Constructing queries from data examples

MSc Thesis (*Afstudeerscriptie*)

written by

Valentino Filipetto
(born 24th February, 1998, in Varese, Italy)

under the supervision of **Balder ten Cate**, and submitted to the Examinations Board in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam*.

Date of the public defense:	**Members of the Thesis Committee:**
24th August 2022	Prof. Ekaterina Shutova (chair)
	Prof. Balder ten Cate (supervisor)
	Dr. Johannes Marti
	Dr. Gregor Behnke

INSTITUTE FOR LOGIC, LANGUAGE AND COMPUTATION

Abstract

The Query-by-Example problem (QBE) is about the existence of a fitting query for a given database instance and positive/negative labelled data examples. The question is: given an input database instance and sets of user examples, does there exist an explanatory query whose answers to it include the positive examples but not the negatives? Intuitively, what a fitting query does it to generalize the data according to certain constraints given by the user examples. From a complexity theoretical-standpoint, it was proven in 2010 that the Query-by-Example problem is **coNExpTime**-complete if we consider *conjunctive queries* as query language [18].

While in the relevant literature it is sometimes assumed that the fitting queries for the Query-by-Example problem cannot contain constants, in this thesis we consider precisely what happens if constants can appear in them and we check is there are changes in terms of computational complexity. Throughout the thesis we show how natural it is to allow constants in the queries and we present inputs for QBE where, if constants are now allowed to appear in the queries, no fitting query exists. Our analysis shows that, in most cases, constants do not really influence the computational complexity of QBE.

Acknowledgments

In this space, I want to express feelings of gratitude to those who helped me, in different ways, to write this thesis.

I would like to thank my supervisor Balder ten Cate for always welcoming me with a smile and teaching me patience and precision. Thanks should also go to the members of the defense committee who took the time to read this work.

I thank my family: mum, dad, brother, sister. You have always supported me in my choices and also when, among many doubts, I decided to leave Italy.

I warmly thank all my friends of the Master of Logic: I admire you all for different reasons and I keep within me a space for each of you. I here include all the singers of the Vocal band that I have had the pleasure of getting to know since February 2022.

Two women deserve special thanks (and really these few lines are not enough): Thanks Elena for helping me, from afar, to get off Novecento's ship. Your listening skills are unique and always surprising. Thank you S. for being simply "you", that is, irresistible, delicate and free. If I was able to do this it is also thanks to your constant support and your beautiful personality.

Finally, I should say that writing this thesis, which already was quite the challenge for a series of unforeseen circumstances, would have been much harder without the *music I listened to* in the process – especially Mahler's music; to the grandiose works of Mahler I owe much of the stimulus to my actions, and a particular, intense way of feeling. To this day, I still think that it is a gift of fate to be able to live in the same era in which Mahler was discovered.

Contents

1 Introduction

Managing and querying large volumes of data has become central to many activities and jobs. Being able to ask queries to a database and, more generally, work with data in a relational database is growing in importance beyond the scientific community. For example, SQL, acronym for *Structured Query Language* [13], which has already been a standard for many years, is now recommended to users that don't necessarily work in the IT field [15]. However, not everybody is an expert database user, and constructing suitable queries is, by itself, a challenging task. To deal with this, one possible approach is to provide a repository for users to share their queries [16]. However, this service still requires a certain knowledge of SQL. But there are also approaches that try not to presuppose any background knowledge of this programming language, like [9]. In this case, the user is only required to be able to asses whether or not a given output table is the result of their target query on a given input database. Given that, the system is able to select the desired query among different possibilities.

We study a problem called *Query-by-Example* (QBE), which is a formalization of an approach to solve the problems above and, in this thesis, it is studied from a complexity-theoretical angle. Moreover, the QBE problem shares with the problems above the ultimate goal of understanding how to derive certain queries given a database. QBE is the problem – given a database and sets of positive and negative labelled data examples – to decide if there exists a query that fits the data examples, i.e. the answers to the query should include the positive examples but not the negatives.

To better understand the problem, let us see an example. Take the following database instance (Figure 1.1):

WorksInDepartment	
Name	**Department**
John	Literature
Ada	Philosophy
Jane	Philosophy
David	Computer Science

Teaches	
Name	**Course**
Ada	Political Philosophy
Jane	Ethics
David	Distributed Algorithms
David	Distributed Systems

Figure 1.1: First example of a database instance.

In this case we only have two binary relations *WorksInDepartment* and *Teaches*. Moreover, let us assume that our positive examples are Jane and Ada, while John is the only negative one (note that these examples might be provided by a user as well as come from other sources). A possible fitting conjunctive query is then:

$$q(x) := \exists y \exists z (WorksInDepartment(x, y) \wedge Teaches(x, z))$$

In recent years, researchers that studied QBE from a complexity theoretical standpoint managed to prove that QBE is **coNExpTime**-complete if we consider conjunctive queries as possible fitting queries [1, 8], and **coNP**-complete if we consider union of conjunctive queries [12]. These complexity bounds were obtained under the assumption that a possible fitting query can only contain *variables*, either free or bounded by an existential quantifier. However, it is easy to see how natural it is to consider the case in which queries can also contain *constants*; this way we might obtain queries that are less general but it might be possible to find fitting queries for a certain input while it was impossible in the previous setting. As an example, take again the database instance in Figure 1.1. Furthermore take, as positive example, David, and as negatives Jane, John and Ada. A possible fitting conjunctive query containing a constant (i.e. $ComputerScience$) is the following:

$$q(x) := \exists y (WorksInDepartment(x, ComputerScience) \wedge Teaches(x, y))$$

However, as the reader will realize, without allowing constants in the queries there is no fitting query in this case.

Note that we may also consider *union of conjunctive queries* $q = q_1 \vee \cdots \vee q_n$, where each q_i is a conjunctive query. In this case, an interesting phenomenon happens: we can trivialize the problem by taking the positive examples in the input and creating a conjunctive query for each of them. In

each of these queries the positive example appears in the head of the query, and so the query denotes it when it is evaluated against a database. As a result, the disjunction of all these conjunctive queries is a fitting query. In this thesis we will also see how to deal with this aspect of union of conjunctive queries.

Overall, the aim of this thesis is to try and fill a gap in the study of the Query-by-Example problem by studying the same problem under the assumption that constants can appear in queries.

Our original contribution is then twofold:

1. We provide a *conceptual contribution*, namely while we present a setting in which constants are allowed in the queries, we also try to avoid a *trivialization* of the problem. In fact, as we already mentioned above, a possible trivialization happens if we consider union of conjunctive queries and we can mention constants freely in the queries. One possible way to do so is to say that we can take advantage of the constants but we cannot "overfit" the queries. In other words, we slightly modify the Query-by-Example problem in such a way that there are constraints on how constants can appear in the queries.

2. We provide a *technical contribution*, namely we prove some complexity bounds for the QBE problem with constants. In particular, we prove that the QBE problem is still **coNExpTime**-complete if constants can appear (freely) in the queries and the input database contains only constants. We also prove complexity bounds for the QBE problem with constraints on how constants can appear in the queries.

In Chapter 2 we introduce the necessary background to understand the content of this thesis; Section 2.1 introduces databases, queries and some important operations between database instances. Moreover, Section 2.2 provides essential notions of computational complexity theory.

In Chapter 3 we present in detail the Query-by-Example problem. In Section 3.1 we present the problem and we give examples and motivation. Furthermore, in Section 3.2 we present the proofs for the known results that QBE is **coNExpTime**-complete if we consider conjunctive queries as possible fitting queries and **coNP**-complete if we consider union of conjunctive queries.

In Chapter 4 we turn to the main contributions of this thesis. In Section 4.1 we discuss the high-level setting for introducing constants in the queries, while in Section 4.2 we present definitions of the same notions we saw in

Section 2.1 but this time taking into account constants. Finally, in Section 4.3 we present upper bounds for different QBE problems with the addition that now constants can appear in the queries. In 4.4 we present some lower bounds for the same problems.

2 Background

In this chapter we provide the necessary background in database theory and computational complexity theory that is needed for the following chapters. We first introduce the concepts of database and queries, then we see how they interact via homomorphisms with the classic Chandra-Merlin theorem. Moreover, we present two basic propositions about the direct product operation that we will use later in the thesis. At the end, we discuss some well-known complexity-theoretical notions.

2.1 Background on database theory

We introduce the notion of database in the context of the so-called *relational model*, i.e. a model for organizing the data of database in tuples and relations. The relational model was introduced by Edgar F. Codd in 1970 and, later, the same researcher proposed two query languages: relational algebra and relational calculus [3, 4]. As it is clear from the name, the relational model is based on the concept of *relation*. Formally, a relation is a subset of a cartesian product of sets, while informally a relation can be thought as just a "table" with rows and columns. This motivates the idea of a "relation schema" $R(A_1, \ldots, A_k)$ where $(A_1, \ldots, A_k)$ is a sequence of k attributes. Here the attributes $(A_1, \ldots, A_k)$ can get names standing for any property we might want, such as name, surname, salary, BSN number and so on. In logic though we do not usually refer to the attributes of a relation by name but by *position*, and we just consider *relation symbols* of a certain associated arity k.

We define:

FatherOf	
Father	**Son**
Mark	John
John	Robert
Robert	David

Figure 2.1: Example of a database instance.

Definition 2.1.1 (Database schema)**.**
A database schema $\sigma = \{R_1, \ldots, R_n\}$ is a set of relation symbols with an associated arity $arity(R_i)$.

We are now ready to define the concept of database instance:

Definition 2.1.2 (Database instance)**.**
Fix some infinite set of values $\mathcal{V}$. A database instance $\mathcal{D}$ over a database schema $\sigma = \{R_1, \ldots, R_n\}$ is a finite tuple of this form: $(R_1^{\mathcal{D}}, \ldots, R_k^{\mathcal{D}})$, where $R_i^{\mathcal{D}} \subseteq \mathcal{V}^k$ for $k = arity(R_i)$. We further define the *active domain* $\text{adom}(\mathcal{D}) \subseteq \mathcal{V}$ of $\mathcal{D}$ as the set containing exactly the elements of $\mathcal{V}$ appearing in each $R_i^{\mathcal{D}}$.

Definition 2.1.3 (Fact)**.**
Fix some infinite set of values $\mathcal{V}$ and a database schema $\sigma = \{R_1, \ldots, R_n\}$. We say that $R_i(\overline{a}_i)$ is a *fact*, where $R_i \in \sigma$ and has arity $n > 0$, and $\overline{a}_i = a_1, \ldots, a_n$ are values coming from $\mathcal{V}$.

We introduced the concept of database instance as a sequence $(R_1^{\mathcal{D}}, \ldots, R_k^{\mathcal{D}})$, but note that later we may also think of a database instance as a finite *set of facts* of the form $\{R_1(\overline{a}_1), \ldots, R_n(\overline{a}_n)\}$. See 2.1 for an example of a database instance.

As mentioned earlier, Codd introduced two query languages for the relational model, i.e. relational algebra and relational calculus. While relational algebra is an algebraic formalism, relational calculus is a logical one, and queries are expressed as formulas of first-order logic. In this thesis we only focus on the relational calculus and – even more specifically – on a single type of query:

Definition 2.1.4 (Conjunctive query)**.**
Given $k \geq 0$, a k-ary *conjunctive query* (CQ) q over a schema σ is an expression of the form

$$q(\overline{x}) := \exists y_1, \ldots, y_m(\alpha_1 \wedge \cdots \wedge \alpha_n)$$

with $m \geq 0$ and $n \geq 1$. Moreover, $\overline{x} = x_1, \ldots, x_k$ is a sequence of variables, and each α_i is an atomic formula using a relation from σ and variables from $x_1, \ldots, x_k, y_1, \ldots, y_m$. In addition, it is required that each variable in $\overline{x}$ occurs in at least one conjunct α_i (this requirement is called *safety*). Besides, note that the tuple $\overline{x}$ may contain repetitions. Lastly, a remark on terminology: given the conjunctive query above, we sometimes call $q(\overline{x})$ the "head" of the query while we call $\exists y_1, \ldots, y_m(\alpha_1 \wedge \cdots \wedge \alpha_n)$ the "body" of the query.

Conjunctive queries are the core language for databases [17] and they are, for instance, expressible in the *structured query language* SQL by an expression of this form:

```
SELECT <list of attributes>
FROM   <list of relation names>
WHERE  <conjunction of equalities>
```

Example 2.1.1. We now present some examples of conjunctive queries. The first CQ asks for all the pairs (x, y) such that there is a path of length 3 between x and y.

$$q(x, y) := \exists z \exists w (R(x, z) \wedge R(z, w) \wedge R(w, y))$$

This other CQ instead asks if there is cycle of length 3:

$$q() := \exists x \exists y \exists z (R(x, y) \wedge R(y, z) \wedge R(z, x))$$

Queries are used to ask questions to a database, and to really understand what this means if we define their *semantics*. To this end, we also say that, given a database instance $\mathcal{D}$, we may consider certain elements of its active domain as "distinguished", and we write $(\mathcal{D}, \overline{a})$ to say that $\overline{a}$ is a tuple of distinguished elements belonging to the active domain of $\mathcal{D}$. Moreover, now that we have the notion of distinguished element, we require that our database instances with distinguished elements are always *safe*, i.e. each distinguished element of the database instance occurs in at least one fact of it,

We are now ready to present the semantics of CQs:

Definition 2.1.5 (Semantics of CQs)**.**

Given a database instance with distinguished elements $(\mathcal{D}, \overline{a})$ over σ, a k-ary CQ $q(\overline{x}) := \exists \overline{y}(R_1(\overline{y}_1) \wedge \cdots \wedge R_n(\overline{y}_n))$, a valuation $v : \mathrm{Var}(q) \to \mathrm{adom}(\mathcal{D})$, then:

$$q(\mathcal{D}) = \{v(\overline{x}) \mid \text{where } \forall i = 1, \ldots, n : v(\overline{y}_i) \in R_i^{\mathcal{D}}\}$$

where Var(q) is the set of variables contained in q and $v(\overline{x})$ stands for $(v(x_1), \ldots, v(x_k))$.

Intuitively what this means is that, when evaluated against $\mathcal{D}$, the CQ q returns a set $q(\mathcal{D})$ that consists of all k-tuples $(a_1, \ldots, a_k)$ that make the formula "true" in $\mathcal{D}$. Note that we may deal with *boolean* CQs, i.e. CQs without free variables; in that case, in the definition of the semantics of CQs, $v(\overline{x})$ is the empty tuple.

Example 2.1.2. The first CQ of example 2.1.1, rewritten as

$$q(x, y) := \exists z \exists w (FatherOf(x, z) \wedge FatherOf(z, w) \wedge FatherOf(w, y))$$

and evaluated against the database of example 2.1 returns $q(\mathcal{D}) = \{(Mark, David)\}$.

In the next chapters we will also work with union of conjunctive queries (UCQs):

Definition 2.1.6 (Union of conjunctive queries)**.**
A union of conjunctive queries q is an expression of the form $q_1 \vee q_2 \vee \cdots \vee q_n$ where each q_i is a CQ and has the same arity k.

Example 2.1.3. For example, the UCQ $q = q_1 \vee q_2$ (where $q_1(x, y) := \exists z(R(x, z) \wedge R(z, y))$ and $q_2(x, y) := \exists w \exists z(R(x, w) \wedge R(w, z) \wedge R(z, y))$) returns the endpoints of each path of length exactly 2 or exactly 3.

