FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Enumerating the Answers to a Query: Beyond Conjunctive Queries

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Markus Kröll
Registration Number 00730146

to the Faculty of Informatics

at the TU Wien

Advisor: Reinhard Pichler
Second advisor: Stefan Szeider

The dissertation has been reviewed by:

Nadia Creignou   Arnaud Durand

Vienna, 9th May, 2019

Markus Kröll

Declaration of Authorship

Markus Kröll
Neubaugasse 10, 1070 Wien

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 9th May, 2019

Markus Kröll

Acknowledgements

First of all, I would like to thank my advisor Reinhard Pichler, both for offering me the position as his PhD student in the first place, as well as for his guidance and inspiration during my time at the TU Wien.

I owe my deepest gratitude to my amazing parents Juliana and Josef Kröll. Their continuous support throughout the years made all of this possible.

I would also like to thank the following friends and colleagues, listed in alphabetical order: Nofar Carmeli, Benny Kimelfeld, Julia Pöllabauer, Sebastian Skritek, and Jeremias Yehdegho.

Abstract

Conjunctive Queries (CQs) are one of the most fundamental classes of database queries. As such, the enumeration problem for CQs - that is, given a query and a database, to output all answers of the query over the database in an efficient way- is well understood. However, it is not clear how or even if results on the enumeration complexity do extend to query languages that are natural generalizations of CQs.
In this thesis, we aim to evaluate the enumeration complexity of CQs that are extended in three different directions: CQs in the presence of Functional Dependencies (FDs), Unions of Conjunctive Queries (UCQs), and Well-designed pattern trees (wdPTs), which extend CQs by allowing partial matching. For CQs in the presence of FDs as well as for UCQs, our focus is on the ability to list output tuples with a constant delay in between, following a linear-time preprocessing. For wdPTs, we show that even some of the simplest queries can not be enumerated within such bounds. Therefore, we consider the task of enumerating the answers to wdPTs with a polynomial delay between answers.
In a seminal paper by Bagan et al., a dichotomy is shown which classifies the acyclic self-join-free CQs into those that admit enumeration with a constant delay. The class of such queries coincides with the class of free-connex CQs. However, this classification no longer holds in the common case where the database exhibits dependencies among attributes: There exist queries that are classified as hard, but their answers can in fact be enumerated with a constant delay, if dependencies are accounted for. We establish a generalization of the dichotomy to accommodate FDs. In addition, we generalize a hardness result for cyclic CQs to accommodate unary FDs, leading to a dichotomy for enumeration with linear delay.
Next, we extend the notion of free-connexity from CQs to UCQs, thus showing that some unions containing intractable CQs are, in fact, tractable. Interestingly, some unions consisting of only intractable CQs are tractable too. The question of finding a full characterization of the tractability of UCQs remains open. Nevertheless, we prove that for several classes of queries, free-connexity fully captures the tractable UCQs.
For wdPTs, several computational problems have been studied in recent years, such as the evaluation problem under structural restrictions. We show that the parameterized complexity of the evaluation problems is in fact strongly connected to enumeration. Thus extending the existing structural restrictions, we identify several tractable and intractable cases of this problem both from a classical complexity point of view and from a parameterized complexity point of view.

Kurzfassung

Konjunktive Abfragen (CQs) zählen zu den fundamentalsten Klassen von Datenbankabfragen. Das Enumerationsproblem für CQs, welches als Eingabe eine Datenbank und eine Abfrage erhählt und alle Antworten der Abfrage zur Datenbank ausgibt, wurde bereits weitgehend untersucht. Für natürliche Generalisierungen von CQs hingegen ist jedoch nicht bekannt, ob bzw. wie sich Ergebnisse über die Komplexität des Enumerationsproblems erweitern lassen.

Das Ziel dieser Arbeit ist es, das Enumerationsproblem für drei solcher natürlichen Erweiterungen zu untersuchen. Diese Erweiterungen sind: Die Vereinigung von konjunktiven Abfragen (UCQs), konjunktive Abfragen mit funktionalen Abhängigkeiten (FDs), und Well-designed pattern trees (wdPTs). Für CQs mit FDs als auch für UCQs liegt unser Fokus auf dem Enumerieren von Antworten mit einer linearen Vorbereitungszeit, gefolgt von einer Enumerierungsphase mit konstanter Verzögerung zwischen der Ausgabe von Antworten. Wir zeigen, dass diese Schranke selbst für simple wdPTs nicht erreichbar ist, weswegen für diese Abfragesprache eine polynomielle Verzögerung zwischen der Ausgabe von Antworten unser Ziel ist. Eine Dichotomie, die klassifiziert, welche azyklischen CQs ohne self-joins effizient enumeriert werden können, wurde in einer grundlegenden Arbeit von Bagan et al. gezeigt. Diese Klasse von CQs fällt mit der Klasse sogenannter fre–connex CQs zusammen. Für Datenbanken, die FDs aufweisen, ist diese Dichotomie aber bereits nicht mehr gültig: Es gibt Abfragen, die als ineffizient klassifiziert werden, deren Antworten aber mit einer konstanten Verzögerung ausgegeben werden können, sofern man die gegebenen FDs berücksichtigt. Wir generalisieren die Dichotomie von Bagan et al. für CQs mit FDs. Des Weiteren generalisieren wir eine untere Schranke für die Komplexität des Enumerierungsproblems von zyklischen CQs mit unären FDs.

Wir erweitern das Konzept von free-connex CQs auf UCQs und zeigen dadurch, dass UCQs, die ineffiziente Anfragen enthalten, effizient sein können. Die vollständige Klassifizierung von UCQs bleibt offen. Dennoch beweisen wir für einige Klassen von Anfragen, dass free-connex UCQs exakt die Menge von effizienten UCQs ausmacht.

Für wdPTs wurden in den letzten Jahren verschiedenste Probleme behandelt, wie beispielsweise das Evaluierungsproblem unter strukturellen Einschränkungen. Wir zeigen einen direkten Zusammenhang zwischen der parametrisierten Komplexität des Evaluierungsproblems und der Komplexität des Enumerierungsproblems. Durch die Einführung weiterer struktureller Einschränkungen identifizieren wir schliesslich wdPTs, die immer effizient

bzw. nicht effiziernt enumeriert werden können, sowohl im Kontext von parametrisierter Komplexität als auch im Kontext von klassischer Komeplexität.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Conjunctive Queries (CQs) are one of the most fundamental and basic query languages in database managements systems. Not only is the expressive power of CQs equivalent to that of Select–Project–Join queries, but they are also the building block of many other query languages. Due to the importance of this query language, many computational problems regarding CQs have been studied intensively, such as the Boolean evaluation, the containment problem or the counting problem [CM77, CR00, GLS01, PS13, DM15].

An integral computational problem for database theory is the *enumeration problem*, which amounts to producing all answers to a query without duplicates. Starting with a seminal work by Durand and Grandjean [DG07], there has been a renewed interest in examining the complexity of the enumeration problem, focusing on a fine-grained analysis [BDG07, KS13a, SSV18, NS18, FRU^{+}18]. When enumerating the solutions to a query over a database, the number of results may be larger than the size of the database itself. Enumeration complexity offers specific measures for the hardness of such problems. In terms of data complexity, the best we can hope for is to output all answers with a *constant delay* between consecutive answers. In the case of query enumeration, this can be done after a *linear preprocessing* phase required to read the database and decide the existence of a first answer. The enumeration class achieving these time bounds is denoted by $\mathsf{DelayC}_{\mathsf{lin}}$.

