over time. This should be taken care of by a database administrator. It seams reasonable to apply the system to ensure completeness of queries in a short time frame after the completeness statements have been added.

A way to apply different reasoning features is presented in Section 7.2. To apply the system in a greater scale, we have to extend the corresponding forms with new information like the schema, the set of constraints and change the query accordingly. This can be done inductively, adding relations and constraints one by one in the corresponding tabs of the forms in the Reasoning Page 7.3.

The first version of the system was demonstrated at CIKM 2012:

`http://www.cikm2012.org/`

CHAPTER 8

Conclusions

In this chapter, we sum up results from the previous chapters. We demonstrate limitations of the theory of query completeness. We also make suggestions for future work.

8.1 Results

We have analyzed the query completeness problem from the logic programming perspective. We have formalized completeness reasoning under set and bag semantics; we have explained how to make an effective check of TC-QC entailment. We have also provided an algorithm that reduces the TC-QC problem to the problem of evaluating of an answer set program. The reduced problem belongs to the same complexity class as the original problem.

We have introduced the completeness reasoning procedure in the presence of finite domain constraints in the schema. We have extended the characterization to capture completeness reasoning in the presence of constraints. In the case of finite domain constraints, the complexity of the problem is higher than in the relational case. This problem with finite domain constraints is Π_2^P-hard. That is the reason why we have introduced disjunction into the encoding. As before, the complexity of reduced problem is the same as the complexity of the original problem. Namely, the original TC-QC problem in the presence of finite domain constraints belongs to Π_2^P and reasoning over disjunctive answer set programs belongs to Π_2^P.

In the same spirit, we have extended the reasoning procedure with foreign key constraints. We have explained two possible semantics of foreign keys in the context of completeness reasoning. Assuming acyclicity of foreign keys, we have provided a characterization and a corresponding reduction for both types of semantics. We showed that under the acyclicity assumption, the complexity stays the same as in the relational case. We have also introduced a reasoning procedure and an encoding to handle both types of constraints in the combination.

We have presented an implementation of a completeness reasoning system. It is able to deal with completeness reasoning problems in the presence of foreign keys and finite domain constrains under set and bag semantics. We have provided an explanation how to test the examples

from the work in the system. We also presented a short description and links to the detailed resources about the project.

Summing up, we conclude that a step towards the development of a query completeness reasoning system has been made. The encoding covers a wide class of commonly used queries. All proposed algorithms have the same complexity as the original problems, therefore they are arguably efficient. Finally, we provided a demonstrator of the system that shows feasibility of the approach.

8.2 Limitations

It is an important question to answer what class of queries and constraints the approach can handle in principle.

Let us start with the soundness assumption. It says there is no wrong data in the available database. Under this assumption, the interpretation of the table completeness statements is transferring of atoms from the ideal database to the available database.

Let us illustrate this with an example. Assume, there is a schema

$$\textit{student}(\underline{\textit{Name}}, \textit{Major}).$$
$$\textit{study}(\underline{\textit{Name}}, \textit{Language}).$$

Assume, there is a query

What are the names of all students who study Latin?

We formulate it as a datalog rule

$$Q(\textit{Name}) \leftarrow \textit{student}(\textit{Name}, \textit{Major}), \textit{study}(\textit{Name}, \text{“}\textit{Latin}\text{”}).$$

The minimal set of TC statements to ensure completeness is

$$\textit{Compl}(\textit{student}(\textit{Name}, \textit{Major}); \textit{study}(\textit{Name}, \text{“}\textit{Latin}\text{”})).$$
$$\textit{Compl}(\textit{study}(\textit{Name}, \text{“}\textit{Latin}\text{”}); \textit{student}(\textit{Name}, \textit{Major})).$$

Under the soundness assumption, they transfer atoms from D^i to D^a.

The situation changes dramatically, if we introduce negation into the query

What are the names of all students who do not study Latin?

Then, we lose our classical interpretation of the TC statements as transferring rules. Now TC statements also can ensure the fact that we have all students in the database who study Latin. It is no longer only a statement about completeness but it is also a statement about correctness. By adding more facts to the available database we can make previous results incorrect due non-monotonicity of the query.

Therefore, as it is described in Denecker et al. [4], negation introduces a duality of correctness and completeness. To extend the theory with negation we must revise the standard interpretation of TC statements. Furthermore, under the current assumption of soundness and the classic interpretation of TC statements, we cannot reason about queries with negation.