There is a correspondence between conjunctive queries and database instances that we are now going to explain. On the one hand, we can get the *canonical database instance* of a CQ $q(\overline{x})$:

Definition 2.1.7 (Canonical database instance)**.**
Given a CQ $q(\overline{x})$, we say that $(\mathcal{D}_q, \overline{x})$ is its *canonical database instance*, where the facts of $\mathcal{D}_q$ are the conjuncts of q and the active domain of $\mathcal{D}_q$ is the set of variables occurring in q

Example 2.1.4. Given a CQ $q(x, y, z) := R(x, y) \wedge R(z, z)$, its canonical database instance $(\mathcal{D}_q, (x, y, z))$ is $\{R(x, y), R(z, z)\}$.

And the other way around:

Definition 2.1.8 (Canonical query)**.**
Given a database instance with distinguished elements $(\mathcal{D}, \overline{a})$, we can associate to it a k-ary *canonical conjunctive query* $q_{\mathcal{D}}(\overline{x})$, namely the query that has a variable x_a for every value a in the active domain of $\mathcal{D}$ occurring in at least one fact, and a conjunct for every fact of $\mathcal{D}$. By construction, $q_{\mathcal{D}}(\overline{x})$ is safe, because each variable in $\overline{x}$ has to occur in at least one conjunct of the query.

Example 2.1.5. Given a database instance $\mathcal{D} = \{R(a, b), R(b, a), R(c, c)\}$, where $\mathcal{D}$ has the tuple $\overline{a} = (a, b)$ as distinguished elements, the canonical query of $\mathcal{D}$ is $q(x_a, x_b) := \exists x_c(R(x_a, x_b) \wedge R(x_b, x_a) \wedge R(x_c, x_c))$.

A concept that will be particularly important in this thesis in the concept of *homomorphism*:

Definition 2.1.9 (Homomorphism)**.**
Given database instances with distinguished elements $(\mathcal{D}_1, \overline{a})$ and $(\mathcal{D}_2, \overline{b})$, we say that a *homomorphism* h is a map from adom$(\mathcal{D}_1)$ to adom$(\mathcal{D}_2)$ such that for all $R \in \mathcal{D}_1$ and every tuple $\overline{a} = (a_1, \ldots, a_n)$ belonging to $R^{\mathcal{D}_1}$, the tuple $h(\overline{a}) = (h(a_1), \ldots, h(a_n))$ belongs to $R^{\mathcal{D}_2}$. In addition, regarding the distinguished elements, it must be the case that $h(\overline{a}) = \overline{b}$.

The notion of homomorphism allows us to give an alternative characterization of the semantics of CQs, which is expressed by the following, classic, result:

Theorem 2.1.1 (Chandra-Merlin, [5, 2])**.**
Given a CQ $q(\overline{x})$ and a database instance with distinguished elements $(\mathcal{D}, \overline{a})$, the following holds:

$$\overline{a} \in q(\mathcal{D}) \text{ if and only if } (\mathcal{D}_q, \overline{x}) \to (\mathcal{D}, \overline{a}).$$

This theorem can also be generalized to UCQs in the following way:

Theorem 2.1.2 (Chandra-Merlin for UCQs, [5])**.**
Given a UCQ $q = q_1 \vee \cdots \vee q_n$ and a database instance with distinguished elements $(\mathcal{D}, \overline{a})$, the following holds:

$$\overline{a} \in q(\mathcal{D}) \text{ if and only if } (\mathcal{D}_{q_i}, \overline{x}) \to (\mathcal{D}, \overline{a}) \text{ for some } 1 \leq i \leq n.$$

Two operations between database instances that will come in handy later are the operation of *union* and the operation of *direct product*:

Definition 2.1.10 (Union of database instances)**.**
Given database instances $\mathcal{D}_1$ and $\mathcal{D}_1$ such that $\mathcal{D}_2 = \{R_1(\overline{a}_1), \ldots, R_n(\overline{a}_n)\}$ and $\mathcal{D}_2 = \{R_2(\overline{a}_1), \ldots, R_m(\overline{a}_m)\}$, the *union* of $\mathcal{D}_1$ and $\mathcal{D}_2$ is $\mathcal{D}_1 \cup \mathcal{D}_2$. If $\text{adom}(\mathcal{D}_1) \cap \text{adom}(\mathcal{D}_2) = \emptyset$, we will call their union *disjoint union* and we will denote it as $\mathcal{D}_1 \uplus \mathcal{D}_2$.

Definition 2.1.11 (Direct product of database instances)**.**
The direct product of n-ary tuples $\overline{a} = a_1, \ldots, a_n$ and $\overline{b} = b_1, \ldots, b_n$ (i.e. $\overline{a} \otimes \overline{b}$) is the n-ary tuple $((a_1, b_1), \ldots, (a_n, b_n))$. If $\mathcal{D}_1$ and $\mathcal{D}_2$ are database instances, we define $\mathcal{D}_1 \otimes \mathcal{D}_2$ to be the following database instance specified as a set of facts:

$$\{R(\overline{a} \otimes \overline{b}) \mid R \in \sigma, R(\overline{a}) \in \mathcal{D}_1, R(\overline{b}) \in \mathcal{D}_2\}$$

We use $(\mathcal{D}_1, \overline{a}) \otimes (\mathcal{D}_2, \overline{b})$ to denote the pair $(\mathcal{D}_1 \otimes \mathcal{D}_2, \overline{a} \otimes \overline{b})$, and write $\prod_{1 \leq i \leq m}(\mathcal{D}_i, a_i)$ as a shorthand for $(\mathcal{D}_1, a_1) \otimes \cdots \otimes (\mathcal{D}_m, a_m)$ (note that the "$\otimes$" operation is associative up to isomorphism).

Example 2.1.6. As an example, consider the following direct product: if we have a database instance $\mathcal{D} = \{R(a, b), R(b, c), S(c, d)\}$, $\overline{a}_1 = (a, b)$ and $\overline{a}_2 = (c, d)$, then $\overline{a}_1 \otimes \overline{a}_2 = ((a, c), (b, d))$ and, if we take the product of $\mathcal{D}$ with itself, the resulting database instance is:

$$\mathcal{D} \otimes \mathcal{D} = \{R((a,a),(b,b)), R((b,b),(c,c)), \\ R((a,b),(b,c)), R((b,a),(c,b)), S((c,c),(d,d))\}$$

The direct product operation "$\otimes$" defines the least upper bound in the lattice of databases defined by the notion of homomorphism. This is testified by the following two propositions:

Proposition 2.1.3 ([1])**.** *Given database instances $(\mathcal{D}_1, \overline{a}_1), \ldots, (\mathcal{D}_n, \overline{a}_n)$, the following holds:*

$$\prod_{i \leq n}(\mathcal{D}_i, \overline{a}_i) \to (\mathcal{D}_i, \overline{a}_i)$$

We will prove a more general version of this in Proposition 4.2.3 in Chapter 4.

Proposition 2.1.4 ([1])**.** *Given database instances $(\mathcal{D}_1, \overline{a}_1), \ldots, (\mathcal{D}_n, \overline{a}_n)$ and a database instance $(\mathcal{D}, \overline{b})$, the following are equivalent:*

- $(\mathcal{D}, \bar{b}) \to \prod_{i \leq n}(\mathcal{D}_i, \bar{a}_i)$.
- $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_i, \bar{a}_i)$ *for each* $i \leq n$.

We will prove a more general version of this in Proposition 4.2.4 in Chapter 4.

2.2 Background on complexity theory

We present some basic notions from computational complexity theory that will soon become important. In this presentation, we closely follow the classic [11].

Definition 2.2.1 (Deterministic time)**.**
Let $T : \mathbb{N} \to \mathbb{N}$ be some function. We write $\mathbf{DTIME}(T(n))$ to mean the set of all functions that are computable by a Turing machine in $c \cdot T(n)$-time for a constant $c > 0$.

This allows us to introduce:

Definition 2.2.2 (The class $\mathbf{P}$)**.**
$\mathbf{P} = \cup_{c \geq 1}\mathbf{DTIME}(n^c)$

We then of course need to briefly introduce some concepts from the heavy-handed theory of **NP**-completeness.

Definition 2.2.3 (The class $\mathbf{NP}$)**.**
For every $L \subseteq \{0,1\}^*$ we say that $L \in \mathbf{NP}$ if there is a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a function $g \in \mathbf{P}$ such that for all $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } g(x,u) = 1.$$

When, given a $x \in L$ and a $u \in \{0,1\}^{p(|x|)}$, we have that $\mathbb{M}(x,u) = 1$ then we say that x is a *certificate* for the input x. Roughly speaking, a problem is in **NP** if given a candidate solution to it, it is computationally easy to check for its correctness.

For a further investigations of other useful complexity classes, and in order to give an alternative definition of the class **NP**, we introduce nondeterministic Turing machines. A nondeterministic Turing machines $\mathbb{M}$ (NDTM from now on) is different from the usual Turing machine in that the former has two transitions functions δ_0 and δ_1; then, at each computational step, the machine makes a choice as to which of its two transitions functions to

apply. The machine $\mathbb{M}$ outputs 1 on a certain input x if there is a *sequence* of choices that make the machine reach an accepting state. In other words, the machine $\mathbb{M}$ accepts if it is able to "guess" a sequence of choices that make it possible to reach the final accepting state. This new machine allows us to define a new complexity class:

Definition 2.2.4 (**NTIME**)**.**
For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$ we say that $L \in \mathbf{NTIME}(T(n))$ if there is a constant $c > 0$ and $c(T(n))$-time NDTM $\mathbb{M}$ such that for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow \mathbb{M}(x) = 1$.

In fact, the meaning of **NP** is *non-deterministic polynomial time*, and this is motivated by the following alternative characterization of this class:

Theorem 2.2.1. $\mathbf{NP} = \cup_{c \in \mathbb{N}} \mathbf{NTIME}(n^c)$

Finally, we introduce the basics of the theory of **NP**-completeness. One of the main concepts of this theory is the interesting idea of *reducing* a problem to another one.

Definition 2.2.5 (Polynomial-time reduction)**.**
Given two languages $A, B \subseteq \{0,1\}^*$ we say that A is polynomial-time reducible (in symbols $A \leq_p B$) if there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$, $x \in A \Leftrightarrow f(x) \in B$.

Definition 2.2.6 (**C**-Completeness)**.**
Given a certain complexity class **C**, A language $A \subseteq \{0,1\}^*$ is **C**-complete (under polynomial-time reductions) if:

- A is in **C**;
- For every problem A' in **C** it is the case that $A' \leq_p A$ (A is then "**C**-hard").

If we prove that a certain problem A is in a certain complexity class, then we say that have established the *upper bound* of A, while determining whether A is complete for that class can be seen as a *lower bound*, because it establishes that A is at least as hard as all the other problems in that complexity class, and thus it is among the hardest problems contained in it. In other words, A cannot be less complex than any problem in that complexity class, i.e. we have a lower bound for it.

For every complexity class **C**, we can define its *complementary* **coC**. This was we can present:

Definition 2.2.7 (The class **coNP**)**.**
The class **coNP** is the class of all problems $L \subseteq \{0,1\}^*$ whose complement is in **NP**.

To conclude, we introduce the exponential-time analogues of **P** and **NP**:

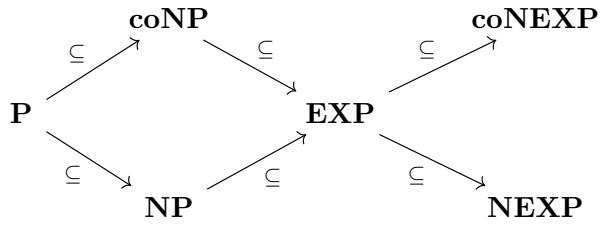
Definition 2.2.8 (The classes **ExpTime** and **NExpTime**)**.**

- $\mathbf{ExpTime} = \cup_{c \geq 0} \mathbf{DTIME}(2^{n^c})$.
- $\mathbf{NExpTime} = \cup_{c \geq 0} \mathbf{NTIME}(2^{n^c})$.

Finally, we define

Definition 2.2.9 (The class **coNExpTime**)**.**
The class **coNExpTime** is the class of all problems $L \subseteq \{0,1\}^*$ whose complement is in **NExpTime**.

To sum up, the following picture gives an overall idea of the relationship between the complexity classes presented:



3

The Query-by-Example problem

In this chapter we will explain the problem we are going to tackle in this thesis. We first introduce the Query-by-Example problem from a high-level perspective and then, in the subsequent sections, we present results, already known in the literature, about complexity bounds on this problem. These bounds cover both the case in which we consider conjunctive queries as fitting queries and the case of union of conjunctive queries.

3.1 Introduction to the problem

In this thesis we study the *Query-by-Example* (QBE) problem. Informally, the idea is that we are given a database instance $\mathcal{D}$ and a certain query language $\mathcal{L}$, as well as positive examples S^+, i.e. tuples of objects from $\text{adom}(\mathcal{D})$ that we want to be included in the denotation of our query once it is evaluated against the database, and negative examples S^-, i.e. tuples from $\text{adom}(\mathcal{D})$ that are undesirable. We then ask if there is a way to reverse-engineer this desired query given the examples.

More formally, we can present the QBE problem in this way:

Query-by-Example for CQs	
Input:	A database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting CQ q for $\mathcal{D}$, i.e. a CQ q such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

We can also consider the same problem, but for UCQs:

QUERY-BY-EXAMPLE PROBLEM FOR UCQS	
Input:	A database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q for $\mathcal{D}$, i.e. a UCQ q such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

Example 3.1.1. Let us consider some examples to develop an intuition of these problems. For QBE for CQs, take the following database instance (already introduced in Chapter 3 (Figure 2.1):

FatherOf	
Father	**Son**
Mark	John
John	Robert
Robert	David

Say that the positive example is the pair (Mark, David) and the negative example is the pair (John, Robert). Moreover, we just have the relation "FatherOf", of arity 2. A fitting CQ is then $q(x, y) := \exists y, \exists z(FatherOf(x, w) \wedge FatherOf(w, z) \wedge FatherOf(z, y))$.

Sometimes it can be the case that, while there is no fitting CQ for a certain input, there is a fitting UCQ. Take again the same database instance above and now take, as positive examples, the following pairs: (Mark, David), (Mark, Robert) and (John, David). Furthermore, assume we have no negative example. As before we just have the binary relation "FatherOf". We observe that a CQ alone is not enough, we need UCQs. A possible fitting UCQ is then:

$$q_1(x, y) := \exists y, \exists z(FatherOf(x, w) \wedge FatherOf(w, z) \wedge FatherOf(z, y)) \\ \vee\, q_2(x, y) := \exists z(FatherOf(x, z) \wedge FatherOf(z, y))$$

As a last example, consider the database instance 3.1. We say that the positive example is Mark while the negatives are John, Alice and Terence. Furthermore, we only have the binary relation "WorksInDepartment". As the reader will realize, there is no fitting query in this case. Note though that in Chapter 4 we will discuss this example again and see that, if we allow constants to appear in the queries, we can actually find a fitting query for this input.