A result by Bagan et al. [BDG07] includes not only a constant delay enumeration algorithm, but a full dichotomy on the enumeration complexity of acyclic CQs. Specifically, the enumeration complexity of a CQ Q is determined by its structure. An acyclic query is called *free-connex* if the query remains acyclic when treating the head of the query as an additional atom. Enumerating the answers to Q is tractable if and only if $\mathcal{H}(Q)$ is acyclic free-connex.

Example 1.1. Consider a relational database storing data about movies using the relational schemas $\mathsf{Cast}(\mathit{Actor}, \mathit{Movie})$ and $\mathsf{Release}(\mathit{Movie}, \mathit{ProductionCompany})$. The following CQ asks for a list of actors and the production companies they work with:

$$Q(x, y) \leftarrow \mathsf{Cast}(x, z), \mathsf{Release}(z, y). \tag{1.1}$$

Since this query is not acyclic free-connex, we know that the enumeration complexity is not within $\mathsf{DelayC_{lin}}$. □

The above mentioned dichotomy only holds when applied to databases with no additional assumptions, but oftentimes this is not the case. In practice, there is usually a connection between different attributes, and *Functional Dependencies* (FDs) and *Cardinality Dependencies* (CDs) are widely used to model situations where some attributes in a relation imply others. As the following example shows, these constraints also have an immediate effect on the complexity of enumerating answers for queries over such a schema.

Example 1.2. Again consider the CQ given in equation (1.1). If we take into account that a movie has only one production company, we exhibit the FD $\mathsf{Release} : \mathit{Movie} \rightarrow \mathit{ProductionCompany}$, and the enumeration problem becomes easy: We only need to iterate over all tuples of Cast and replace the Movie value with the single $\mathit{ProductionCompany}$ value that the relation $\mathsf{Release}$ assigns to it. This can be done in linear time by first sorting (in linear time [Gra96]) both relations according to Movie. □

Example 1.2 shows that the dichotomy by Bagan et al. [BDG07] no longer holds in the presence of FDs. In fact, we believe that dependencies between attributes are so common in real life that ignoring them in such dichotomies can mean that we miss a significant portion of the tractable cases. Therefore, to get a realistic picture of the enumeration complexity of CQs, we have to take dependencies into account.

A natural extension of CQs is given by *Unions of CQs* (UCQs), as they describe the union of the answers to several CQs. UCQs form an important class of queries as well, as this class captures the positive fragment of relational algebra. Previous work, which implied results on the enumeration complexity of UCQs, imposes strong restrictions on the underlying database [SV17]. However, a dichotomy similar to that of Bagan et al. classifying a UCQ based on its structure alone, is still missing. Lifting Example 1.1 to the setting of a UCQ strongly suggests that extending the original dichotomy on CQs is far from straightforward:

Example 1.3. We enhance the relational database from the previous examples by the binary relational schema $\mathsf{Television}(\mathit{Actor}, \mathit{TVShow})$. Let $Q = Q_1 \cup Q_2$ be a UCQ with

$$\begin{aligned} &Q_1(x, y, w) \leftarrow \mathsf{Cast}(x, z), \mathsf{Release}(z, y), \mathsf{Television}(x, w) \text{ and} \\ &Q_2(x, y, w) \leftarrow \mathsf{Cast}(x, y), \mathsf{Release}(y, w). \end{aligned} \tag{1.2}$$

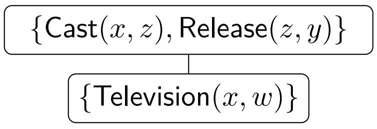
This query lists triples that consist of an actor, an associated production company and either a movie or a TV show that the actor appeared in. According to the dichotomy of Bagan et al. [BDG07], the enumeration problem for Q_2 is in $\mathsf{DelayC_{lin}}$, while the enumeration problem for Q_1 is not. Intuitively, one might be tempted to expect a union of enumeration problems to be harder than a single problem within the union, making such a UCQ intractable as well. Yet, it turns out that Q is in fact in $\mathsf{DelayC_{lin}}$: Since Q_1 and Q_2 are evaluated over the same database, we can use $Q_2(I)$ to find $Q_1(I)$. We can compute $Q_2(I)$ efficiently, and try to extend every such solution to solutions of Q_1 with a constant delay. For every new combination a, c of an output $(a, b, c) \in Q_2(I)$, we find all d values with $(b, d) \in \mathsf{Television}^I$ and then output the solution $(a, c, d) \in Q_1(I)$. $\square$

As this example illustrates, to compute the answers to a UCQ in an efficient way, it is not enough to view it as a union of isolated instances of CQ enumeration. In fact, this task requires an understanding of the possible interactions between several queries.

Going back to CQs, consider the query (1.2) and assume that we evaluate it over a database with incomplete information, that is, for some actors in the database the list of the television shows they appeared in might be missing. If we are mostly interested in a list of actors and associated production companies, and see the TV shows as optional information, then query (1.2) is too restrictive, as it only lists triples with TV shows. What we need in this case is *partial matching* as an extension of CQs.

Well-designed pattern trees (wdPTs) have been introduced [LPPS13] as an extension of CQs to allow for such a partial matching – analogously to the OPTIONAL (OPT) operator of the semantic web query language SPARQL. Intuitively, the OPT operator (which corresponds to the left outer join in the relational algebra) allows the user to extend CQs by optional parts, which are retrieved from the data if available, but which do not cause the partial answers to get lost in case the optional information is not available.

Example 1.4. Consider the following wdPT:


$\{\mathsf{Cast}(x, z), \mathsf{Release}(z, y)\}$
$\{\mathsf{Television}(x, w)\}$

This query consists of two CQs, one for each node of the tree. As an output, the wdPT produces triples of the form $t = (\mathit{actor}, \mathit{movie}, \mathit{productioncompany})$ that correspond to answers of the root node. Then, for each such triple t, it extends the solution to $t' = (\mathit{actor}, \mathit{movie}, \mathit{productioncompany}, \mathit{tvshow})$, if a pair $(\mathit{actor}, \mathit{tvshow})$ exists in the $\mathsf{Television}$ relation. If such an extension exists, t' is output, otherwise t is output. $\square$

To make wdPTs a proper extension of CQs, wdPTs are enhanced with projection. This means that by projecting to the variables x, y and w in Example 1.4, we obtain exactly the output that we were aiming for.

The complexity of enumerating the answers to wdPTs has hardly been considered so far. A notable exception is [LPPS13], where it was shown that enumeration of wdPTs without projection is tractable provided that the CQs contained in the wdPTs are from some tractable fragment. In contrast, for wdPTs with projection, the enumeration problem of wdPTs was shown intractable in [LPPS13] even when allowing only acyclic CQs to appear at each node of a wdPT. To list all answer tuples with a constant or at least a polynomial delay, we need to define classes of wdPTs with restrictions both to the CQs in each node as well as the underlying tree structure.

1.2 Problem Statement

As illustrated by Examples 1.2, 1.3 and 1.4, there are three different natural ways of extending the setting of CQs, for which the complexity of the enumeration problem is still widely unknown: CQs in the presence of dependencies, UCQs and wdPTs. The main goal of this thesis is to initiate a systematic study on this problem, and to ultimately either achieve a dichotomy close to the original dichotomy by Bagan et al., which we often refer to as the *original dichotomy*, or to identify islands of tractability and intractability for the respective query languages.
In this work, the terms *tractability* and *intractability* usually depend on the context: When enumerating the answers to CQs in the presence of FDs or the answers to UCQs, we aim for $\mathsf{DelayC_{lin}}$, and any enumeration problem is tractable if and only if it is within this class. In contrast to that, when we want to enumerate the answers to wdPTs, we mostly aim for enumeration within $\mathsf{OutputP}$ or $\mathsf{OutputFPT}$, which then become the boundaries for tractability.