8.3 Future Research

Taking into account results and limitations of the theory of query completeness, we propose two main directions of the future research. Firstly, we can investigate other possible database constraints that may affect completeness reasoning e.g. general inclusion constraints or type restrictions. Also, we can extend the developed algorithms to work in more general settings. Secondly, we can enrich the model with fuzzy inferences, e.g., the rules and tables can hold only partially.

Both extensions go into orthogonal directions. The first direction assumes precise reasoning that extracts additional completeness information from the constraints. The second direction changes the setting. It allows to ensure completeness up to a certain level of confidence.

Technically, there are many constraints that one can take into account. Triggers are definitely an interesting kind of integrity constraints that allows us to make additional completeness inferences. We can take into account restrictions on the data types – it is possible to extract finite domain constraints directly from the enumeration types in the database schema. It is also interesting to extend the completeness reasoning procedure with additional information about a database instance.

The other direction is to add fuzziness into the reasoning procedure. For example, we can add a certainty level to the knowledge about completeness of the database. Alternatively, we can interpret finite domain constraints as a measure of completeness, e.g. if we establish completeness for five out of seven cases, then we are five out of seven complete. Additionally, if we take into account instance of the database, we can estimate the size of uncertain data to ensure a confidence level of the data completeness.

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases: The Logical Level*. Addison Wesley, 1994.

[2] Andrea Calì, Domenico Lembo, and Riccardo Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *In Proc. of PODS 2003*, pages 260–271, 2003.

[3] Robert Demolombe. Validity queries and completeness queries. In Zbigniew W. Ras and Maciej Michalewicz, editors, *Foundations of Intelligent Systems, 9th International Symposium, ISMIS 96, Zakopane, Poland, June 9-13, 1996, Proceedings*, volume 1079 of *Lecture Notes in Computer Science*, pages 253–263. Springer, 1996.

[4] Marc Denecker, Álvaro Cortés-Calabuig, Maurice Bruynooghes, and Ofer Arieli. Towards a logical reconstruction of a theory for locally closed databases. *ACM Trans. Database Syst.*, 35(3):22:1–22:60, July 2008.

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

[6] Wayne W. Eckerson. *Data Quality and the Bottom Line*. TDWI, 2002.

[7] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer Set Programming: A Primer. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset, and Renate A. Schmidt, editors, *5th International Reasoning Web Summer School (RW 2009), Brixen/Bressanone, Italy, August 30–September 4, 2009*, volume 5689 of *LNCS*, pages 40–110. Springer, September 2009.

[8] Wenfei Fan and Floris Geerts. Relative information completeness. *ACM Trans. Database Syst.*, 35:27:1–27:44, 2010.

[9] Michael R. Genesereth and Nils J. Nilsson. *Logical foundations of artificial intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[10] D. S. Johnson and A. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, PODS '82, pages 164–169, New York, NY, USA, 1982. ACM.

[11] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7:499–562, 2002.

[12] Alon Y. Levy. Obtaining complete answers from incomplete databases. In *In Proc. of the 22nd Int. Conf. on Very Large Data Bases (VLDB'96)*, pages 402–412, 1996.

[13] Jack Minker and Donald Perlis. Computing protected circumscription. *J. Log. Program.*, 2(4):235–249, 1985.

[14] Amihai Motro. Integrity = validity + completeness. *ACM Trans. Database Syst.*, 14(4):480–502, December 1989.

[15] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010.

[16] Felix Naumann, Johann-Christoph Freytag, and Ulf Leser. Completeness of integrated information sources. *Inf. Syst.*, 29(7):583–615, September 2004.

[17] Simon Razniewski and Werner Nutt. Completeness of queries over incomplete databases. *PVLDB*, 4(11):749–760, 2011.

[18] Werner Nutt Simon Razniewski. Checking query completeness over incomplete data. Technical report, KRDB Research Centre, Free University of Bozen-Bolzano, Bolzano, Italy, March 2011.

[19] Adnan H. Yahya and Lawrence J. Henschen. Deduction in non-horn databases. *J. Autom. Reasoning*, 1(2):141–160, 1985.

[20] Álvaro Cortés-calabuig, Marc Denecker, Ofer Arieli, and Maurice Bruynooghe. Representation of partial knowledge and query answering in locally complete databases. In *In Proc. 13th LPAR, LNCS 4246*, pages 407–421. Springer, 2006.