WorksInDepartment	
Name	**Department**
Mark	Sales
John	Service
Alice	IT
Terence	Service

Figure 3.1: New database instance.

The Query-by-Example problem has been extensively studied, and in some cases it has been slightly modified to adapt it to other, more specific, settings. We will now review some of these approaches.

- In [7] the authors first propose a setting in which positive and negative examples are identified with a label and we want to find a sequence of features (i.e., in that case, conjunctive queries) such that they *separate* the positive and negative examples according to a classifier. The use of a classifier makes this approach more similar to a machine-learning problem than a logic one though, and it provides a more fine-grained tool for classifying the examples. The relationship between this setting and the Query-by-Example problem is that the latter problem corresponds to the case of a single-feature classifier. In other words, QBE can be seen as a more specific framework than the one proposed in [7]. At the same time, the authors use QBE and its multiple results (e.g. the fact that the Query-by-Example problem for CQs is **coNExpTime**-complete) as a stepping stone to prove results in their own setting.

- In [9] the authors propose a method, called "Query-from-Examples" (QFE), that can help non-expert database users construct the SQL queries they want. QFE takes as input a database instance and the database instance that is the result of evaluating the target query against the former database. Then, QFE can identify the target query by seeking the user's feedback on a sequence of modified versions of the original pair database-resulting database. These additional pairs are generated by the system. By looking at the user's feedback, QFE figures out the query the user is looking for.

- In [10], yet another view of the QBE problem is presented. In this case, the framework requires as input a database instance and a resulting

database instance T that we get when we evaluate a *known* or *unknown* target query Q against the former database. Then, through a data classification approach, the system tries to construct a new query Q' such that – when we evaluate it against D – we get T. The most obvious application of this framework is to discover alternative characterizations of the data, or even to uncover hidden relationships within a complex database. As an example, consider a case in which we query a movie database about the movies directed by James Cameron after 1997. The framework above will propose another query Q', i.e. "select movies that have gross-revenue greater than $2 billion" that, once it is evaluated against the same database, gives the same output.

3.2 Query-by-Example for CQs and UCQs

We now review the complexity of the Query-by-Example problem[1] if we consider conjunctive queries and union of conjunctive queries. Throughout this chapter we will mostly refer to [8], which will guide our explanations. Note that, in this section, we will present detailed proofs for results already known in the literature because – in the remaining of the thesis – we will build on these results.

3.2.1 Query-by-Example for CQs

Let us consider again the Query-by-Example problem:

QUERY-BY-EXAMPLE FOR CQS	
Input:	A database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting CQ q for $\mathcal{D}$, i.e. a CQ q such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

As shown in [8], this problem is in **coNExpTime**. To see why, we can think of a procedure to check if there is a fitting CQ which involves the notion of direct product, already introduced in Chapter 2. This procedure can be introduced in terms of a "test", which we will call *QBE test*. Here is how the QBE test works: the test takes as input a database instance $\mathcal{D}$ and n-ary relations S^+ and S^- over $\mathcal{D}$. It accepts if and only if:

[1] The name "Query-by-Example" might sound misleading due to the fact that the problem shares the name with a graphical database query language created at IBM during the mid-1970s [19]. We decided to use the name "Query-by-Example" anyway because it was introduced in [8] and, in the end, it was easier to stick to it.

1. $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is safe.
2. $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$.

Then, in [8], the authors provide a "semantic" characterization of the test via the following proposition:

Proposition 3.2.1 ([8])**.** *Given a database instance $\mathcal{D}$ as well as relations S^+ and S^- over it, there is a fitting conjunctive query (i.e. a query q such that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$) if and only if the QBE test accepts $\mathcal{D}$ and both S^+ and S^-.*

Proof.

Fix an arbitrary database instance $\mathcal{D}$ and relations S^+ and S^- over $\mathcal{D}$. For the forward direction, assume that there is a conjunctive query $q(\overline{x})$ such that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$. Assume for a contradiction that there is a $\overline{b} \in S^-$ such that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \to (\mathcal{D}, \overline{b})$. We assumed that $S^+ \subseteq q(\mathcal{D})$ and this, by the Chandra-Merlin theorem, implies that $(\mathcal{D}_q, \overline{x}) \to (\mathcal{D}, \overline{a})$ for all $\overline{a} \in S^+$. This in turns implies, by Proposition 2.1.4, that $(\mathcal{D}_q, \overline{x}) \to \prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$. But we also assumed that there is a $\overline{b} \in S^-$ such that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \to (\mathcal{D}, \overline{b})$, so we can compose together these last two maps to claim that $(\mathcal{D}_q, \overline{x}) \to (\mathcal{D}, \overline{b})$. Contradiction. It follows that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$ for all $\overline{b} \in S^-$. Furthermore, we need to show that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is safe. To do so, assume for a contradiction that it isn't, i.e. there a distinguished element $(a_1, \ldots, a_k)$ of the product such that $(a_1, \ldots, a_k)$ does not occur in *any* fact in the product. We know that $(\mathcal{D}_q, \overline{x}) \to \prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$, so there is some x_i in $\overline{x}$ that is mapped to $(a_1, \ldots, a_k)$ by an homomorphism. Besides, by definition we know that $\mathcal{D}_q$ is safe, so there is a fact f in $\mathcal{D}_q$ where x_i occurs. But this is a contradiction, because the aforementioned homomorphism would not be able to map f to a fact in $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$. We can therefore conclude that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is safe. As a result, the QBE test accepts on input $(\mathcal{D}, S^+, S^-)$.

For the backward direction, assume that the QBE test accepts on input $(\mathcal{D}, S^+, S^-)$. This means that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$, and that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is safe. Now, for notational convenience we say that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) = \mathcal{C}$ and we take the canonical query of the product $q_{\mathcal{C}}(\overline{x})$.

We want to show that $q_{\mathcal{C}}(\overline{x})$ is a fitting CQ for $(\mathcal{D}, S^+, S^-)$. We can first show that $S^+ \subseteq q_{\mathcal{C}}(\mathcal{D})$ by noticing that, by Proposition 2.1.3, the direct product of the positive examples (i.e. $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$) can be homomorphically mapped to all the positive examples themselves. Hence, it is easy to see that $(\mathcal{D}_{q_{\mathcal{C}}}, \overline{x}) \to (\mathcal{D}, \overline{a})$ for all $\overline{a} \in S^+$. We then want to show that $S^- \cap q_{\mathcal{C}}(\mathcal{D})$. To

do so, note that we assumed that $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$. Again, it is easy to check that this transfers to the canonical database of the canonical query $q_{\mathcal{C}}(\mathcal{D})$ of $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) = \mathcal{C}$, that is to say, we can claim that $(\mathcal{D}_{q_{\mathcal{C}}}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$. This was to show.

We can conclude that $q_{\mathcal{C}}(\overline{x})$ is a fitting CQ for $(\mathcal{D}, S^+, S^-)$, as wanted. □

Thanks to Proposition 3.2.1, we can obtain an upper bound **coNExpTime** on the complexity of the Query-by-Example problem through the previously introduced QBE test. Assume that S^+ and S^- are, respectively, relations of positive and negative examples over a database instance $\mathcal{D}$. We see that to check that there is *not* a fitting conjunctive query for S^+ and S^- over $\mathcal{D}$, either we check whether $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is unsafe, which takes exponential time because the direct product is an exponential construction, or we try and guess a negative example $\overline{b}$ in S^- and a homomorphism $h : \prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \to (\mathcal{D}, \overline{b})$. This second task also takes exponential time, because the homomorphism h (our *certificate*) is exponential in the size of $\mathcal{D}$ (note that the domain of a direct product of a database instance $\mathcal{D}$ with itself has cardinality $|\mathcal{D}|^{|\mathcal{D}|}$), and therefore it can be verified only in exponential time by a *deterministic* machine. Since we showed that the *complement* of the problem runs in **NExpTime**, the upper bound for this problem is **coNExpTime**. We can therefore claim that:

Theorem 3.2.2 ([8])**.** *The Query-by-Example problem for CQs is in* **coNExpTime***.*

Proving the lower bound for the same problem is a bit more involved. In [1] it is proven that the so-called *CQ-definability problem* is **coNExpTime**-hard already for unary queries over a fixed schema consisting of a single binary relation. This is done through a reduction from the so-called *Product Homomorphism Problem* (with a single binary relation):

Product Homomorphism Problem (with a single binary relation).	
Input:	A collection of database instances $\mathcal{D}_1, \ldots, \mathcal{D}_n, \mathcal{D}$ over the same schema σ, where σ only contains a single binary relation R.
Question:	Is there a homomorphism from $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n$ to $\mathcal{D}$?

In the same paper, it is shown that Product Homomorphism Problem for databases with a single binary relation is **coNExpTime**-complete.

The CQ-definability problem takes as input a database instance $\mathcal{D}$ and a relation S over the active domain of $\mathcal{D}$, and ask if there is a conjunctive query

q such that $q(\mathcal{D}) = S$. Clearly, this problem is a special case of the fitting problem for CQs: simply take $S = S^+$, and so if $q(\mathcal{D}) = S$ then $S^+ \subseteq q(\mathcal{D})$ and given S^-, because $S = S^+ \cap S^- = \emptyset$, $q(\mathcal{D}) \cap S^- = \emptyset$.

We now present a proof of the **coNExpTime**-hardness of the CQ-definability problem; the proof is just a more detailed version of a proof contained in [1] and it deserves a presentation here also because we build on it in the remaining of the thesis. As a last remark, we say that the original proof uses first-order structures instead of database instances, which we will employ in this presentation. The difference between the two notions is that – unlike database instances – structures contain a *domain*. However, this difference does not influence the proof that we are going to present.

Theorem 3.2.3. ([1]) *The CQ-definability problem is* **coNExpTime***-hard already for unary queries over a fixed schema consisting of a single binary relation.*

Proof.

We give a reduction from the Product Homomorphism Problem with a single binary relation. Let $\mathcal{D}_1, \ldots, \mathcal{D}_n, \mathcal{D}$ be the input of the Product Homomorphism Problem where the schema contains only a binary relation R. We can assume without loss of generality that $\mathcal{D}_1, \ldots, \mathcal{D}_n, \mathcal{D}$ are disjoint from each other, because we can consider isomorphic copies of these databases without affecting the answer to the Product Homomorphism Problem. Let $\mathcal{C}$ be the database instance consisting of the union of $\mathcal{D}_1, \ldots, \mathcal{D}_n$ and $\mathcal{D}$, extended with the facts $R(a_i, x)$ for all $i \leq n$ and $x \in \text{adom}(\mathcal{D}_i)$, and $R(b, x)$ for all $x \in \text{adom}(\mathcal{D})$, where $a_1, \ldots, a_n$ and b are fresh elements. Let $S = \{a_1, \ldots, a_n\}$. Our claim is that $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n \to \mathcal{D}$ if and only if S is not definable inside $\mathcal{C}$ by a conjunctive query.

For the forward direction, assume that there is $h : \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n \to \mathcal{D}$, and assume for a contradiction that S is definable by a conjunctive query q, i.e. $S = q(\mathcal{C})$. By assumption, and by the Chandra-Merlin theorem, $(\mathcal{D}_q, x) \to (\mathcal{C}, a)$ for all $a \in S$. Besides, by Proposition 2.1.4, this is equivalent to $(\mathcal{D}_q, x) \to \prod_{a \in S}(\mathcal{C}, a)$. Now, we want to show that there is a homomorphism $h' : \prod_{a \in S}(\mathcal{C}, a) \to (\mathcal{C}, b)$. For simplicity, assume that we only deal with binary vectors, i.e. that all elements h' has to deal with are pairs. We define h' in the following way:

$h'((x_1, \ldots, x_n)) = h((x_1, \ldots, x_n))$ if $(x_1, \ldots, x_n)$ belongs to $\text{adom}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$
$h'((x_1, \ldots, x_n)) = b$ if $(x_1, \ldots, x_n) = (a_1, \ldots, a_n)$.
$h'((x_1, \ldots, x_n)) = x_1$ if $(x_1, \ldots, x_n)$ is not of the above form.

This is a well-defined mapping. We now have to show that h' is a homomorphism, i.e. we have to show that h' it preserves all facts. Take an arbitrary fact $R((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ contained in $(\mathcal{C}, a_1) \otimes \cdots \otimes (\mathcal{C}, a_n)$. We have three cases:

1. Both $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ belong to $\text{adom}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$. By the way h' is defined, we realize that $R((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ is mapped to $R(h((x_1, \ldots, x_n)), h((y_1, \ldots, y_n)))$. Does the latter fact belong to $(\mathcal{C}, b)$? Yes, as otherwise h would be a wrong homomorphism (recall that $\mathcal{D} \subseteq \mathcal{C}$).

2. $(x_1, \ldots, x_n)$ belongs to $\text{adom}(\mathcal{C} \otimes \cdots \otimes \mathcal{C} \setminus \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$ while $(y_1, \ldots, y_n)$ belongs to $\text{adom}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$. This implies that $x_1, \ldots, x_n = a_1, \ldots, a_n$. Consequently, and by the way $\mathcal{C}$ is constructed, $(y_1, \ldots, y_n)$ cannot be a distinguished element. As a result, we observe that h' maps $R((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ to $R(b, h((y_1, \ldots, y_n)))$, which is contained in $(\mathcal{D}, b)$.

3. Both $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ belong to $\text{adom}(\mathcal{C} \otimes \cdots \otimes \mathcal{C} \setminus \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$. By the way h' is defined, $R((x_1, \ldots, x_n), (y_1, \ldots, y_n))$ is mapped to $R(x_1, y_1)$. This latter fact indeed belongs to $(\mathcal{C}, b)$.

The case distinction above indeed covers all possible cases, as we cannot have the following (remaining) scenario: $(x_1, \ldots, x_n) \in \text{adom}(\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$ and $(y_1, \ldots, y_n) \in \text{adom}(\mathcal{C} \otimes \cdots \otimes \mathcal{C} \setminus \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n)$. In fact, because we also have facts $R(x_1, x_2), \ldots, R(x_{n-1}, x_n)$ and $R(y_1, y_2), \ldots, R(y_{n-1}, y_n)$, we would get a contradiction with our assumption that $\mathcal{D}_1, \ldots, \mathcal{D}_n$ and $\mathcal{D}$ are disjoint. Take the first fact: it has to belong to one of the database instances $\mathcal{D}_1, \ldots, \mathcal{D}_n$ or $\mathcal{D}$ (note that of course $(x_1, \ldots, x_n)$ cannot be a distinguished element). But then this implies that two database instances in the above list share an element.

After composing together the respective homomorphisms, we can claim that $(\mathcal{D}_q, x) \to (\mathcal{C}, b)$, which implies that $S \neq q(\mathcal{C})$. Contradiction.