Effects of FDs on Enumeration Complexity

For CQs in the presence of FDs, the goal is to generalize the dichotomy by Bagan et al. to fully accommodate FDs. To achieve this goal, we introduce an extension of a query Q according to the FDs. The extension is called the FD-extended query, and is denoted by Q^+. This way, instead of classifying every combination of CQ and FDs directly, we want to encode the dependencies within the extended query, and use the classification of Q^+ to gain insight about Q. We aim to show that free-connexity of Q^+ fully classifies the enumeration complexity of Q in the presence of FDs.

Thus, for this encoding to make sense when considering the enumeration problem, we have to show both the positive and the negative side of a dichotomy. For the positive side, this amounts to showing that enumerating the solutions of Q under FDs can be reduced to enumerating the solutions of Q^+ in an efficient way. For the negative side, we need to answer the following question: Is it possible that Q can be enumerated efficiently even if Q^+ is not free-connex?

To show that an enumeration problem is not within a given class, enumeration complexity has few tools to offer. In order to deal with this problem, Bagan et al. reduced the matrix multiplication problem to enumerating the answers to any query that is acyclic but not free-connex. However, this reduction fails when dependencies are imposed on the data, as the constructed database instance does not necessarily satisfy the underlying dependencies. We therefore need to find a new reduction for this case.

The dichotomy by Bagan et al. is not the only one we are interested in extending. Considering cyclic CQ, we also try to extend a similar dichotomy by Brault-Baron [BB13] when dealing with FDs. Moreover, we also consider Cardinality Dependencies (CDs), which are a natural generalization of FDs.

Understanding Unions of CQs

In Example 1.3, we show that a union containing intractable CQs can be enumerated in $\mathsf{DelayC_{lin}}$. We aim to understand when a union containing intractable CQs allows for tractable enumeration. To do so, we want to identify structural properties of UCQs that guarantee tractability. This structural property needs to be a generalization of free-connexity, as CQs are a special case of UCQs.

To make sure that this structural property classifies tractable unions, we want to identify hard queries as well, thus recognizing which unions are always hard, based on some computational hypothesis for a lower bound. As with the case of CQs with FDs, the reduction for the lower bound in the original dichotomy does not hold for UCQs. Similarly to the case of CQs with self-joins, which is usually excluded when aiming for lower bounds, relational symbols that appear multiple times within a query can interfere with the reduction. Indeed, when encoding a hard problem to an intractable CQ within a union, a different CQ in the union evaluates over the same relations, and may also produce answers. A large number of such supplementary answers, with constant delay per answer, accumulates to a long delay until we obtain the answers that correspond to the computationally hard problem. If this delay is larger than the lower bound we assume for the hard problem, we cannot conclude that the UCQ is intractable. We thus need to either identify classes of UCQs for which we can use similar reductions to the ones used for CQs, or introduce alternative reductions.

Enumeration Complexity of wdPTs

As we will see in Section 6.1, enumerating the answers to wdPTs with a constant delay after a linear preprocessing time fails for some of the most basic structures. Therefore, when aiming for a constant delay between outputs, we need to evaluate whether a slightly longer preprocessing phase can be used to achieve tractability.

Since the enumeration complexity of wdPTs has been widely ignored so far, we are also interested in two other notions of efficient enumeration: that of polynomial delay (DelayP) and that of output-polynomial time ($\mathsf{OutputP}$) results in the setting of combined complexity respectively. The former means that the time before outputting the first solution, the time between outputting any two solutions and the time between the last output and termination are all bounded by a polynomial in the size of the input. The latter means that the total time for outputting all solutions is bounded by a polynomial in the combined size of the input plus the output. Above all, we aim at identifying the boundary between tractable and intractable enumeration for the set of both, all solutions and maximal solutions.

Our complexity analysis of the enumeration problem will bring to light an interesting relationship between fixed-parameter tractable enumeration and evaluation. More formally, we will show that FPT-delay of the enumeration problem for some type of wdPTs implies that the evaluation problem for this type of wdPTs is in FPT as well. We thus resume the quest for tractable evaluation from [BPS15] and inspect the intractable cases (classical complexity) from a parameterized complexity point of view. Thus, we aim at delineating the border between fixed-parameter tractability and fixed-parameter intractability to get a better understanding of the nature of the intractability.

1.3 Main Results

The main results of this thesis are grouped into the three different directions by which we extended the setting of CQs. Note that for all of the lower bounds in this thesis, we will use some well-established computational hypotheses, all of which can be found in the preliminaries in Section 2.2.1.

Conjunctive Queries in the Presence of Functional Dependencies

• *Extended Class of Tractable CQs.* We extend the class of queries that are known to be in $\mathsf{DelayC_{lin}}$ by incorporating FDs, and show that enumerating the solutions of Q under FDs can be reduced to enumerating the solutions of Q^+. Therefore, tractability of Q^+ ensures that Q can be efficiently solved as well. By using the positive result in the known dichotomy, Q^+ is tractable w.r.t enumeration if it is free-connex. Moreover, the structural restrictions of acyclicity and free-connex are closed under FD-extensions. Hence, the class of all queries Q such that Q^+ is free-connex is a proper extension of the class of free-connex queries.

• *Dichotomy for Enumeration.* We establish a dichotomy for the enumeration complexity of self-join-free FD-acyclic CQs. That is, we show that the tractability of enumerating the answers of a self-join-free query Q in the presence of FDs is exactly characterized by the structure of Q^+: Given an FD-acyclic query Q, we can enumerate the answers to Q within the class $\mathsf{DelayC_{lin}}$ if and only if Q is FD-free-connex. For the lower bound of this dichotomy, we rewrite the proof of the lower bound for the original dichotomy in order to incorporate FDs. We establish a dichotomy by carefully expanding the reduced instance such that on the one hand, the dependencies hold and on the other hand, the reduction can still be performed within linear time.

• *Cyclic CQs.* The resulting extended dichotomy, as well as the original one, provides insight into the case of acyclic queries. For cyclic CQs, it was shown that even outputting a first solution is not feasible after a linear preprocessing by Brault-Baron [BB13]. Based on the same assumptions used by Brault-Baron, we show that the Boolean evaluation problem for a self-join-free CQ in the presence of unary FDs where Q^+ is cyclic cannot be solved in linear time. As linear time preprocessing is not enough to achieve the first result, enumeration within $\mathsf{DelayC_{lin}}$ is impossible in that case. This covers all types of self-join-free CQs and shows a full dichotomy for the case of unary FDs. Moreover, we show how our results can be easily used to yield additional results, such as a dichotomy for the enumeration of CQs with linear delay.

• *Cardinality Dependencies.* We show the extension of all our results to schemas with CDs and CQs.

Unions of Conjunctive Queries

• *Union extended Queries.* We introduce the concept of *union extended* queries, which captures the interplay of several CQs within a union w.r.t. enumeration. We then use union extensions as a central tool for evaluating the enumeration complexity of UCQs, as the structure of such queries has implications on the tractability of the UCQ. We then show that union extended queries can be used to efficiently enumerate the answers of UCQs that *only* contain queries that are hard according to the original dichotomy. By lifting the concept of free-connex queries from CQs to UCQs via union extended queries, we show that free-connex UCQs are always tractable. This gives us a sufficient global condition for membership in $\mathsf{DelayC_{lin}}$ that goes beyond any classification of individual CQs.