For the backward direction, assume that there is no map h such that $h : \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n \twoheadrightarrow \mathcal{D}$. We claim that we can define a query q defining S as follows: first of all we take $q_1 = \exists y_1, \ldots, y_k \psi(y_1, \ldots, y_k)$ to be the canonical Boolean CQ of $\mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n$ (where "Boolean" means that it contains no free variables). Then, we define $q(x)$ to be the unary CQ

$$\exists y_1 \dots y_k \exists x_1 \dots x_{r+1}(R(x, y_1) \wedge \dots \wedge R(x, y_k) \wedge \psi(y_1, \dots, y_k) \\ \wedge R(x, x_1) \wedge \dots \wedge R(x_r, x_{r+1}))$$

By construction, $q(\mathcal{C})$ contains all elements of S and does not contain b (in fact, in $q(x)$ we put the canonical query of $\mathcal{D}_1 \otimes \dots \otimes \mathcal{D}_n$ as a conjunct). Furthermore, we need to show that $q(\mathcal{C})$ contains *exactly* S. To see this, first note that – by inspection of Theorem $1(3)$ in [1] – we can assume that in each database instance $\mathcal{D}_1, \dots, \mathcal{D}_n, \mathcal{D}$, the maximum length of a directed path is precisely r, for some fixed $r \in \mathbb{N}$. Moreover, observe that each a_i and also b have an outgoing path of length $r + 1$, while no other elements have an outgoing path of length $r + 1$. It is then easy to see that, by the way $q(x)$ is constructed, $q(\mathcal{C})$ only selects element with an outgoing path of length $r+1$, i.e. precisely S. This was to show. □

In light of this result, and having previously showed the upper bound, we can claim that:

Corollary 3.2.4 ([8])**.** *The Query-by-Example problem for CQs is* **coNExpTime***-complete.*

3.2.2 Query-by-Example for UCQs

We now turn to the Query-by-Example problem for union of conjunctive queries (UCQs).

QUERY-BY-EXAMPLE FOR UCQS	
Input:	A database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q of $\mathcal{D}$, i.e. a UCQ q such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

As shown in [8], this problem is in **coNP**. To see why, we first introduce the "*QBE test for UCQs.*" This test takes as input a database instance $\mathcal{D}$ and relations of positive and negative examples S^+ and S^- over $\mathcal{D}$. It accepts if and only if for all combinations of positive examples $\overline{a} \in S^+$ and negative examples $\overline{b} \in S^-$, it happens that $(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$.

In addition, as we have seen for CQs, we have:

Proposition 3.2.5 ([8]). *Given a database instance $\mathcal{D}$ as well as relations S^+ and S^- over it, there is a fitting UCQ $q = q_1 \vee \cdots \vee q_n$ (i.e. a union of conjunctive queries q such that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$) if and only if the QBE test for UCQs accepts $\mathcal{D}$, S^+ and S^-.*

Proof.

Fix a database instance $\mathcal{D}$ as well as relations S^+ and S^- over it. For the forward direction, assume towards a contradiction that there are tuples $\overline{a} \in S^+$ and $\overline{b} \in S^-$ such that there is a homomorphism $h_1 : (\mathcal{D}, \overline{a}) \to (\mathcal{D}, \overline{b})$. Now, by assumption, $q = q_1 \vee \cdots \vee q_n$ is a fitting UCQ, so $S^+ \subseteq q(\mathcal{D})$. Take the positive example $\overline{a}$ we previously considered: because $\overline{a} \in q(\mathcal{D})$, by the Chandra-Merlin theorem for UCQs there must be a q_i (for $1 \leq i \leq n$) such that there is a homomorphism $h_2 : (\mathcal{D}_{q_i}, \overline{x}) \to (\mathcal{D}, \overline{a})$. It then suffices to compose together h_1 and h_2 to get a contradiction with the assumption that the UCQ q is a fitting UCQ for the given input.

For the backward direction, we can see $(\mathcal{D}, \overline{a})$ for all $\overline{a} \in S^+$ as canonical databases of queries, and take the union of all these queries. This resulting UCQ is then a fitting query because, as we assumed that for all combinations of positive examples $\overline{a} \in S^+$ and negative examples $\overline{b} \in S^-$ it happens that $(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, by the Chandra-Merlin theorem for UCQs, we achieve the condition $q(\mathcal{D}) \cap S^- = \emptyset$. Furthermore, it is easy to check that we can map our UCQ to all the positive examples in S^+, thus – again with the help of the Chandra-Merlin theorem for UCQs – obtaining the other condition, i.e. that $q(\mathcal{D}) \subseteq S^+$, as wanted. □

Now, because we don't need to check the safety of the direct product, we avoid the exponential-time check from before. Furthermore, now the guessed homomorphisms are polynomial in the size of $\mathcal{D}$, so they can be checked in polynomial time. Thus, we obtain a **coNP** upper bound on the problem.

What about the lower bound for this problem? A proof that it is **coNP**-hard was already provided in [12], however, because the proof uses the formalism of graph query languages, which are not considered in this thesis, we decided to propose a new proof of the same result by reducing from the complement of the following known **NP**-complete problem ([14]):

GRAPH HOMOMORPHISM PROBLEM (HOM)	
Input:	Two directed graphs $G = (V(G), E(G))$ and $H = (V(H), E(H))$.
Question:	Is there a homomorphism h from G to H, i.e. a map that preserves all edges of G?

Theorem 3.2.6 ([12]). *The Query-by-Example problem for UCQs is* **coNP***-hard.*

Proof.

We reduce from the complement of the graph homomorphism problem ($\overline{\mathsf{HOM}}$). Given a pair of graphs (G, H) we get an input for the fitting problem for UCQs in the following way. We first get a database instance $\mathcal{D}$ where $\mathrm{adom}(\mathcal{D}) = (V(G) \cup V(H) \cup \{a, b\})$, where a and b are fresh elements. Moreover, we add facts $R(x_1, x_2)$ for every $(x_1, x_2) \in E(G) \cup E(H)$. Finally, we extend $\mathcal{D}$ with facts $R(a, x)$ for all $x \in V(G)$ and $R(b, x)$ for all $x \in V(H)$. We need to show that $(G, H) \in \overline{\mathsf{HOM}}$ if and only if $(\mathcal{D}, S^+ = \{a\}, S^- = \{b\})$ has a fitting UCQ q.

For the forward direction, fix an arbitrary pair (G, H) and assume that $(G, H) \in \overline{\mathsf{HOM}}$. Recall that, by Proposition 3.2.5, $(\mathcal{D}, S^+ = \{a\}, S^- = \{b\})$ has a fitting UCQ q if and only if $(\mathcal{D}, a) \nrightarrow (\mathcal{D}, b)$. Assume for a contradiction that there is a homomorphism h such that $h : (\mathcal{D}, a) \to (\mathcal{D}, b)$. We can show that h is also a map from G to H that preserves all edges. Take an arbitrary edge $(x_1, x_2) \in E(G)$. It is not difficult to realize that $(h(x_1), h(x_2)) \in E(H)$, as h will map (a, x_1) to $(b, h(x_1))$ and (a, x_2) to $(b, h(x_2))$, and so $(h(x_1), h(x_2))$ must be an edge of H. Contradiction with the assumption that $(G, H) \in \overline{\mathsf{HOM}}$. We can conclude that $(\mathcal{D}, S^+ = \{a\}, S^- = \{b\})$ has a fitting UCQ q.

For the backward direction, assume that $(\mathcal{D}, S^+ = \{a\}, S^- = \{b\})$ has a fitting UCQ q, which again by Proposition 3.2.5 implies that $(\mathcal{D}, a) \nrightarrow (\mathcal{D}, b)$. Assume for a contradiction that $(G, H) \notin \overline{\mathsf{HOM}}$, i.e. there is a homomorphism h such that $h : G \to H$. We now present a new homomorphism h' (constructed on the basis of h) such that $h' : (\mathcal{D}, a) \to (\mathcal{D}, b)$:

$$
\begin{aligned}
&h'(a) = b \\
&h'(b) = b \\
&h'(x) = h(x) \text{ if } x \in V(G) \\
&h'(x) = x \text{ if } x \in V(H)
\end{aligned}
$$

It remains to show that h' preserves all facts. First note that it suffices to show that $h' : \mathcal{D} \to \mathcal{D}$ because, by definition, h' always maps a to b. Take an arbitrary fact $R(x_1, x_2) \in \mathcal{D}$. We have four cases:

1. Both x_1 and x_2 belong to $V(G)$. Then h' gives us $R(h(x_1), h(x_2))$, which is of course in $\mathcal{D}$ because $(h(x_1), h(x_2)) \in E(H)$.
2. Both x_1 and x_2 belong to $V(H)$. Then h' gives us the same relation $R(x_1, x_2)$, which is in $\mathcal{D}$ by assumption.
3. $x_1 = a$ and $x_2 \in V(G)$. In this case, h' gives us $R(b, h(x_2))$, which is in $\mathcal{D}$ by construction (recall that $h(x_2) \in V(H)$).
4. $x_1 = b$ and $x_2 \in V(H)$. In this case, h' gives us $R(b, x_2)$, which is in $\mathcal{D}$ by assumption.

Because we proved that $h' : (\mathcal{D}, a) \to (\mathcal{D}, b)$ is a correct homomorphism we get a contradiction. Hence, we can conclude that $(G, H) \in \overline{\mathsf{HOM}}$. This completes the proof.

□

In light of this result, and having previously showed the upper bound for the same problem, we can claim that:

Corollary 3.2.7 ([8])**.** *The Query-by-Example problem for UCQs is* **coNP***-complete.*

4 Introducing constants

In this chapter we present the new results we were able to obtain; in particular, we introduce a new setting for allowing constants in the queries and we revisit some definitions already seen in Chapter 2. Besides, we revisit the Query-by-Example problem for CQs and UCQs in different ways. Then, on the technical side, we prove some complexity bounds of these problems.

4.1 High-level setting

In the literature on the Query-by-Example problem (for example in [8]) it is sometimes assumed that the queries cannot contain constants. This is, we believe, a shortcoming of prior work on the QBE problem that, in this thesis, we try to address. In Chapter 3 we studied the following two problems:

1. Query-by-Example problem for Conjunctive Queries (**coNExpTime**-complete).
2. Query-by-Example problem for Union of Conjunctive Queries (**coNP**-complete).

Now, even before formally define what it means to "allow constants" in the queries, there are some examples that may lead to think that constants are interesting to study in the context of the Query-by-Example problems. Consider the example 3.1 that we already introduced in Chapter 3:

WorksInDepartment	
Name	**Department**
Mark	Sales
John	Service
Alice	IT
Terence	Service

Assume that the positive example is Mark while the negatives are John, Alice and Terence. Furthermore, assume that we only have the binary relation "WorksInDepartment". Without the possibility of allowing constants in the queries it is not possible to construct a fitting query for this input (a way of formally proving this is via the QBE test, introduced in Chapter 3). However, if we do allow constants, then a possible fitting CQ is:

$$q(x) := WorksInDepartment(x, Sales)$$

Now, consider these three different ways of allowing constants in the queries:

a) Given an input database instance $\mathcal{D}$ for the Query-By-Example problem as well as relations of positive and negative examples S^+ and S^- over it, we ask if there is a fitting query q for $\mathcal{D}$, where q is allowed to contain constants.

b) Given an input database instance $\mathcal{D}$ for the Query-By-Example problem, a certain set $\mathcal{K}$ of constants as well as relations of positive and negative examples S^+ and S^- over $\mathcal{D}$, we ask if there is a fitting query q for $\mathcal{D}$, and q is allowed to contain constants belonging to the set of constants $\mathcal{K}$.

c) Given an input database instance $\mathcal{D}$ for the Query-By-Example, relations of positive and negative examples S^+ and S^- over it and a certain $k \in \mathbb{N}$, we ask if there is a fitting query q for $\mathcal{D}$ if we put a bound k on the number of constants that can appear in q.

While in a) we put no constraints on how the constants could appear in the queries, in b) and c) we have some. We can give examples to better understand the meaning of b) and c). Regarding b), let us think about *feature engineering* in machine learning. In feature engineering we have to handle a set of features to solve a certain problem and of course not all features are relevant if we want to find a solution to it.

A possible motivation for c) is that, if we work with unions of conjunctive queries and we don't have a bound on the number of constants appearing in them, then the fitting problem trivializes because we can just "mention" each positive example in our query (more on this at the end of this chapter). To continue with the parallel of feature engineering, this case would be the same as *overfitting*, i.e. allowing too many features to be used and thus causing our model to be useful only in reference to its initial data set.

4.2 Definitions

We now formally define the concepts we will need in the further sections, in light of the ideas we introduced in the previous section.

As a first step to take, let us partition our infinite set of values $\mathcal{V}$ that we introduced in Chapter 2 into two disjoint subsets: $\mathcal{C}$ and $\mathcal{N}$. We say that $\mathcal{C}$ is a set of *constants* while $\mathcal{N}$ is a set of *labelled nulls*. Of course, $\mathcal{V} = \mathcal{C} \cup \mathcal{N}$ (we will say a bit more about labelled nulls along the way and the end of this section). Most importantly, from now on we will consider database instances over $\mathcal{V} = \mathcal{C} \cup \mathcal{N}$. We use labelled nulls (conceptually akin to variables) because we want to preserve the conceptual distinction between variables and constants once we take the canonical database $\mathcal{D}_q$ of a query q, which may contain both variables and constants.

We then need to give a precise definition of the a conjunctive query containing constants:

Definition 4.2.1 (Conjunctive query containing constants)**.**

Given $k \geq 0$, a *k*-ary *conjunctive query* (CQ) q *containing constants* (over a schema σ) is an expression of the form:

$$q(\bar{t}) := \exists y_1, \ldots, y_m(\alpha_1 \wedge \cdots \wedge \alpha_n)$$

with $m \geq 0$ and $n \geq 1$. Moreover, $\bar{t} = t_1, \ldots, t_k$ is a sequence of variables and/or constants; each α_i is an atomic formula using a relation from σ and it may use variables from $\bar{t}$ as well as constants (and of course variables from $y_1, \ldots, y_m$). In addition, it is required that each variable/constant in $\bar{t}$ occurs in at least one conjunct α_i (this requirement is called *safety*). Lastly, note that the tuple $\bar{t}$ may contain repetitions.

An example of a CQ containing constant is $q(x, c) := \exists z R(x, z, c)$.

We also need to update the notion of homomorphism:

Definition 4.2.2 (C-homomorphism)**.**
Given databases $(\mathcal{D}_1, \overline{a})$ and $(\mathcal{D}_2, \overline{b})$, we say that a *c-homomorphism* h is a map from adom$(\mathcal{D}_1)$ to adom$(\mathcal{D}_2)$ such that for all $R \in \mathcal{D}_1$ and every tuple $\overline{a} = (a_1, \ldots, a_n) \in R^{\mathcal{D}_1}$, the tuple $h(\overline{a}) = (h(a_1), \ldots, h(a_n)) \in R^{\mathcal{D}_2}$. Regarding the distinguished elements, it must be the case that $h(\overline{a}) = \overline{b}$. Lastly, for a constant $a \in$ adom$(\mathcal{D}_1)$, it must be the case that $h(a) = a \in$ adom$(\mathcal{D}_2)$.

Note that we only added the final condition, about mapping constants to constants, to the previous definition of homomorphism. Essentially, the reason for this addition is that the Chandra-Merlin theorem relies on the fact that we can see homomorphisms as first-order valuations of variables, but a valuation can now evaluate a constant, which should map to that constant itself. As a result, if we do not add a further condition on homomorphisms, the Chandra-Merlin theorem would fail.

But the Chandra-Merlin theorem requires some clarity for the notion of canonical database and canonical query as well.