• *Lower Bounds.* We prove that for several classes of queries, free-connexity fully captures the tractable UCQs. A non-free-connex union of two CQs is intractable in the following two cases: both CQs are intractable, or they both represent the same CQ up to a different projection. We then focus on unions of body-isomorphic CQs. For a UCQ of two such queries, we achieve a full dichotomy. For the case of more than two body-isomorphic CQs, we identify a class of queries that is not covered by any lower-bound hypothesis in this work. Excluding such queries, we also achieve an upper as well as a lower bound for a union of more than two body-isomorphic CQs.

• *Hard Cases.* Based on examples, we raise questions to initiate a discussion describing the challenges that will need to be resolved in order to achieve a full characterization of the tractable UCQs. This discussion also focuses on the case of a UCQ containing cyclic CQs.

Well-Designed Pattern Trees

• *Constant Delay Enumeration.* We rule out the possibility of linear time preprocessing and constant delay for a very restricted class of wdPTs. However, if we relax the time restriction of the preprocessing phase and allow polynomial time, then constant delay is achievable for an appropriately restricted class of wdPTs.

• *Combined complexity.* Our complexity analysis of the enumeration problem reveals an interesting effect: It turns out that enumerating the maximal solutions is harder than enumerating all solutions, in contrast to the evaluation problem of wdPTs. A yet more detailed picture of the complexity of the enumeration problem (for all solutions resp. for the maximal solutions) is provided by also studying the parameterized complexity of this problem under various restrictions. In addition, we show that the enumeration problem is strongly connected a parameterized version of the evaluation problem. Thus, results for evaluation directly imply results for enumeration.

• *Evaluation problem.* We revisit the evaluation problem of wdPTs. More precisely, we subject the intractable cases from [BPS15] for the evaluation problem to a parameterized complexity analysis. By establishing fixed-parameter tractability as well as intractability results, we provide a more fine-grained picture of the complexity of this problem, thus also achieving enumeration results.

1.4 Structure of this Thesis

Chapter 2 introduces the notation and established results that form the basis for this thesis.

In Chapter 3, we discuss the currently known results for enumeration complexity of CQs as well as as other query languages related to the topic of this thesis.

The main part of this work starts with Chapter 4, which deals with CQs in the presence of FDs. We first introduce the notion of FD-extended queries and establish the equivalence between a query and its FD-extension. After that, the dichotomy for acyclic CQs is shown, followed by a lower bound for cyclic queries under unary FDs. At the end of this section, we show that all results from the previous sections extend to CDs.

In Chapter 5, we study the enumeration complexity of UCQs. We first generalize the notion of free-connexity to UCQs and show that such queries are in $\mathsf{DelayC}_{\mathsf{lin}}$. We then move on to prove lower bounds for evaluating UCQs within the time bounds of $\mathsf{DelayC}_{\mathsf{lin}}$, beginning with some general observations regarding cases where a single CQ is not harder than a union containing it, and then continue to explore other cases. After that, we address the cases of UCQs containing only intractable CQs, two body-isomorphic CQs, and finally, UCQs containing an arbitrary number of body-isomorphic CQs. We close the chapter with a discussion of classes of UCQs that are not classified by our approach and thus need to be resolved in order to achieve a future dichotomy.

In Chapter 6, we study the enumeration complexity of wdPTs. We first show that a linear preprocessing phase is not enough to achieve constant delay enumeration with linear preprocessing for very restricted wdPTs. However, we give positive results for a slightly longer preprocessing. After that, we revisit the evaluation problem of wdPTs. More precisely, we subject the intractable cases from [BPS15] for the evaluation problem to a parameterized complexity analysis. Finally, we study the combined complexity of enumerating all solutions and of enumerating the maximal solutions, and evaluate the tractability of this problem for several classes of wdPTs.

The last chapter of this thesis provides a summary of our results, a discussion of open problems as well as possible future directions.

This thesis is based on the following work:

- Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. To appear in the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems
- *Submission:* Nofar Carmeli and Markus Kröll. Enumeration Complexity of Conjunctive Queries with Functional Dependencies. Extended Version of [CK18], accepted with minor revision for publication in ACM Transactions on Computer Systems (TOCS).
- Markus Kröll, Reinhard Pichler, and Sebastian Skritek. On the complexity of enumerating the answers to well-designed pattern trees. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016

During my doctoral studies, I also worked on the following publications and submissions, which are beyond the scope of this thesis:

- Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, Markus Kröll. Complexity Bounds for Relational Algebra over Document Spanners. To appear in the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.
- Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and static analysis for well-designed pattern trees with projection. *ACM Trans. Database Syst.*, 43(2):8:1–8:44, 2018
- Markus Kröll, Reinhard Pichler, and Stefan Woltran. On the complexity of enumerating the extensions of abstract argumentation frameworks. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1145–1152. ijcai.org, 2017
- Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory (ICDT 2018)*, volume 98 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik

- Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. On the complexity of hard enumeration problems. In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden*, pages 183–195, 2017
- Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. A complexity Theory for Hard Enumeration Problems. Extended Version of [KPW17], to appear in Discrete Applied Mathematics

CHAPTER 2

Preliminaries

This chapter provides definitions as well as notations about most of the objects and basic concepts which will be used in this thesis.

2.1 Queries and Databases

Relational Data Model and Functional Dependencies

A *schema* $\mathcal{S}$ is a pair $(\mathcal{R}, \Delta)$ where $\mathcal{R}$ is a finite set $\{R_1, \dots, R_n\}$ of *relational symbols* and Δ is a set of *Functional Dependencies* (FDs). When the set Δ is empty, we often omit Δ from the schema. Every relational symbol $R_i \in \mathcal{R}$ has an *arity*, which we denote by $\text{arity}(R_i)$. An FD $\delta \in \Delta$ is a constraint of the form $R_i \colon A \to B$, where $R_i \in \mathcal{R}$ and A, B are non-empty sets of integers with $A, B \subseteq \{1, \dots, \text{arity}(R_i)\}$.

Let *dom* be a finite set of constants, also called the *domain*. A *database instance* (which we often simply call *database* or *instance*) $\mathcal{D}$ over schema $\mathcal{S}$ consists of a finite relation $R_i^{\mathcal{D}} \subseteq dom^{\text{arity}(R_i)}$ for every relational symbol $R_i \in \mathcal{R}$, such that all FDs in Δ are *satisfied*. An FD $\delta = R_i \colon A \to B$ is said to be satisfied if, for all tuples $u, v \in R_i^{\mathcal{D}}$ that are equal on the indices of A, u and v are equal on the indices of B. For a database instance $\mathcal{D}$ defined over *dom*, we say that *dom* is the domain of $\mathcal{D}$ as well as the domain of every relation $R_i^{\mathcal{D}}$.

In this work, we will often assume w.l.o.g. that all FDs are of the form $R_i \colon A \to b$, where $b \in \{1, \dots, \text{arity}(R_i)\}$, as we can replace an FD of the form $R_i \colon A \to B$ where $|B| > 1$ by the set of FDs $\{R_i \colon A \to b \mid b \in B\}$. If $|A| = 1$, we say that the FD $\delta : A \to B$ is a *unary* FD.