Definition 4.2.3 (C-canonical database)**.**
Given a CQ $q(\overline{t})$ containing constants, we say that $(\mathcal{D}_q, \overline{t})$ is its *C-canonical database*, where the facts of $\mathcal{D}_q$ are the conjuncts of q. Note though that we replace the variables occurring in q with labelled nulls whenever we take the canonical database of q.

As anticipated, we use labelled nulls to keep the distinction between variables and constants once we take the canonical database $\mathcal{D}_q$ of a query q, which may contain both variables and constants.

Definition 4.2.4 (C-canonical query)**.**
Given a database instance $(\mathcal{D}, \overline{a})$, for $\overline{a} = a_1, \ldots, a_k$, we can associate to it a k-ary *c-canonical conjunctive query* $q_c^{\mathcal{D}}(\overline{t})$. We can get this query in two steps:

1. Regarding the head of $\hat{q}_c^{\mathcal{D}}(\overline{t})$, we obtain $t_1, \ldots, t_k$, where each t_i is either a variable (if a_i is a labelled null) or directly the constant a_i (if a_i is a constant). The resulting tuple $t_1, \ldots, t_k$ is then our $\overline{t}$ in $\hat{q}_c(\overline{t})$.

2. The body of $\hat{q}_c^{\mathcal{D}}(\overline{t})$ is just the conjunction of all the facts in $(\mathcal{D}, \overline{a})$. Every labelled null that may occur in a fact of the database instance gets replaced with a variable.

As explained, we are allowed to keep the Chandra-Merlin theorem for both CQs and UCQs, even for the case with constants:

Theorem 4.2.1 (Chandra-Merlin, with constants)**.**
Given a CQ $q(\bar{t})$ and a database instance $\mathcal{D}$, the following holds:

$$\bar{a} \in q(\mathcal{D}) \text{ if and only if } (\mathcal{D}_q, \bar{t}) \to (\mathcal{D}, \bar{a}).$$

Theorem 4.2.2 (Chandra-Merlin for UCQs, with constants)**.**
Given a UCQ $q = q_1 \vee \cdots \vee q_n$ and a database instance $\mathcal{D}$, the following holds:

$$\bar{a} \in q(\mathcal{D}) \text{ if and only if } (\mathcal{D}_{q_i}, \bar{t}) \to (\mathcal{D}, \bar{a}) \text{ for some } 1 \leq i \leq n.$$

We also need to slightly update the notion of direct product of database instances:

Definition 4.2.5 (C-direct product)**.**
A *c-direct product* is entirely similar to a direct product, with the following requirement: we replace each value of the form $(c, \ldots, c)$ in the active domain of the product with a constant c and we regard all the other values (i.e. all the values that are not of the form $(c, \ldots, c)$ for a fixed constant c) as nulls.

Note that this way of defining the direct product operation when we have both constants and labelled nulls is actually not new: it was already used, for example, in [6] (see Proposition 5).

We now prove the same results about products we introduced in Chapter 2. This is needed for further proofs and in order to show that indeed the direct product operation "$\otimes$" defines the least upper bound in the lattice of databases defined by the notion of homomorphism. The following two propositions are more general then their similar versions in Chapter 2 (i.e. Proposition 2.1.3 and Proposition 2.1.4, respectively), because we assume that our database instances can now contain constants and labelled nulls and moreover we employ the notion of c-homomorphism, c-canonical database and c-direct product. Being more general, the proofs we provide for them are suitable for the case without constants, in that we can just replace all constants with nulls in those cases.

Proposition 4.2.3. *Given database instances $(\mathcal{D}_1, \bar{a}_1), \ldots, (\mathcal{D}_n, \bar{a}_n)$, the following holds:*

$$\prod_{i \leq n} (\mathcal{D}_i, \bar{a}_i) \to (\mathcal{D}_i, \bar{a}_i) \text{ for each } i \leq n.$$

Proof.

Fix database instances $(\mathcal{D}_1, \overline{a}_1), \ldots, (\mathcal{D}_n, \overline{a}_n)$. To keep the proof simple and intuitive we only consider the case when $n = 2$ and thus our direct product has the shape $\prod_{i \leq 2}(\mathcal{D}_i, \overline{a}_i) = (\mathcal{D}_1, \overline{a}_1) \otimes (\mathcal{D}_2, \overline{a}_2)$. Consider the following c-homomorphisms h_1 and h_2, where $h_1 : (\mathcal{D}_1 \otimes \mathcal{D}_2, \overline{a}_1 \otimes \overline{a}_2) \to (\mathcal{D}_1, \overline{a}_1)$ and $h_2 : (\mathcal{D}_1 \otimes \mathcal{D}_2, \overline{a}_1 \otimes \overline{a}_2) \to (\mathcal{D}_2, \overline{a}_2)$:

$$\left.\begin{aligned} h_1((x_1, x_2)) &= x_1 \\ h_2((x_1, x_2)) &= x_2 \end{aligned}\right\} \text{ where } (x_1, x_2) \text{ is a labelled null.}$$

$$\left.\begin{aligned} h_1(c) &= c \\ h_2(c) &= c \end{aligned}\right\} \text{ where } c \text{ is a constant.}$$

The above map relies on the following (intuitive) observation: every fact in $(\mathcal{D}_1 \otimes \mathcal{D}_2, \overline{a}_1 \otimes \overline{a}_2)$ has the shape $R((x_1, x_1'), (x_2, x_2'), \ldots, (x_n, x_n'))$ for facts $R(x_1, x_2, \ldots, x_n) \in (\mathcal{D}_1, \overline{a}_1)$ and $R(x_1', x_2', \ldots, x_n') \in (\mathcal{D}_2, \overline{a}_2)$ and for a relation $R \in \sigma$.

We now have to show that indeed h_1 and h_2 preserve all facts. We will only look at h_1, because the reasoning for h_2 is entirely symmetric. For now, assume that our arbitrary fact only contains labelled nulls, thus being of the form $R((x_1, x_1'), (x_2, x_2'), \ldots, (x_n, x_n'))$. We will deal with constants later. Note that h_1 maps $R((x_1, x_1'), (x_2, x_2'), \ldots, (x_n, x_n'))$ to $R(x_1, x_2, \ldots, x_n)$, and indeed the latter fact is contained in $(\mathcal{D}_1, \overline{a}_1)$. Furthermore, it is easy to see that, whenever we have a distinguished element from $\overline{a}_1 \otimes \overline{a}_2$, it is mapped to the correct distinguished element in $\overline{a}_1$. Simply take an arbitrary distinguished element $(a_1, a_2) \in \overline{a}_1 \otimes \overline{a}_2$; see that $a_1 \in \overline{a}_1$ while $a_2 \in \overline{a}_2$. Besides, h_1 maps (a_1, a_2) to a_1, and this is the correct distinguished element.

Note that if any element occurring in our arbitrary fact is instead a constant, the facts is still correctly preserved. As an example, let us consider the following fact $R((x_1, x_2), c)$ where c is a constant. Besides, let's assume that we have to map it to $(\mathcal{D}_1, \overline{a}_1)$. See that in the latter database instance we must have a fact $R(x_1, c)$, which is the correct fact to which we have to map $R((x_1, x_2), c)$.

This completes the proof.

□

And then we take care of the second proposition:

Proposition 4.2.4. *Given database instances $(\mathcal{D}_1, \overline{a}_1), \ldots, (\mathcal{D}_n, \overline{a}_n)$ and a database instance $(\mathcal{D}, \overline{b})$, the following are equivalent:*

- $(\mathcal{D}, \overline{b}) \to \prod_{i \leq n}(\mathcal{D}_i, \overline{a}_i)$.

- $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_i, \bar{a}_i)$ *for each* $i \leq n$.

Proof.

Fix database instances $\mathcal{D}_1, \mathcal{D}_2$ and n-ary relations S_1, S_2 over $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively; we also keep the same constraints on products of Proposition 4.2.3. We further assume, for reasons of clarity, that our schema σ only contains relations of arity 2.

For the forward direction, assume that there is a homomorphism h_1 such that $h_1 : (\mathcal{D}, \bar{a}) \to (\mathcal{D}_1, \bar{a}_1) \otimes (\mathcal{D}_2, \bar{a}_2)$. By Proposition 4.2.3 it follows that there are homomorphisms h_2 and h_3 such that $h_2 : (\mathcal{D}_1, \bar{a}_1) \otimes (\mathcal{D}_2, \bar{a}_2) \to (\mathcal{D}_1, \bar{a}_1)$ and that $h_3 : (\mathcal{D}_1, \bar{a}_1) \otimes (\mathcal{D}_2, \bar{a}_2) \to (\mathcal{D}_2, \bar{a}_2)$. Hence, by first composing h_1 with h_2 we can claim that $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_1, \bar{a}_1)$ and by composing $h1$ with h_3 we can claim that $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_2, \bar{a}_2)$. As a result, we get that $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_i, \bar{a}_i)$ for each $i \leq 2$, as wanted.

For the backward direction, assume that $(\mathcal{D}, \bar{b}) \to (\mathcal{D}_i, \bar{a}_i)$ for each $i \leq 2$. We propose a c-homomorphism h_3 from $(\mathcal{D}, \bar{b})$ to $\prod_{i \leq 2}(\mathcal{D}_i, \bar{a}_i)$. By assumption, we know that there must be c-homomorphisms h_1 and h_2 of the form:

$$h_1 : (\mathcal{D}, \bar{b}) \to (\mathcal{D}_1, \bar{a}_1)$$
$$h_2 : (\mathcal{D}, \bar{b}) \to (\mathcal{D}_2, \bar{a}_2)$$

In the following, we describe the map h_3, given an arbitrary value $x \in \text{adom}(\mathcal{D})$:

$$h_3(x) = (h_1(x), h_2(x)) \text{ if } x \text{ is a labelled null.}$$
$$h_3(x) = x \text{ if } x \text{ is a constant.}$$

We now show that h_3 is indeed a correct c-homomorphism as it preserves all facts. Take an arbitrary fact $R(x_1, x_2) \in (\mathcal{D}, \bar{b})$, for $R \in \sigma$. As for Proposition 4.2.3, assume that our arbitrary fact only contains labelled nulls; we will deal with constants later. By the way we defined h_3, $R(x_1, x_2)$ gets mapped to $R((h_1(x_1), h_2(x_1)), (h_1(x_2), h_2(x_2)))$. It is easy to check that indeed $R((h_1(x_1), h_2(x_1)), (h_1(x_2), h_2(x_2)))$ belongs to $\prod_{i \leq 2}(\mathcal{D}_i, \bar{a}_i)$. As for the distinguished elements, h_3 correctly maps them. To see this, consider an arbitrary element $b_i \in \bar{b}$. By the way we defined h_3, $(h_1(a), h_2(a))$ must be the correct distinguished element, as otherwise we would contradict our assumption (i.e. the assumed homomorphisms h_1 and h_2 would be incorrect).

Note that if any element occurring in our arbitrary fact is instead a constant, the fact is still correctly preserved. As an example, let us consider the

following fact $R(x, c)$ where c is a constant. Now, by assumption we have that

$$R(x, c) \mapsto R(h_1(x), h_1(c) = c)$$
$$R(x, c) \mapsto R(h_2(x), h_2(c) = c)$$

Thus, in the direct product we will have $R((h_1(x), h_2(x)), c)$. It is then easy to check that h_3 correctly maps $R(x, c)$ to it, and that it works in general for all four cases considered above.

This completes the proof.

□

As a last note before diving into the new results of this thesis, it may be interesting to have a digression on the concept of *labelled null*. In this thesis, the distinction between constants and labelled nulls is used purely as a theoretical construction used to define the notion of *canonical instance* for queries with constants. However, labelled nulls arise in practice in some settings like data exchange. In this setting we have a *source schema* $\mathbf{S}$, a *target schema* $\mathbf{T}$, a set of *source-to-target dependencies* Σ_{st} and a set of *target dependencies* Σ_t. The set Σ_{st} is essentially a set of constraints connecting $\mathbf{S}$ to $\mathbf{T}$. The aim is to see if it possible to construct a target instance J satisfying Σ_{st}. Sometimes though the constraints are incomplete, meaning that we are free to instantiate the constraint as we want. In that case, when constructing J, instead of those values, we put *labelled nulls*, that are – in this sense – very similar to variables. More formally, suppose that Σ_{st} contains the following constraint:

$$R(x, y) \rightarrow \exists z S(x, y, z)$$

Then, we can get many different possible resulting databases J because we can think of facts $S(a, b, c_1), S(a, b, c_2), S(a, b, c_3)$ and so on. But if, instead of constants c_1, c_2 and c_3, we put a labelled null N, we can get a "minimal" solution that contains $S(a, b, N)$.

4.3 Upper bounds for CQs and UCQs

With all these definitions and propositions in hand, we now want to find the upper bound of some complexity problems. The following problems are essentially based on the discussion of problems a), b) and c) that we had in Section 4.1, plus a "general" version in which the database instance

may contain constants as well as labelled nulls; in the last three problems we will use the expression "ground database instance" to refer to a database instance that only contains constants, and all these constants come from the previously introduced set $\mathcal{C}$. In this thesis we are primarily interested in ground instances, but it is convenient to define a "general" problem as an intermediate problem to reduce to.

We will first look at problems for conjunctive queries and, at the end of this section, we will introduce similar problems but for unions of conjunctive queries, and we will discuss them.

General Query-by-Example for CQs	
Input:	A database instance $\mathcal{D}$ with values from $\mathcal{V} = \mathcal{C} \cup \mathcal{N}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting CQ q (possibly containing constants) for $\mathcal{D}$, i.e. a CQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

Free Query-by-Example for CQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting CQ q (possibly containing constants) for $\mathcal{D}$, i.e. a CQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

Limited Query-by-Example for CQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$, and a subset $\mathcal{C}' \subseteq \mathcal{C}$.
Question:	Is there a fitting CQ q (possibly containing constants) for $\mathcal{D}$, i.e. a CQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$, such that q can only contain constants coming from a subset $\mathcal{C}' \subseteq \mathcal{C}$?

Bounded Query-by-Example for CQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$, and a certain $k \in \mathbb{N}$.
Question:	Is there a fitting CQ q (possibly containing constants) for $\mathcal{D}$, i.e. a CQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$, such that q can contain at most k constant/s from $\mathcal{C}$, for a certain $k \in \mathbb{N}$?

To settle an upper bound for the General Query-by-Example problem we will try to follow the same path we took in Chapter 3, thus revisiting the Query-by-Example test.

Considering CQs, the QBE test is: we have as input a ground database instance $\mathcal{D}$ plus two relations S^+ and S^- over it. It accepts if and only if:

1. $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ is safe.
2. $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$

Of course, we have to keep in mind all the new definitions of Chapter 4, that are used implicitly in the test and in the remaining of the thesis.

Proposition 4.3.1. *Given a database instance $\mathcal{D}$ as well as relations S^+ and S^-, there is a fitting conjunctive query possibly containing constants (i.e. a query q, possibly containing constants, such that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$) if and only if the QBE test accepts $\mathcal{D}$ and both S^+ and S^-.*

Proof.
The proof of Proposition 3.2.1 applies here as well, for we still have the propositions used there, this time considering constants (i.e. Proposition 4.2.3 and Proposition 4.2.4) and the Chandra-Merlin theorem for CQs also holds if we allow constants in the queries, as explained. □

Thanks to this last proposition, and following the same route we considered in Chapter 3, we can show that the upper bound of the General Query-by-Example problem for CQs is again in **coNExpTime**. Besides, this allows us to settle the upper bound for the Free Query-by-Example problem as well:

Theorem 4.3.2. *The Free and the General Query-by-Example problem for CQs are in* **coNExpTime**.

In fact, the Free Query-by-Example problem is a special case of the General problem, where the database instance only contains constants, i.e. the input for the Free-Query-by-Example problem is a valid input for the general problem as well.

We now prove that the upper bound for the Limited Query-by-Example problem for CQs is **coNExpTime**. To this end, we will reduce it to the General Query-by-Example problem.

Theorem 4.3.3. *The Limited Query-by-Example problem for CQs is in* **coNExpTime**.

Proof.

We reduce the Limited Query-by-Example problem for CQs to the general problem, whose upper bound is **coNExpTime** (Proposition 4.3.1). Take an input of the former problem $(\mathcal{D}_1, S_1^+, S_1^-, \mathcal{C}')$, where $\mathcal{C}' \subseteq \mathcal{C}$. We show how to reduce this input to an input $(\mathcal{D}_2, S_2^+, S_2^-)$ for the general problem. Firstly, we see $\mathcal{D}_1$ as set of facts $\{R_1(\bar{t}_1), \ldots, R_n(\bar{t}_n)\}$, and we define $\mathcal{D}_2 = \{R_1'(\bar{t}_1'), \ldots, R_n'(\bar{t}_n')\}$, where each $R_i'(\bar{t}_i') \in \mathcal{D}_2$ is obtained from $R_i(\bar{t}_i) \in \mathcal{D}_1$ in the following way:

1. $R_i = R_i'$.
2. Take $t_i = c_1, \ldots, c_k$. Note that either $c_i \in \mathcal{C}'$ or $c_i \in \mathcal{C} \setminus \mathcal{C}'$. We show how to obtain $t_i' = c_1', \ldots, c_k'$. If $c_i \in \mathcal{C}'$, then $c_i' = c_i$ and if $c_i \in \mathcal{C} \setminus \mathcal{C}'$ then $c_i = n_i \in \mathcal{N}$.

S_2^+ and S_2^- are then obtained accordingly. It is also easy to check that the so-obtained input $(\mathcal{D}_2, S_2^+, S_2^-)$ is a valid input for the general problem. We now need to show that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting CQ q if and only if $(\mathcal{D}_2, S_2^+, S_2^-)$ does.

For the forward direction, assume that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting CQ $q(\bar{t})$, i.e. $S_1^+ \subseteq q(\mathcal{D}_1)$ and $S_1^- \cap q(\mathcal{D}_1) = \emptyset$. We want to show that $S_2^+ \subseteq q(\mathcal{D}_2)$ and $S_2^- \cap q(\mathcal{D}_2) = \emptyset$. For the positive examples S_2^+, by assumption we know that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{a})$ for all $\bar{a} \in S_1^+$. It follows that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{b})$ for all $\bar{b} \in S_2^+$, because, by construction, $\mathrm{adom}(\mathcal{D}_2)$ contains the constants from $\mathcal{C}'$ that are present in $\mathrm{adom}(\mathcal{D}_1)$ and nulls for every constant in $\mathcal{C}$ appearing in $\mathcal{D}_1$. Moreover, $\mathcal{D}_q$ only contains constants coming from $\mathcal{C}'$.

For the negative examples, assume towards a contradiction that there is $\bar{b} \in S_2^-$ such that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{b})$. It is easy to see that, by construction, we can reuse the same homomorphism to claim that there is $\bar{a} \in S_1^-$ such that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{a})$. Contradiction. We can conclude that $S_2^- \cap q(\mathcal{D}_2) = \emptyset$.

For the backward direction, let us assume that $(\mathcal{D}_2, S_2^+, S_2^-)$ has a fitting CQ $q(\bar{t})$, i.e. $S_2^+ \subseteq q(\mathcal{D}_2)$ and $S_2^- \cap q(\mathcal{D}_2) = \emptyset$. We want to show that $S_1^+ \subseteq q(\mathcal{D}_1)$ and $S_1^- \cap q(\mathcal{D}_1) = \emptyset$. For the positive examples S_2^+, by assumption we know that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{a})$ for all $\bar{a} \in S_2^+$. Analogously to what we saw for the forward direction, we can reuse the same homomorphisms to claim that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{b})$ for all $\bar{b} \in S_1^+$. The only difference is that of course now the homomorphisms map nulls in $\mathcal{D}_q$ to constants (belonging to $\mathcal{C}$) in $\mathcal{D}_1$.

For the negative examples, assume for a contradiction that there is $\bar{a} \in S_1^-$ such that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{a})$. It is easy to see that, by construction, we

can reuse the same homomorphism to claim that there is $\bar{b} \in S_2^-$ such that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{b})$. Contradiction. We can conclude that $S_1^- \cap q(\mathcal{D}_1) = \emptyset$.

This completes the proof. □

We now look at the same problems, but this time we consider UCQs. Hence, now the problems are the following four:

General Query-by-Example for UCQs	
Input:	A database instance $\mathcal{D}$ with values from $\mathcal{V} = \mathcal{C} \cup \mathcal{N}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q (possibly containing constants) for $\mathcal{D}$, i.e. a UCQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

Free Query-by-Example for UCQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q (possibly containing constants) for $\mathcal{D}$, i.e. a UCQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$?

Limited Query-by-Example for UCQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q (possibly containing constants) for $\mathcal{D}$, i.e. a UCQ q such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$, such that q can only contain constants coming from a subset $\mathcal{C}' \subseteq \mathcal{C}$?

Bounded Query-by-Example for UCQs	
Input:	A ground database instance $\mathcal{D}$ and relations S^+, S^- over $\mathcal{D}$.
Question:	Is there a fitting UCQ q (possibly containing constants) for $\mathcal{D}$, i.e. a UCQ q (possibly containing constants) such that $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$, and such that q can contain at most k constant/s from $\mathcal{C}$, for a certain $k \in \mathbb{N}$?

Following the idea already presented in 3, we propose a new Query-by-Example test for UCQs. This test takes as input a database instance $\mathcal{D}$ and relations of positive and negative examples S^+ and S^- over $\mathcal{D}$. It accepts if and only if for all combinations of positive examples $\bar{a} \in S^+$ and negative examples $\bar{b} \in S^-$, it happens that $(\mathcal{D}, \bar{a}) \nrightarrow (\mathcal{D}, \bar{b})$.

In this case we just need to consider the following proposition:

Proposition 4.3.4. *Given a database instance $\mathcal{D}$ as well as relations S^+ and S^- over it, there is a fitting UCQ $q = q_1 \vee \cdots \vee q_n$ possibly containing constants (i.e. a union of conjunctive queries q, whose CQs can contain constants, such that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$) if and only if the QBE test for UCQs accepts $\mathcal{D}, S^+$ and S^-.*

Proof.
The proof of Proposition 3.2.5 applies here as well, for we still have the Chandra-Merlin theorem for UCQs, as explained. □

Thanks to this last proposition, and so following the same route we considered in Chapter 3, we can claim that the upper bound of the General Query-by-Example problem for UCQs is again **coNP**.

Theorem 4.3.5. *The General Query-by-Example problem for UCQs is in* **coNP**.

If now we look at the upper bound of the Free Query-by-Example problem for UCQs, we see that the problem trivializes:

Theorem 4.3.6. *The Free Query-by-Example problem for UCQs is in* **P**.

Proof.
Fix an input database instance $\mathcal{D} = \{R_1(\overline{a}_1), \ldots, R_k(\overline{a}_k)\}$ and relations of positive and negative examples S^+ and S^- (respectively) over it. Because $\mathcal{D}$ only contains constants, also the positive and negative examples S^+ and S^- will only contain constants. We now present a procedure, running in polynomial time, that decides the problem. First check if S^+ and S^- are disjoint. If they are not, the input does not have a fitting UCQ. If however, $S^+ \cap S^- = \emptyset$, we show how to construct a fitting UCQ q. For simplicity, consider the case of unary UCQs. Assume that we have positive examples $S^+ = \{a_1, \ldots, a_n\}$ and negative examples $S^- = \{b_1, \ldots, b_m\}$. We first construct CQs q_i of the following form

$$q_i(a_i) := R_1(\overline{a}_1) \wedge \cdots \wedge R_k(\overline{a}_k)$$

for $1 \leq i \leq n$. We then say that a fitting UCQ for $\mathcal{D}$ is:

$$q = q_i \vee \cdots \vee q_n$$

It is then trivial to check that indeed $S^+ \subseteq q(\mathcal{D})$ (actually, it is the case that $S^+ = q(\mathcal{D})$) and it also follows that $S^- \cap q(\mathcal{D}) \neq \emptyset$, because we assumed that S^+ and S^- are disjoint. This was to show. □

Regarding the Limited Query-by-Example problem for UCQs, we can prove that:

Theorem 4.3.7. *The Limited Query-by-Example problem for UCQs is in* **coNP**.

Proof.

We reduce the Limited Query-by-Example problem for UCQs to the general problem, whose upper bound is **coNExpTime** (Theorem 4.3.5). Take an input of the former problem $(\mathcal{D}_1, S_1^+, S_1^-, \mathcal{C}')$, where $\mathcal{C}' \subseteq \mathcal{C}$. We will use the same reduction we saw for Theorem 4.3.3. We now need to show that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting UCQ q if and only if $(\mathcal{D}_2, S_2^+, S_2^-)$ does. We observe that the proof of Theorem 4.3.3 goes through in this case as well; the only difference is that we have to exploit the Chandra-Merlin theorem for UCQs instead of the one for CQs. □

4.4 Lower bounds for CQs and UCQs

Having established the upper bounds for the fitting problems for CQs and UCQs in the previous section, we turn now to the lower bounds for the Free Query-by-Example problem, Limited Query-by-Example problem and Bounded Query-by-Example problem, and then we will turn to UCQs.

4.4.1 Lower bounds for CQs

We will now provide a proof of the lower bound for the Free Query-by-Example problem for CQs and we will see that, by the way the proof is constructed, it also applies to Limited Query-by-Example and Bounded Query-by-Example.

We start by proving the following two lemmas:

Lemma 4.4.1. *Given a database instance $\mathcal{D}$ and non-empty relations of positive and negative examples S^+ and S^- (respectively) over it, if $\mathcal{D}$ only contains labelled nulls and q is a fitting CQ for $(\mathcal{D}, S^+, S^-)$, then q does not contain constants.*

Proof.

Fix an arbitrary database instance $\mathcal{D}$ and non-empty relations of positive and negative examples S^+ and S^- (respectively) over it, and assume by contraposition that q contains a constant a. We want to show that $\mathcal{D}$ contains the very same constants a, thus falsifying the claim that it contains only labelled nulls. Observe that, because we assumed that q is a fitting CQ for $(\mathcal{D}, S^+, S^-)$, and because our homomorphisms map constant to constants, the constants a must in fact appear in $\mathcal{D}$. □

For the second lemma we also need to introduce the notion of *c-connected* CQs. We will first define c-connectedness of a database instance through the notion of *incidence graph*, so we first need to introduce:

Definition 4.4.1 (Incidence graph)**.**
Given a database instance $(\mathcal{D}, \overline{a})$, the *incidence graph* of $\mathcal{D}$ is the bipartite multi-graph containing all elements of the active domain of $\mathcal{D}$ as well as the facts of $\mathcal{D}$, and an edge (a, f) whenever the element a occurs in the fact f.

Definition 4.4.2 (Connected graph)**.**
An graph $G = (V(G), E(G))$ is *connected* if every pair of elements in $V(G)$ are connected, i.e. there is a path between every pair of vertices.

Definition 4.4.3 (C-connected database instance)**.**
We say that a database instance $(\mathcal{D}, \overline{a})$ is *c-connected* if every connected component of its incidence graph contains at least one distinguished element.

Note that this definition of c-connectedness is meaningful only if $k > 0$, i.e. there is at least one distinguished element in the active domain of the database instance we are considering. Furthermore, c-connectedness is weaker than connectedness as, in the case of c-connectedness, there is no guarantee that the distinguished elements are connected to each other. Finally, it is easy to see that, if we have only one distinguished element in the active domain, c-connectedness is the same as connectedness.

Definition 4.4.4 (C-connected CQ)**.**
We say that a CQ $q(\overline{t})$ is *c-connected* if its canonical database $(\mathcal{D}_q, \overline{t})$ is.

With these notions in hand, we can finally prove the following lemma:

Lemma 4.4.2. *Given a c-connected CQ $q(\overline{t})$ and disjoint database instances $\mathcal{D}_1$ and $\mathcal{D}_2$, if there is a homomorphism $h : (\mathcal{D}_q, \overline{t}) \to (\mathcal{D}_1 \uplus \mathcal{D}_2, \overline{a})$ (for $\overline{a} \in$ adom$(\mathcal{D}_1 \uplus \mathcal{D}_2)$), then either $h : (\mathcal{D}_q, \overline{t}) \to (\mathcal{D}_1, \overline{a})$ or $h : (\mathcal{D}_q, \overline{t}) \to (\mathcal{D}_2, \overline{a})$.*

Proof.
Fix a c-connected CQ $q(\overline{t})$ and disjoint database instances $\mathcal{D}_1$ and $\mathcal{D}_2$. Assume that there is a homomorphism $h : (\mathcal{D}_q, \overline{t}) \to (\mathcal{D}_1 \uplus \mathcal{D}_2, \overline{a})$ (for $\overline{a} \in$ adom$(\mathcal{D}_1 \uplus \mathcal{D}_2)$) and, for a contradiction, assume that $h : (\mathcal{D}_q, \overline{t}) \nrightarrow (\mathcal{D}_1, \overline{a})$ and $h : (\mathcal{D}_q, \overline{t}) \nrightarrow (\mathcal{D}_2, \overline{a})$. See that the latter assumption, as well as the assumption that $h : (\mathcal{D}_q, \overline{t}) \to (\mathcal{D}_1 \uplus \mathcal{D}_2, \overline{a})$, allow us to claim that adom$(\mathcal{D}_q)$ can be partitioned into the following two (non-empty) sets:

$$\text{adom}(\mathcal{D}'_q) = \{a \mid a \in \text{adom}(\mathcal{D}_q) \text{ and } h(a) \in \text{adom}(\mathcal{D}_1)\}$$
$$\text{adom}(\mathcal{D}''_q) = \{a \mid a \in \text{adom}(\mathcal{D}_q) \text{ and } h(a) \in \text{adom}(\mathcal{D}_2)\}$$

Note that all the distinguished variables $\bar{t}$ must either be in $\text{adom}(\mathcal{D}'_q)$ or in $\text{adom}(\mathcal{D}''_q)$, as otherwise we would have to map $\mathcal{D}_q$ entirely into one of the two databases $\mathcal{D}_1$ and $\mathcal{D}_2$, contradicting our assumptions. Moreover, in the subset without distinguished elements there must be at least one non-distinguished element and, by c-connectedness of $\mathcal{D}_q$, it must be connected by a path in the incidence graph of $\mathcal{D}_q$ to at least one of the distinguished elements in the other subset. This means that there is a fact in $\mathcal{D}_q$ that contains an element from $\text{adom}(\mathcal{D}'_q)$ and an element from $\text{adom}(\mathcal{D}''_q)$. However, this implies that $\mathcal{D}_1$ and $\mathcal{D}_2$ are not disjoint, because there will be a fact in $\mathcal{D}_1 \uplus \mathcal{D}_2$ that contains an element from $\text{adom}(\mathcal{D}_1)$ and an element from $\text{adom}(\mathcal{D}_2)$. Contradiction. We can conclude that either $h : (\mathcal{D}_q, \overline{x}) \to (\mathcal{D}_1, \overline{a})$ or $h : (\mathcal{D}_q, \overline{x}) \to (\mathcal{D}_2, \overline{a})$. □

We now prove the lower bound results. We first present a proof that the Free Query-by-Example problem for *c-connected* CQs is **coNExpTime**-hard, and then we show that in fact this result extends to *any* conjunctive query. To this end, we also need to define what it means to be the *c-connected part* of a CQ:

Definition 4.4.5 (C-connected part of a CQ)**.**
Given a CQ $q(\bar{t})$, we can obtain the *c-connected part* of $q(\bar{t})$ (which we will denote as $\hat{q}(\bar{t})$) in the following way. Take the incidence graph of $q(\bar{t})$ and remove all facts in it that are not connected to any distinguished element by a path. Finally, considering this modified incidence graph of $q(\bar{t})$, we define $\hat{q}(\bar{t})$ to be the same query as $q(\bar{t})$ but without the facts that we removed from its original incidence graph.