Conjunctive Queries and Unions of Conjunctive Queries

Let *var* be a set of variables disjoint from *dom*, which is the set of all *variables.* A *Conjunctive Query* (CQ) over schema $\mathcal{S} = (\mathcal{R}, \Delta)$ is an expression of the form

$$Q(\vec{p}) \leftarrow R_1(\vec{v}_1), \dots, R_m(\vec{v}_m), \tag{2.1}$$

where $R_1, \dots, R_m$ are relational symbols of $\mathcal{R}$, the tuples $\vec{p}, \vec{v}_1, \dots, \vec{v}_m$ consist of variables and domain elements such that $|\vec{v}_j| = \text{arity}(R_j)$ for every $R_j \in \mathcal{R}$, and every variable in $\vec{p}$ appears in at least one of $\vec{v}_1, \dots, \vec{v}_m$. We often denote this query as $Q(\vec{p})$ or even Q. Define the variables of Q as $var(Q) = \bigcup_{i=1}^{m} \vec{v}_i$, and define the *free variables* of Q as $\text{free}(Q) = \vec{p}$. We call $Q(\vec{p})$ the head of Q, and the atomic formulas $R_i(\vec{v}_i)$ are called *atoms.* For a set $\mathcal{A}$ of atoms we use $dom(\mathcal{A})$ to denote the set of constants and variables appearing in $\mathcal{A}$, while $var(\mathcal{A})$ refers to the variables only. We further use $\text{atoms}(Q)$ to denote the set of atoms of Q. A CQ is said to contain *self-joins* if some relation symbol appears in more than one atom.

We next turn to the *evaluation* $Q(\mathcal{D})$ of a CQ Q with free variables $\vec{p}$ over a database $\mathcal{D}$. Let Q be a CQ of the form (2.1). A *homomorphism* from Q into $\mathcal{D}$ is a mapping $\mu : var \to dom$ such that for every atom $R(\vec{v}) \in \text{atoms}(Q)$ there exists some tuple $t \in R^{\mathcal{D}}$ with $\mu(\vec{v}) = t$. We define $Q(\mathcal{D})$ to be the set of all *mappings* $\mu|_{\vec{p}}$ such that μ is a homomorphism from Q into $\mathcal{D}$, where $\mu|_{\vec{p}}$ denotes the restriction (or projection) of μ to the variables $\vec{p}$. A mapping $\mu|_{\vec{p}} \in Q(\mathcal{D})$ is called an *answer to Q over $\mathcal{D}$* or *solution to Q over $\mathcal{D}$*. If Q and $\mathcal{D}$ are clear from the context, we simply call $\mu|_{\vec{p}}$ an *answer* or *solution.*

Example 2.1. Consider the query $Q(x, y) \leftarrow \mathsf{Cast}(x, z), \mathsf{Release}(z, y)$ from Example 1.1 over the schema $\mathcal{S} = \{\{\mathsf{Cast}, \mathsf{Release}\}, \{\delta = \mathsf{Release} : 1 \mapsto 2\}\}$. A database $\mathcal{D}$ over $\mathcal{S}$ which satisfies the functional dependency δ is given by

$$\mathsf{Cast}^{\mathcal{D}} = \{(\text{“HarrisonFord”}, \text{“StarWars”}), (\text{“HarrisonFord”}, \text{“RaidersoftheLostArk”})\},$$
$$\mathsf{Release}^{\mathcal{D}} = \{(\text{“StarWars”}, \text{“LucasfilmLtd.”}), (\text{“AmericanGraffiti”}, \text{“LucasfilmLtd.”})\}.$$

Let *dom* be the domain of $\mathcal{D}$. A homomorphism from Q into D is given by a mapping $\mu : var \to dom$, such that $x \mapsto$ “HarrisonFord”, $z \mapsto$ “StarWars” and $y \mapsto$ “LucasfilmLtd.”. Hence the set of solutions contains $\mu|_{\{x,y\}}$, and since this is also the only solution, we have $Q(\mathcal{D}) = \{\{x \mapsto \text{“HarrisonFord”}, y \mapsto \text{“LucasfilmLtd.”}\}\}$. □

A *Union of Conjunctive Queries (UCQ)* Q is a set of CQs, denoted $Q = \bigcup_{i=1}^{\ell} Q_i$, where $\text{free}(Q_{i_1}) = \text{free}(Q_{i_2})$ for all $1 \leq i_1, i_2 \leq \ell$. Semantically, we have that $Q(\mathcal{D}) = \bigcup_{i=1}^{\ell} Q_i(\mathcal{D})$. As with CQs, the elements in $Q(\mathcal{D})$ are called *answers* or *solutions.* A CQ is a special case of a UCQ, so we generalize the notations from CQs to UCQs by setting $\text{atoms}(Q) = \bigcup_{i=1}^{\ell} \text{atoms}(Q_i)$, $var(Q) = \bigcup_{i=1}^{\ell} var(Q_i)$ and $\text{free}(Q) = \text{free}(Q_1)$.

Given a CQ or UCQ, we often omit the corresponding schema $S = (\mathcal{R}, \Delta)$. In this case, we can always assume that $\mathcal{R}$ consists of the relational symbols appearing in $\text{atoms}(Q)$ and $\Delta = \emptyset$.

We also need to introduce some technical terms. For a tuple u consisting of variables and/or constants, we denote by $u[i]$ the i-th element of the tuple u. Let $R(\vec{v})$ be an atom of a CQ and $t \in R^{\mathcal{D}}$. We say that a tuple $t \in R^{\mathcal{D}}$ assigns a variable x with the value c if for every index i such that $\vec{v}[i] = x$ we have that $t[i] = c$. We say that a tuple $t_a \in R_a^{\mathcal{D}}$ agrees with a tuple $t_b \in R_b^{\mathcal{D}}$ on the value of a variable x if the following holds: For every pair of indices i_a, i_b and every sequence of variables v_a, v_b with $R_a(v_a), R_b(v_b) \in \text{atoms}(Q)$ such that $\vec{v_a}[i_a] = \vec{v_b}[i_b] = x$, we have that $t_a[i_a] = t_b[i_b]$.

Given a query Q over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, we often identify an FD $\delta \in \Delta$ as a mapping between variables. That is, if δ has the form $R_i : A \to b$ for $A = \{a_1, \ldots, a_{|A|}\}$ and $R_i(\vec{v_i}) \in \text{atoms}(Q)$, we sometimes denote it by $\delta = R_i \colon \{\vec{v_i}[a_1], \ldots, \vec{v_i}[a_{|A|}]\} \to \vec{v_i}[b]$. To distinguish between these two representations, we usually denote subsets of integers by $A, B, C, \ldots$, integers by $a, b, c, \ldots$, and variables by letters from the end of the alphabet.

Example 2.2. Given the CQ Q from Example 2.1 over schema $\mathcal{S}$, we can also denote the FD $\delta = \mathit{Release} : 1 \mapsto 2$ as $\delta = \mathit{Release} : z \mapsto y$. □

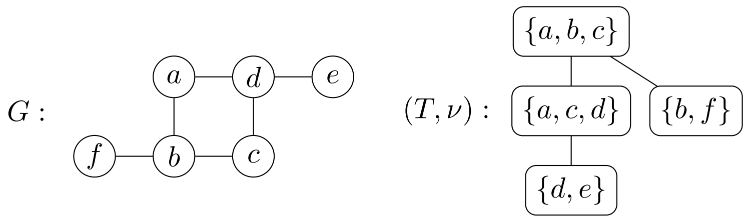
Graphs and Hypergraphs

We consider undirected, simple graphs $G = (V, E)$. For a graph G, we may write $V(G)$ and $E(G)$ to denote the set of nodes and edges, respectively. As usual, a *tree* is an acyclic graph, and a subtree is a connected subgraph of a tree. A *tree decomposition* of a graph $G = (V, E)$ is a pair (T, ν), where T is a tree and $\nu : V(T) \to 2^V$, that satisfies the following:

1. For each $u \in V$ the set $\{t \in V(T) \mid u \in \nu(t)\}$ is a connected subset of $V(T)$, and
2. each edge of E is contained in one of the sets $\nu(t)$, for $t \in V(T)$.