Lemma 4.4.3. *Given a database instance $\mathcal{D}$ as well as relations S^+ and S^- over $\mathcal{D}$, $(\mathcal{D}, S^+, S^-)$ has a fitting CQ if and only if $(\mathcal{D}, S^+, S^-)$ has a fitting c-connected CQ.*

Proof.
The right-to-left direction is trivial. For the forward direction, fix a database instance $\mathcal{D}$ as well as relations S^+ and S^- over $\mathcal{D}$. We want to show that if $(\mathcal{D}, S^+, S^-)$ has a fitting query $q(\bar{t})$ then $\hat{q}(\bar{t})$ (i.e. the *c-connected part* of q) fits $(\mathcal{D}, S^+, S^-)$. Assume that $(\mathcal{D}, S^+, S^-)$ has a fitting query $q(\bar{t})$. This means that $S^+ \subseteq q(\mathcal{D})$ and that $S^- \cap q(\mathcal{D}) = \emptyset$.

For the positive examples, from $S^+ \subseteq q(\mathcal{D})$ it follows that $(\mathcal{D}_q, \bar{t}) \rightarrow (\mathcal{D}, \bar{a})$ for all $\bar{a} \in S^+$, and it is trivial to see that we can exploit the same homomorphism to preserve all facts of $\hat{q}(\bar{t})$, hence allowing us to claim that $(\mathcal{D}_{\hat{q}}, \bar{t}) \rightarrow (\mathcal{D}, \bar{a})$ for all $\bar{a} \in S^+$.

For the negative examples, from $S^- \cap q(\mathcal{D}) = \emptyset$ it follows that $(\mathcal{D}_q, \bar{t}) \not\rightarrow (\mathcal{D}, \bar{b})$ for all $\bar{b} \in S^-$. Assume for a contradiction that there is $\bar{b} \in S^-$ and a homomorphism h such that $h : (\mathcal{D}_{\hat{q}}, \bar{t}) \rightarrow (\mathcal{D}, \bar{b})$. Note that when we go from a CQ q to its c-connected part $\hat{q}$, if q isn't already c-connected, we also get a "remaining" CQ $\hat{q}'$, which is what remains of q once we have removed the c-connected part of it. Because we know that $(\mathcal{D}_q, \overline{x}) \rightarrow (\mathcal{D}, \bar{a})$ for all $\bar{a} \in S^+$, it is easy to see that there is a homomorphism h' such that $h' : \mathcal{D}_{\hat{q}'} \rightarrow \mathcal{D}$. Then, because we also assumed that $h : (\mathcal{D}_{\hat{q}}, \overline{x}) \rightarrow (\mathcal{D}, \bar{b})$, we can easily get a new homomorphism h'' such that $h'' : (\mathcal{D}_q, \overline{x}) \rightarrow (\mathcal{D}, \bar{b})$ in the following way. Given $x \in \text{adom}(\mathcal{D}_q)$:

$$h''(x) = h(x) \text{ if } x \in \text{adom}(\mathcal{D}_{\hat{q}}).$$
$$h''(x) = h'(x) \text{ if } x \in \text{adom}(\mathcal{D}_{\hat{q}'}).$$

It is clear that h'' is a correct homomorphism as $\text{adom}(\mathcal{D}_q) = \text{adom}(\mathcal{D}_{\hat{q}}) \cup \text{adom}(\mathcal{D}_{\hat{q}'})$ and $\text{adom}(\mathcal{D}_{\hat{q}}) \cap \text{adom}(\mathcal{D}_{\hat{q}'}) = \emptyset$. Moreover, by assumption, h and h' are correct homomorphisms. Lastly, note that $\mathcal{D}_q$ cannot contain a fact like, say, $R(a, b)$ where $a \in \text{adom}(\mathcal{D}_{\hat{q}})$ and $b \in \text{adom}(\mathcal{D}_{\hat{q}'})$, because if $a \in \text{adom}(\mathcal{D}_{\hat{q}})$ then b must also be in the c-connected part. But $h'' : (\mathcal{D}_q, \overline{x}) \rightarrow (\mathcal{D}, \bar{b})$ is a contradiction. We can conclude that $S^- \cap \hat{q}(\mathcal{D}) = \emptyset$.

As a result, we can claim that $\hat{q}$ is a fitting c-connected CQ for $(\mathcal{D}, S^+, S^-)$. □

Theorem 4.4.4. *The Free Query-by-Example problem for CQs is* **coNExpTime***-hard.*

Proof.

We give a reduction from the Query-by-Example problem for CQs. Take a database instance $\mathcal{D}$ and non-empty relations of positive and negative examples S^+ and S^- (respectively) over it. In the following we present our reduction. We create a database instance $\mathcal{D}_1$ in the following way: assuming that $\mathcal{D} = \{R_1(\bar{a}_1), \ldots, R_k(\bar{a}_k)\}$, $\mathcal{D}_1$ is $\{R'_1(\bar{c}_1), \ldots, R'_k(\bar{c}_k)\}$, where each $R'_i(\bar{c}_i)$ is obtained in the following way:

1. $R'_i = R_i$.
2. $\bar{c}_i$ is obtained from $\bar{a}_i$ be replacing each $a_i \in \bar{a}$ with a fresh constant c_i.

Then we create another database instance $\mathcal{D}_2$ in an entirely similar fashion. Note that, by the way our reduction works, $\mathcal{D}_1$ and $\mathcal{D}_2$ are disjoint. Finally, sets of positive and negative examples from $\mathcal{D}_1$ and $\mathcal{D}_2$ are obtained accordingly. We say that $\mathcal{D}_1 \cup \mathcal{D}_2 = \mathcal{D}'$ and that $S_1^+ \cup S_2^+ = S'^+$ while $S_1^- \cup S_2^- = S'^-$. Now, by Lemma 4.4.3, we can restrict attention to c-connected CQs, and thus we need to show that $(\mathcal{D}, S^+, S^-)$ has a fitting c-connected CQ q without constants if and only if $(\mathcal{D}', S'^+, S'^-)$ has a fitting CQ possibly with constants.

For the forward direction, assume that there is a fitting c-connected CQ $q(\bar{t})$ for $\mathcal{D}$, i.e. given relations S^+ and S^- over $\mathcal{D}$, $S^+ \subseteq q(\mathcal{D})$ and $S^- \cap q(\mathcal{D}) = \emptyset$. Note that, because we assumed that S^+ is non-empty, since $\mathcal{D}$ only contains labelled nulls, by Lemma 4.4.1, q does not contain any constant. By assumption, and by the Chandra-Merlin theorem, we know that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}, \bar{a})$ for all $\bar{a} \in S^+$. It is easy to see that, by construction, and using the same query $q(\bar{t})$, we get:

$$(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{c}) \text{ for all } \bar{c} \in S_1^+$$
$$(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{c}) \text{ for all } \bar{c} \in S_2^+$$

From this, we need to show the following:

$$(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c}) \text{ for all } \bar{c} \in S'^+$$

But this follows immediately: take an arbitrary $\bar{c} \in S_1^+ \cup S_2^-$: it is clear that either $\bar{c} \in S_1^+$ or $\bar{c} \in S_2^+$. Assume that $\bar{c} \in S_1^+$ (the second case is analogous): we know that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{c})$, so it is also true that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c})$, as $\mathcal{D}_1 \subseteq \mathcal{D}'$. Hence, given the positive examples S'^+, it follows that $S'^+ \subseteq q(\mathcal{D}')$.

To show that $S'^- \cap q(\mathcal{D}') = \emptyset$, we first assume that $S^- \cap q(\mathcal{D}) = \emptyset$. Moreover, assume towards a contradiction that $S'^- \cap q(\mathcal{D}') \neq \emptyset$, i.e. there is $\bar{c} \in S'^-$ such that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c})$. Because we assumed that $S^- \cap q(\mathcal{D}) = \emptyset$, it follows that – for all $\bar{b} \in S^-$ – $(\mathcal{D}_q, \bar{x}) \not\to (\mathcal{D}, \bar{b})$. We want to show that there is a homomorphism h from $(\mathcal{D}', \bar{c})$ to $(\mathcal{D}, \bar{b})$, for an arbitrary $\bar{b} \in S^-$. At first, this seems impossible because, while $\bar{c}$ is a tuple of constants, $\bar{b}$ is a tuple of labelled nulls, and we have to map constants to constants. However recall that, by Lemma 4.4.1, q does not contain any constant, and so we replace $\bar{c}$ with a tuple $\bar{c}'$ of labelled nulls, therefore obtaining a new database $\mathcal{D}''$. We are allowed to do this because, since there are no constants in q, we have that $q(\mathcal{D}') = q(\mathcal{D}'')$. It is then easy to design a homomorphism $h : (\mathcal{D}'', \bar{c}') \to (\mathcal{D}, \bar{b})$: $\mathcal{D}''$, by definition, contains two "copies" of each fact in $\mathcal{D}$, so we can map each pair of facts in $\mathcal{D}''$ to the original fact in $\mathcal{D}$. It follows that

$$(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}'', \bar{c}') \to (\mathcal{D}, \bar{b})$$

Contradiction. We can therefore conclude that $S'^{-} \cap q(\mathcal{D}') = \emptyset$, as wanted.

For the backward direction, assume that there is a c-connected fitting CQ $q(\bar{t})$ for $\mathcal{D}'$, i.e. – given relations S'^{+} and S'^{-} over $\mathcal{D}'$ – $S'^{+} \subseteq q(\mathcal{D}')$ and $S'^{-} \cap q(\mathcal{D}') = \emptyset$.

Assume for a contradiction that $S^{+} \not\subseteq q(\mathcal{D})$, i.e. there is a tuple $\bar{a} \in S^{+}$ such that $(\mathcal{D}_q, \bar{x}) \not\to (\mathcal{D}, \bar{a})$. We want to show that our CQ $q(\bar{t})$ does not contain any constant. Because $q(\bar{t})$ is c-connected, and because $\mathcal{D}'$ is the union of two disjoint databases, we can apply Lemma 4.4.2: because, by assumption, there is a homomorphism h such that $h : (\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c})$ for all $\bar{c} \in S'^{+}$, then either $h : (\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_1, \bar{c})$ for all $\bar{c} \in S_1^{+}$ or $h : (\mathcal{D}_q, \bar{t}) \to (\mathcal{D}_2, \bar{c})$ for all $\bar{c} \in S_2^{+}$. Note that both S_1^{+} and S_2^{+} are non-empty and, in both cases, because we have to map constants to constants and because we assumed that $h : (\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c})$ for all $\bar{c} \in S'^{+}$, if a constant was actually present in q we would need to map it to two different constants. Therefore no constant is actually present in q.

Now, recall that we assumed that there is a tuple $\bar{a} \in S^{+}$ such that $(\mathcal{D}_q, \bar{x}) \not\to (\mathcal{D}, \bar{a})$. We also assumed that $(\mathcal{D}_q, \bar{t}) \to (\mathcal{D}', \bar{c})$ for all $\bar{c} \in S'^{+}$. Note that, because we figured that q does not contain constants, we can apply the same strategy we saw before to claim that there is a homomorphism from $(\mathcal{D}', \bar{c})$ to $(\mathcal{D}, \bar{a})$, even if the distinguished tuple in $\mathcal{D}'$ is a tuple of constants while the distinguished tuple in $\mathcal{D}$ is a tuple of nulls. From this, we derive $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{a})$. Contradiction. We can conclude that $S^{+} \subseteq q(\mathcal{D})$.

Regarding the negative examples, by assumption we know that for all $\bar{c} \in S'^{-}$, $(\mathcal{D}_q, \bar{t}) \not\to (\mathcal{D}', \bar{c})$. Assume for a contradiction that there is $\bar{b} \in S^{-}$ such that $(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{b})$. It is easy to see that there is a homomorphism from $(\mathcal{D}, \bar{b})$ to $(\mathcal{D}', \bar{c})$, for an arbitrary $\bar{c} \in S'^{-}$, as $\mathcal{D}$ only contains labelled nulls. Therefore we have that

$$(\mathcal{D}_q, \bar{x}) \to (\mathcal{D}, \bar{b}) \to (\mathcal{D}', \bar{c})$$

which is a contradiction. We can conclude that for all $\bar{b} \in S^{-}$, $(\mathcal{D}_q, \bar{x}) \not\to (\mathcal{D}, \bar{b})$, as wanted. As a result, it is indeed the case that $(\mathcal{D}, S^{+}, S^{-})$ has a fitting c-connected CQ q. □

As a corollary, we get the same lower bound for the other two problems:

Theorem 4.4.5. *The Limited Query-by-Example problem for CQs is* **coNExpTime***-hard.*

Proof.

We reduce from the Query-by-Example problem for CQs. Take an input $(\mathcal{D}_1, S_1^+, S_1^-)$ of the Query-by-Example problem for CQs. We use the same reduction we saw for Theorem 4.4.4 and – in addition to the the resulting tuple $(\mathcal{D}_2, S_2^+, S_2^-)$ – we consider $\emptyset$ as $\mathcal{C}'$. It is clear that $(\mathcal{D}_2, S_2^+, S_2^-, \emptyset)$ is a correct input for the Limited Query-by-Example problem for CQs. We have to show that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting query if and only if $(\mathcal{D}_2, S_2^+, S_2^-, \emptyset)$ does.

For the forward direction, we assume that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting query q. By inspection of Theorem 4.4.4 we realize that q does not contain constants and, moreover, it is a fitting CQ for $(\mathcal{D}_2, S_2^+, S_2^-, \emptyset)$, as it is a fitting CQ already for $(\mathcal{D}_2, S_2^+, S_2^-)$.