The *width* of a tree decomposition (T, ν) is given by $(\max\{|\nu(t)| \mid t \in V(T)\}) - 1$. The *treewidth* of G is the minimum width of its tree decompositions. Intuitively, the treewidth of G measures its *tree-likeness*. Notice that if G is an undirected graph, then G is acyclic if and only if it is of treewidth one.

Example 2.3. Consider the graph G below. This graph has a treewidth of at most 2, which can be seen by a corresponding tree decomposition (T, ν), where the label of every node N in the tree corresponds to the set $\nu(N)$. Moreover, this tree decomposition gives us a minimal treewidth, as G is cyclic and thus has a treewidth of at least 2.



□

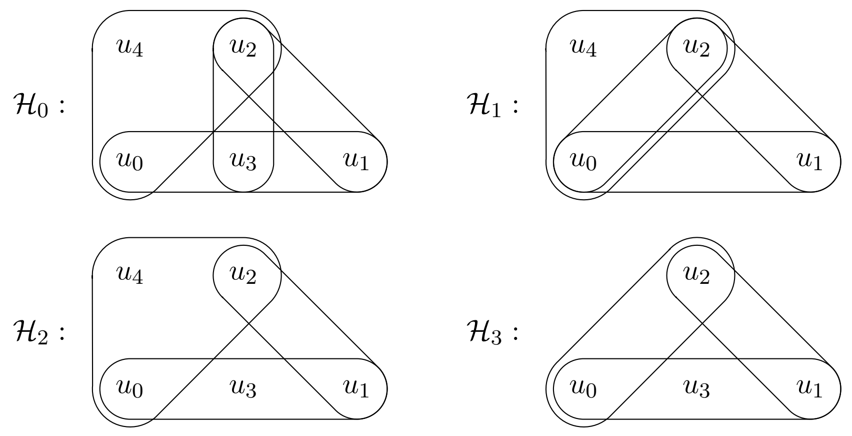


Figure 2.1: The hypergraphs $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ are pseudo-minors of $\mathcal{H}_0$ as given in Example 2.4.

A *hypergraph* $\mathcal{H} = (V, E)$ is a generalization of a graph, consisting of a set V of *vertices*, and a set E of non-empty subsets of V called *hyperedges* (sometimes *edges*). A *join tree* of a hypergraph $\mathcal{H} = (V, E)$ is a tree T where the nodes are the hyperedges of $\mathcal{H}$, and the *running intersection* property holds, which is the same as property (1) for tree decompositions: for all $u \in V$ the set $\{e \in E \mid u \in e\}$ forms a connected subtree in T. A hypergraph $\mathcal{H}$ is said to be *acyclic* if there exists a join tree for $\mathcal{H}$. Two distinct vertices in a hypergraph are said to be *neighbors* if they appear in the same edge. For $n \geq 2$, a tuple $P = (x_1, \ldots, x_n)$ of vertices in $\mathcal{H}$ is a *path* if x_i, x_{i+1} are neighbors for all $1 \leq i \leq n$, and it is *chordless* if these are the only neighbors in P. The *length* of the path P is $n - 1$.

A *clique* of a hypergraph is a set of vertices, which are pairwise neighbors in $\mathcal{H}$. A hypergraph $\mathcal{H}$ is said to be *conformal* if every clique of $\mathcal{H}$ is contained in some edge of $\mathcal{H}$. A *chordless cycle* of $\mathcal{H}$ is a tuple $(x_1, \ldots, x_n)$, $n \geq 3$ such that the set of neighboring pairs of variables of $\{x_1, \ldots, x_n\}$ is exactly $\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq n-1\} \cup \{\{x_n, x_1\}\}$. It is well known (see [BFMY83]) that a hypergraph is acyclic if and only if it is conformal and contains no chordless cycles.

A *pseudo-minor* of a hypergraph $\mathcal{H} = (V, E)$ is a hypergraph obtained from $\mathcal{H}$ by a finite series of the following operations:

1. *vertex removal*: removing a vertex from V and from all edges in E that contain it.
2. *edge removal*: removing an edge e from E provided that some other $e' \in E$ contains it.
3. *edge contraction*: replacing all occurrences of a vertex v (within every edge) with a vertex u, provided that u and v are neighbors.

Example 2.4. Consider the hypergraph $\mathcal{H}_0 = (V, E)$ with $V = \{u_0, u_1, u_2, u_3, u_4\}$ and $E = \{\{u_0, u_2, u_4\}, \{u_1, u_2\}, \{u_0, u_1, u_3\}, \{u_2, u_3\}\}$ as given in Figure 2.1. We obtain the pseudo-minor $\mathcal{H}_1$ by edge contraction, replacing all occurrences of u_3 by u_0. Note that the edge $\{u_0, u_1\} \in E(\mathcal{H}_1)$ is obtained, as a set can not contain duplicates of u_0. The pseudo-minor $\mathcal{H}_2$ is obtained by removing $\{u_0, u_2\}$. Finally, the pseudo-minor $\mathcal{H}_3$ is obtained by removing the vertex u_4. $\square$

If every edge in $\mathcal{H}$ has k many vertices, we call $\mathcal{H}$ k-*u*niform. An *l-hyperclique* in a k-uniform hypergraph $\mathcal{H}$ is a set V' of $l > k$ vertices, such that every subset of V' of size k forms a hyperedge.

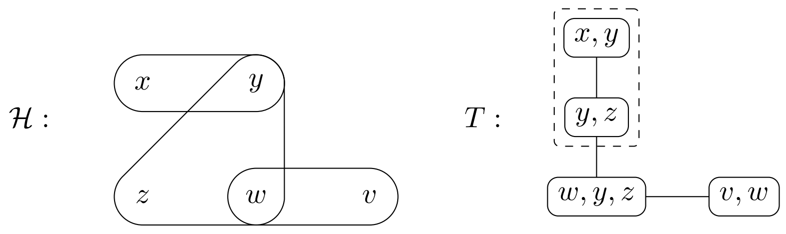
Let $\mathcal{H} = (V, E)$ and $S \subseteq V$. A hypergraph $\mathcal{H}'$ is an *inclusive extension* of $\mathcal{H}$ if every edge of $\mathcal{H}$ appears in $\mathcal{H}'$, and every edge of $\mathcal{H}'$ is a subset of some edge in $\mathcal{H}$. We further say that a tree T is an *ext-S-connex tree* for $\mathcal{H}$ if:

1. T is a join-tree of an inclusive extension of $\mathcal{H}$, and
2. there is a subtree T' of T that contains exactly S [BDG07].

If such a tree T for $\mathcal{H}$ and S exists, then $\mathcal{H}$ is called *ext-S-connex* acyclic. When it is clear from the context that $\mathcal{H}$ is acyclic, we sometimes call $\mathcal{H}$ *ext-S-connex* instead.
We say that $P = (x, z_1, \dots, z_m, y)$ is an S-path of $\mathcal{H}$ if

1. P is a chordless path in $\mathcal{H}$,
2. $x, y \in S$ and
3. $z_1, \dots, z_m \notin S$.

Example 2.5. Consider the hypergraph $\mathcal{H}$ given below. T is an ext-$\{x, y, z\}$-connex tree for hypergraph $\mathcal{H}$. Moreover, the path $P = (x, y, w)$ in $\mathcal{H}$ is an S-path for $S = \{x, w\}$.



$\square$

Classes of CQs

To a CQ Q we associate a hypergraph $\mathcal{H}(Q) = (V, E)$ where the vertices V are the variables of Q and every hyperedge E is a set of variables occurring in a single atom of Q, that is $E = \{\{v_1, \dots, v_n\}\} \mid R_i(v_1, \dots, v_n) \in \text{atoms}(Q)\}$. With a slight abuse of notation, we also identify atoms of Q with edges of $\mathcal{H}(Q)$. A CQ Q is said to be *acyclic* if $\mathcal{H}(Q)$ is acyclic, and it is said to be *free-connex* acyclic if both Q and $(V, E \cup \{\text{free}(Q)\})$

are acyclic. When it is clear that Q is acyclic, we sometimes use the term free-connex instead of free-connex acyclic.

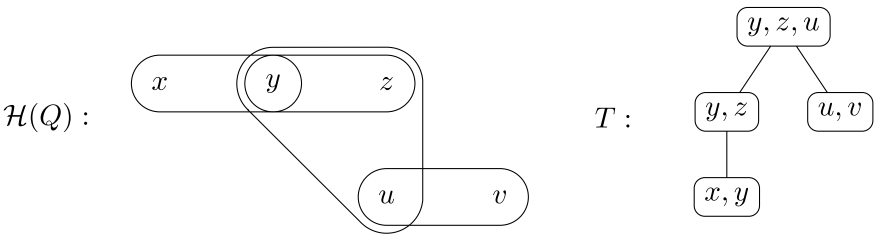
The Gaifman-graph of a CQ Q is a graph $G = (V, E)$ where $V = var(Q)$ and E contains exactly all pairs of variables that jointly occur in some relational atom in Q. The treewidth of a CQ is the treewidth of its Gaifman-graph. We denote by $\mathsf{TW}(k)$ the class of CQs of treewidth at most k, for $k \geq 1$.

A *free-path* for a CQ Q is a sequence of variables $(x, z_1, \ldots, z_k, y)$ with $k \geq 1$, such that:

1. $\{x, y\} \subseteq \text{free}(Q)$
2. $\{z_1, \ldots, z_k\} \subseteq V \setminus \text{free}(Q)$
3. It is a *chordless path* in $\mathcal{H}(Q)$, that is, two succeeding variables appear together in some atom, and no two non-succeeding variables appear together in an atom.

Bagan et al. [BDG07] showed that an acyclic CQ has a free-path if and only if it is not free-connex. It was shown that an acyclic query Q is free-connex if and only if there exists an ext-free(Q)-connex tree for $\mathcal{H}(Q)$ [BB13].

Example 2.6. Consider the CQ $Q(x, y, v) \leftarrow R_1(x, z), R_2(z, y), R_3(y, z, u), R_4(u, v)$. This query is acyclic, since T is a join-tree of $\mathcal{H}(Q)$, see below. Moreover, the query is not free-connex, as there exists a free-path (y, u, v).



$\square$

Well-designed pattern trees

Well-designed pattern trees (wdPTs) are queries that were originally introduced in [LPPS13] as a graphical representation of *well-designed SPARQL* defined in [PAG09] and later extended to arbitrary relational vocabulary [BPS15]. Their formal definition is given below.

Definition 2.7 (wdPTs). A *well-designed pattern tree* (wdPT) p is a tuple $(T, \lambda, \vec{x})$, such that the following holds:

1. T is a rooted tree and λ maps each node $N \in V(T)$ to a set of relational atoms.
2. For every variable y mentioned in T, the set of nodes of T where y occurs is connected.

3. The tuple $\vec{x}$ of distinct variables from T denotes the *free variables* of the wdPT.

We say that $(T, \lambda, \vec{x})$ is *projection-free*, if $\vec{x}$ contains all variables mentioned in T.

Note that unlike SPARQL queries which are defined over so-called RDF triples [PS08], wdPTs are defined over arbitrary relational schemas, abstracting away from the specifics of the semantic web data model RDF. As we will see below, this also means that CQs correspond to the special case of wdPTs consisting of only the root node.

Condition (2) in Definition 2.7 is referred to as the *well-designedness condition* introduced in [PAG09]. In the context of wdPTs, we use upper-case letters to denote nodes of a wdPT, and lower-case letters for vertices of tree decompositions and general graphs. For a wdPT $p = (T, \lambda, \vec{x})$ and a node $N \in V(T)$, we may abbreviate $var(\lambda(N))$ to $var(N)$. Also, for a subtree T' of T we may use $var(T')$ to denote the set $\bigcup_{N \in V(T')} var(N)$. Finally, we may write $var(p)$ instead of $var(T)$.

Let $p = (T, \lambda, \vec{x})$ be a wdPT. We write R to denote the root of T. Given a subtree T' of T rooted in R, we define $Q_{T'}$ to be the CQ $Q_{T'}(\vec{y}) \leftarrow R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)$, where $\{R_1(\vec{v}_1), \ldots, R_m(\vec{v}_m)\} = \bigcup_{N \in T'} \lambda(N)$, and $\vec{y} = var(T')$. Finally, we write $|p|$ to denote the *size* of p in standard relational notation – which corresponds to the size of Q_T.

Example 2.8. Consider the following $\{\mathsf{AND}, \mathsf{OPT}\}$-SPARQL query that is posed over an RDF database that stores information about movies. We use here the algebraic-style notation from [PAG09] rather than the official SPARQL syntax of [PS08].

$$\Big(((x, \texttt{directed\_by}, y) \;\mathsf{AND}\; (x, \texttt{released}, \text{“before\_1980”})) \\ \mathsf{OPT}\; (x, \texttt{oscars\_won}, z)\Big) \;\mathsf{OPT}\; (y, \texttt{first\_movie}, z').$$

This query retrieves all pairs (m, d) such that movie m is directed by d and released before 1980. This is specified by the pattern

$$(x, \texttt{directed\_by}, y) \;\mathsf{AND}\; (x, \texttt{released}, \text{“before\_1980”}).$$

Furthermore, whenever possible, this query also retrieves (one or both of) the following pieces of data: the number of Academy Awards n won by movie m and the first movie m' directed by d. In other words, in addition to (m, d) we also retrieve n and/or m' if the information is available in the database. This is specified by the atoms $(x, \texttt{won\_oscars}, z)$ and $(y, \texttt{first\_movie}, z')$ following the respective OPT-operators.

The wdPT $p = (T, \lambda, \vec{x})$ corresponding to the SPARQL-query above is given in Figure 2.2. Note that wdPTs express the optional operator OPT as an edge, and that we use relational atoms instead of RDF triples.

Formally, the query p is defined as:

- T consists of a root R with child nodes N_1 and N_2,

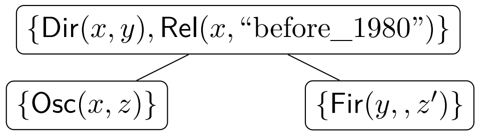

$\{\mathsf{Dir}(x,y), \mathsf{Rel}(x, \text{“before\_1980”})\}$
$\{\mathsf{Osc}(x,z)\}$
$\{\mathsf{Fir}(y,,z')\}$

Figure 2.2: The wdPT of Example 2.8.