For the backward direction, we assume that $(\mathcal{D}_2, S_2^+, S_2^-, \emptyset)$ has a fitting query q. By inspection of Theorem 4.4.4 we realize that, again, q does not contain constants and, in addition, it is a fitting CQ for $(\mathcal{D}_1, S_1^+, S_1^-)$. □

Theorem 4.4.6. *The Bounded Query-by-Example problems for CQs is* **coNExpTime***-hard.*

Proof.

We reduce from the Query-by-Example problem for CQs. Take an input $(\mathcal{D}_1, S_1^+, S_1^-)$ of the Query-by-Example problem for CQs. We use the same reduction we saw for Theorem 4.4.4 and – in addition to the the resulting tuple $(\mathcal{D}_2, S_2^+, S_2^-)$ – we consider $k = 0$ as our input k. It is clear that $(\mathcal{D}_2, S_2^+, S_2^-, 0)$ is a correct input for the Bounded Query-by-Example problem for CQs. We have to show that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting query if and only if $(\mathcal{D}_2, S_2^+, S_2^-, 0)$ does.

For the forward direction, we assume that $(\mathcal{D}_1, S_1^+, S_1^-)$ has a fitting query q. By inspection of Theorem 4.4.4 we realize that q does not contain constants and, moreover, it is a fitting CQ for $(\mathcal{D}_2, S_2^+, S_2^-, 0)$, as it is a fitting CQ already for $(\mathcal{D}_2, S_2^+, S_2^-)$.

For the backward direction, we assume that $(\mathcal{D}_2, S_2^+, S_2^-, 0)$ has a fitting query q. We have that q contains no constants and, by Theorem 4.4.4 it follows that q is a fitting CQ for $(\mathcal{D}_1, S_1^+, S_1^-)$. □

4.4.2 Lower bounds for UCQs

We finally consider the lower bounds for the Limited Query-by-Example problem for UCQs and for the Bounded Query-by-Example problem for UCQs. We have that:

Theorem 4.4.7. *The Limited Query-by-Example problem for UCQs is* **coNP***-hard.*

Proof.

We reduce from the Query-by-Example problem for UCQs which, by Theorem 3.2.6 in Chapter 3, we know to be **coNP**-hard. Take an input $(\mathcal{D}, S^+, S^-)$ for the Query-by-Example problem for UCQs. We show how to obtain an input tuple $(\mathcal{D}', S'^+, S'^-, \mathcal{C}')$ for the Limited problem. Take an arbitrary fact $R(v_1, \ldots, v_k)$. We then say that $\mathcal{D}'$ is the set of facts $R'(v'_1, \ldots, v'_k)$ obtained in the following way:

1. $R' = R$;
2. We replace each v_i in $v_1, \ldots, v_k$ with a constant v'_i.

We then have that S'^+ and S'^- are obtained accordingly. In addition to this, we say that $\mathcal{C}' = \emptyset$. We now have to show that $(\mathcal{D}, S^+, S^-)$ has a fitting UCQ if and only if $(\mathcal{D}', S'^+, S'^-, \emptyset)$ does.

For the forward direction, it is clear that – by construction – a fitting UCQ q for $(\mathcal{D}, S^+, S^-)$ would fit $(\mathcal{D}', S'^+, S'^-, \emptyset)$, as q does not contain any constant. For the backward direction, a similar reasoning applies, as a fitting UCQ q for $(\mathcal{D}', S'^+, S'^-, \emptyset)$ could not contain any constant (as $\mathcal{C}' = \emptyset$), so – by construction – it would be a fitting UCQ for $(\mathcal{D}, S^+, S^-)$ as well. □

And we can finally prove hardness for the Bounded problem as well:

Theorem 4.4.8. *The Bounded Query-by-Example problem for UCQs is* **coNP***-hard.*

Proof.

We reduce from the Query-by-Example problem for UCQs which, by Theorem 3.2.6 in 3, we know to be **coNP**-hard. Take an input $(\mathcal{D}, S^+, S^-)$ for the Query-by-Example problem for UCQs. We show how to obtain an input tuple $(\mathcal{D}', S'^+, S'^-, k)$ for the Bounded problem. Take an arbitrary fact $R(v_1, \ldots, v_k)$. We then say that $\mathcal{D}'$ is the set of facts $R'(v'_1, \ldots, v'_k)$ obtained in the following way:

1. $R' = R$;
2. We replace each v_i in $v_1, \ldots, v_k$ with a constant v'_i.

We then have that S'^{+} and S'^{-} are obtained accordingly. In addition to this, we say that $k = 0$. We now have to show that $(\mathcal{D}, S^{+}, S^{-})$ has a fitting UCQ if and only if $(\mathcal{D}', S'^{+}, S'^{-}, 0)$ does.

For the forward direction, it is clear that – by construction – a fitting UCQ q for $(\mathcal{D}, S^{+}, S^{-})$ would fit $(\mathcal{D}', S'^{+}, S'^{-}, 0)$, as q does not contain any constant. For the backward direction, a similar reasoning applies, as a fitting UCQ q for $(\mathcal{D}', S'^{+}, S'^{-}, 0)$ could not contain any constant (as $k = 0$), so – by construction – it would be a fitting UCQ for $(\mathcal{D}, S^{+}, S^{-})$ as well. □

To conclude, we summarize the new results of this thesis in the following table:

Problem	Classification	Results
Free Query-by-Example for CQs	**coNExpTime**-complete	Theorem 4.3.2 and Theorem 4.4.4
Limited Query-by-Example for CQs	**coNExpTime**-complete	Theorem 4.3.3 and Theorem 4.4.5
Bounded Query-by-Example for CQs	**coNExpTime**-hard	Theorem 4.4.6
Free Query-by-Example for UCQs	in **P**	Theorem 4.3.6
Limited Query-by-Example for UCQs	**coNP**-complete	Theorem 4.3.7 and Theorem 4.4.7
Bounded Query-by-Example for UCQs	**coNP**-hard	Theorem 4.4.8

5 Conclusion

Inspired by how natural it seems to allow constants in the queries when we consider the Query-by-Example problem, we proposed a setting in which constants can be meaningfully taken into account without trivializing the process of finding a fitting query for a given input database instance and positive/negative labelled data examples. On this basis, we obtained complexity bounds for some variations of the Query-by-Example problem, for both conjunctive queries and union of conjunctive queries.

We started our presentation of the Query-by-Example problem in Chapter 3, by introducing the standard setting and by showing complexity results about it already known in the literature; in particular, we presented arguments – coming from [8] and [1] – proving that the QBE problem for CQs is **coNExpTime**-complete while the same problem, but for UCQs, is **coNP**-complete. Then, in Chapter 4, we presented a way of adding constants in the queries and thus we modified the QBE problem in various ways. In this setting, we first partition our universe of values $\mathcal{V}$ into a set of constants $\mathcal{C}$ and a set of labelled nulls $\mathcal{N}$. Then, we obtained three different QBE problems:

1. Free QBE for CQs/UCQs: the input consists of a ground database instance $\mathcal{D}$ and relations S^+/S^- over $\mathcal{D}$, and we ask if there is a fitting CQ/UCQ for the input.

2. Limited QBE for CQs/UCQs: the input consists of a ground database instance $\mathcal{D}$, relations S^+/S^- over $\mathcal{D}$ and a subset $\mathcal{C}' \subseteq \mathcal{C}$, and we ask if there is a fitting CQ/UCQ for the input containing only constants from $\mathcal{C}'$.

3. Bounded QBE for CQs/UCQs: the input consists of a ground database instance $\mathcal{D}$, relations S^+/S^- over $\mathcal{D}$ and a $k \in \mathbb{N}$, and we ask if there is a fitting CQ/UCQ for the input containing at most k constants from $\mathcal{C}$.

In this thesis, we were firstly able to obtain a **coNExpTime** upper bound for the general problem for CQs by following a similar procedure to the one presented in [8]. Then, by reduction to the very same general problem, we obtained a **coNExpTime** upper bound for the free QBE for CQs and, in turn, another **coNExpTime** upper bound for the limited problem for CQs by reducing it to the free version. Turning to UCQs, we proved that the general problem has a **coNP** upper bound but, when we allow the constants to appear freely in the queries, the problem trivializes. We finally obtained a **coNP** upper bound for the limited problem for UCQs by reducing it to the general problem. For hardness, we provided a proof that the free problem for CQs is **coNExpTime**-hard. As a result, we realized that we can use a very similar reduction to show that also the Limited problem and the Bounded problem for CQs are **coNExpTime**-hard. For UCQs, we showed **coNP**-hardness for the Limited problem and the Bounded problem.

These high complexity bounds suggest that the problems we are studying are definitely interesting and non-trivial. Also, one might guess that allowing constants in the queries should make the problem computationally harder, while actually we learned that the complexity does not increase. At the same time, *it does not go down* (except for the Free Query-by-Example problem for UCQs where, as we have seen, the problem trivializes), and this leads us to new reflections. In particular, it is interesting to think about how we could apply this setting to more concrete tasks. About this, let us consider again the Limited Query-by-Example problem. Since the problem allows the fitting query to contain only constants from a certain subset $\mathcal{C}' \subseteq \mathcal{C}$, this problem could be cast as an *optimization problem* (i.e. given an input $(\mathcal{D}, S^+, S^-)$, produce a query with as few constants as possible). Then, a possible further direction is to decide which values of $\text{adom}(\mathcal{D})$ should be mentioned, modulo the optimization idea just presented. We could say that $\mathcal{C}'$ should contain only the "meaningful" values of the database we are considering, but an open question remains: is there any way/heuristics to classify the meaningful values and the non-meaningful ones? Sometimes we actually do have an intuition about what is "meaningful" for a query to refer to, for example if a database lists the name of workers, their department and their phone number, it is probably more meaningful to allow the presence of "categorical" values (e.g. department) and not values that

change for every person (e.g. the phone number, or the name).

Continuing on this reflection on how "practical" our approach can become, we must of course mention the fact that, before any implementation of this framework, we should find a way of lower the complexity (more on this in the next section on future research). Then, it would be important to provide an evaluation method to asses how reliable and efficient our framework is. A possible idea in this case is, given an input $(\mathcal{D}, S^+, S^-)$, provide a fitting query q, and then take new relations S'^+ and S'^- from $q(\mathcal{D})$ to see what query q' we get given $(\mathcal{D}, S'^+, S'^-)$ and how close it is to the original query q.

Future Research. The developed framework can be further explored. In particular, it would be interesting to prove an upper bound for the bounded problems for CQs and UCQs; a possible idea for proving an upper bound for the problem with CQs is to reduce an input $(\mathcal{D}_1, S_1^+, S_1^-, k)$ of it to exponentially-many instances $(\mathcal{D}_2, S_2^+, S_2^-, \mathcal{C}')$ of the limited problem, where each $\mathcal{C}'$ has cardinality k, i.e. we test all subsets $\mathcal{C}'$ of size k. It is likely to be the case that if we find such a reduction for the problem with CQs, the same reduction could work for the problem with UCQs as well.

Another promising line of research comes again from [8], where the authors identify the main sources of complexity of the QBE problem and propose relaxations of them that reduce the complexity. As an example, consider the QBE problem for CQs. Let us recall the *QBE test* explained in Chapter 3: the two main sources of complexity for the QBE problem for CQs are then the construction of $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a})$ and the homomorphism check $\prod_{\overline{a} \in S^+}(\mathcal{D}, \overline{a}) \nrightarrow (\mathcal{D}, \overline{b})$, for each $\overline{b} \in S^-$. The authors propose relaxations for these two components of the problem and, in the end, they are able to show that the combinations of both relaxations yields tractability for the QBE problem for CQs. It would be interesting to see if the same polynomial-time result holds if we introduce the possibility of using constants in the queries or if constants make the complexity grow. It is actually difficult to foresee the consequences – from a complexity-theoretical standpoint – of allowing constants in the queries when we consider these problems.

Bibliography

[1] ten Cate, Balder, and Víctor Dalmau. "The product homomorphism problem and applications." In *18th International Conference on Database Theory (ICDT 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[2] ten Cate, Balder, and Victor Dalmau. "Conjunctive queries: Unique characterizations and exact learnability." In 24th International Conference on Database Theory (ICDT 2021). Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[3] Codd, Edgar F. "A relational model of data for large shared data banks." In *Software pioneers*, pp. 263-294. Springer, Berlin, Heidelberg, 2002.

[4] Codd, Edgar F. *Relational completeness of data base sublanguages.* IBM Corporation, 1972.

[5] Chandra, Ashok K., and Philip M. Merlin. "Optimal implementation of conjunctive queries in relational data bases." In Proceedings of the ninth annual ACM symposium on Theory of computing, pp. 77-90. 1977.

[6] Libkin, Leonid. "Incomplete information and certain answers in general data models." In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 59-70. 2011.

[7] Barceló, Pablo, Alexander Baumgartner, Victor Dalmau, and Benny Kimelfeld. "Regularizing conjunctive features for classification." *Journal of Computer and System Sciences* 119 (2021): 97-124.

[8] Barceló, Pablo, and Miguel Romero. "The complexity of reverse engineering problems for conjunctive queries." *arXiv preprint arXiv:1606.01206* (2016).

[9] Li, Hao, Chee-Yong Chan, and David Maier. "Query from examples: An iterative, data-driven approach to query construction." *Proceedings of the VLDB Endowment* 8, no. 13 (2015): 2158-2169.

[10] Tran, Quoc Trung, Chee-Yong Chan, and Srinivasan Parthasarathy. "Query reverse engineering." *The VLDB Journal* 23, no. 5 (2014): 721-746.

[11] Arora, Sanjeev, and Boaz Barak. *Computational complexity: a modern approach.* Cambridge University Press, 2009.

[12] Antonopoulos, Timos, Frank Neven, and Frédéric Servais. "Definability problems for graph query languages." In *Proceedings of the 16th International Conference on Database Theory*, pp. 141-152. 2013.

[13] Hernandez, Michael J., and John L. Viescas. *SQL queries for mere mortals: a hands-on guide to data manipulation in SQL.* Addison-Wesley Longman Publishing Co., Inc., 2000.

[14] Cygan, Marek, Fedor V. Fomin, Alexander Golovnev, Alexander S. Kulikov, Ivan Mihajlin, Jakub Pachocki, and Arkadiusz Socała. "Tight bounds for graph homomorphism and subgraph isomorphism." In *Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms*, pp. 1643-1649. Society for Industrial and Applied Mathematics, 2016.

[15] Kris Lachance, "Why You Should Learn SQL, Regardless of What Job You Have Now.", Linkedin, accessed July 28, 2022. Link.

[16] Khoussainova, Nodira, YongChul Kwon, Wei-Ting Liao, Magdalena Balazinska, Wolfgang Gatterbauer, and Dan Suciu. "Session-based browsing for more effective query reuse." In *International Conference on Scientific and Statistical Database Management*, pp. 583-585. Springer, Berlin, Heidelberg, 2011.

[17] ten Cate, Balder. "The homomorphism lattice of finite structures, unique characterization, and exact learnability". $A|C$ seminar, Amsterdam, the Netherlands, Oct 6, 2021. Link.

[18] Willard, Ross. "Testing expressibility is hard." In *International Conference on Principles and Practice of Constraint Programming*, pp. 9-23. Springer, Berlin, Heidelberg, 2010.

[19] Zloof, Moshé M. "Query by example." In *Proceedings of the May 19-22, 1975, national computer conference and exposition*, pp. 431-438. 1975.