- $\lambda(R) = \{\mathsf{Dir}(x,y), \mathsf{Rel}(x, \text{“before\_1980”})\}$, $\lambda(N_1) = \{\mathsf{Osc}(x,z)\}$, $\lambda(N_2) = \{\mathsf{Fir}(y,,z')\}$,
- $\vec{x} = \{x,y,z,z'\}$.

Given the subtree T' of T containing nodes R and N_1 as well as the edge between the two nodes, the CQ $Q_{T'}$ is defined as $Q_{T'}(x,y,z) \leftarrow \mathsf{Dir}(x,y), \mathsf{Rel}(x, \text{“before\_1980”}), \mathsf{Osc}(x,z)$.
$\square$

We define the semantics of wdPTs by naturally extending their interpretation under semantic web vocabularies [LPPS13, PS14]. Intuitively, a mapping μ satisfies (T,λ) over a database $\mathcal{D}$ if it is a solution to a CQ $Q_{T'}$ and if no proper extension of μ is a solution to some CQ $Q_{T''}$, where T' and T'' are both subtrees of T. To formally define this, we need to fix some concepts for partial mappings.

Given a partial mapping μ, we denote by $dom(\mu)$ the set of variables on which μ is defined. Given two partial mappings μ_1, μ_2, we say that μ_1 is *subsumed* by μ_2, written as $\mu_1 \subseteq \mu_2$, if $dom(\mu_1) \subseteq dom(\mu_2)$ and $\mu_1(x) = \mu_2(x)$ for every $x \in dom(\mu_1)$. If $\mu_1 \subseteq \mu_2$ but $\mu_2 \subseteq \mu_1$ does not hold, we write $\mu_1 \subset \mu_2$.

The result of evaluating a wdPT $(T,\lambda,\vec{x})$ over $\mathcal{D}$ contains the projection of all mappings satisfying (T,λ) to $\vec{x}$. We formalize this next.

Definition 2.9 (Semantics of wdPTs)**.** Let $p = (T,\lambda,\vec{x})$ be a wdPT and $\mathcal{D}$ a database, and let R be the root of T.

- A *homomorphism* from p to $\mathcal{D}$ is a partial mapping $\mu : var \to dom$ for which there exists a subtree T' of T rooted in R such that $\mu|_{var(T')} \in Q_{T'}(\mathcal{D})$.
- A homomorphism μ from p to $\mathcal{D}$ is *maximal* if there is no homomorphism μ' from p to $\mathcal{D}$ such that $\mu \subset \mu'$.

The *evaluation* of wdPT $p = (T,\lambda,\vec{x})$ over $\mathcal{D}$, denoted $p(\mathcal{D})$, corresponds to the set of all mappings of the form $\mu|_{\vec{x}}$, such that μ is a maximal homomorphism from p to $\mathcal{D}$.

For a wdPT p and a database $\mathcal{D}$, a mapping μ is called a *partial solution* if there exists some solution $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$. Observe that $p(\mathcal{D})$ may contain mappings μ, μ' such that $\mu \subseteq \mu'$. We thus define the set of *maximal solutions* as $p_m(\mathcal{D}) = \{\mu \in p(\mathcal{D}) \mid \text{for all } \mu' \in p(\mathcal{D})\colon \mu \not\subset \mu'\}$. We revisit Example 2.8 to illustrate these ideas.

Example 2.10. Consider the database $\mathcal{D}$ over schema $\mathcal{S} = (\{\mathsf{Dir}, \mathsf{Rel}, \mathsf{Osc}, \mathsf{Fir}\}, \emptyset)$:

$$\begin{aligned}
\mathsf{Dir}^{\mathcal{D}} &= \{(\text{“American\_Graffiti”}, \text{“George\_Lucas”}), (\text{“Star\_Wars”}, \text{“George\_Lucas”})\}\\
\mathsf{Rel}^{\mathcal{D}} &= \{(\text{“American\_Graffiti”}, \text{“before\_1980”}), (\text{“Star\_Wars”}, \text{“before\_1980”})\}\\
\mathsf{Osc}^{\mathcal{D}} &= \{(\text{“Star\_Wars”}, \text{“6”})\}\\
\mathsf{Fir}^{\mathcal{D}} &= \emptyset
\end{aligned}$$

The evaluation over $\mathcal{D}$ of the query from Example 2.8 consists of partial mappings μ_1 and μ_2 defined on variables x, y, z, z' such that:

- $\mu_1 = \{x \mapsto \text{“American\_Graffiti”}, y \mapsto \text{“}George\_Lucas\text{”}\}$, and
- $\mu_2 = \{x \mapsto \text{“Star\_Wars”}, y \mapsto \text{“George\_Lucas”}, z \mapsto \text{“2”}\}$.

Indeed, let T_1 be the subtree of T consisting of the root node R, and T_2 the subtree consisting of R and N_1. The corresponding CQs are given by

$$Q_{T_1}(x, y) \leftarrow \mathsf{Dir}(x, y), \mathsf{Rel}(x, \text{“before\_1980”}) \text{ and}$$
$$Q_{T_2}(x, y, z) \leftarrow \mathsf{Dir}(x, y), \mathsf{Rel}(x, \text{“before\_1980”}), \mathsf{Osc}(x, z).$$

As $\mu_1|_{\{x,y\}} \in Q_{T_1}(\mathcal{D})$ and $\mu_2|_{\{x,y,z\}} \in Q_{T_2}(\mathcal{D})$, the mappings μ_1 and μ_2 are homomorphisms from p to $\mathcal{D}$. Moreover, they are maximal homomorphisms. These are the only two solutions in $p(\mathcal{D})$, and both are maximal solutions.

For a query with solutions that are not maximal, consider the wdPT $p' = (T', \lambda', \vec{x}')$ given below.

$$\begin{tikzcd}
 & \boxed{\{S(x, y, z_0)\}} \arrow[dl, dash] \arrow[dr, dash] & \\
\boxed{\{P(x, z_1)\}} & & \boxed{\{Q(y, z_2)\}}
\end{tikzcd}$$

Formally, p' is defined as follows:

- T' consists of a root R with child nodes N_1, N_2,
- $\lambda'(R) = \{S(x, y, z_0)\}$, $\lambda'(N_1) = \{P(x, z_1)\}$, $\lambda'(N_2) = \{Q(y, z_2)\}$ and
- $\vec{x}' = \{z_0, z_1, z_2\}$.

First note that since $\vec{x}' \subsetneq var(p')$, we have that p' is not projection-free. We define the following database instance $\mathcal{D}'$ over a schema $\mathcal{S} = (\{S, P, Q\}, \emptyset)$ and domain $dom = \{0, 1, 2, 3, 4\}$:

$$S^{\mathcal{D}'} = \{(1, 1, 0), (1, 2, 0), (2, 1, 0), (2, 2, 0)\}, P^{\mathcal{D}'} = \{(0, 3), (2, 3)\}, Q^{\mathcal{D}'} = \{(4, 4), (2, 4)\}.$$

We first claim that there is some partial mapping $\mu : var(p') \to dom$ such that $\mu|_{\vec{x}'} = \{z_0 \mapsto 0\} \in p'(\mathcal{D}')$. Indeed, let $\mu = \{x \mapsto 1, y \mapsto 1, z_0 \mapsto 0\}$. For the subtree T'' of T' only consisting of R we have the CQ $Q_{T''}(x, y, z_0) \leftarrow S(x, y, z_0)$ and thus $\mu|_{var(Q_{T''})} \in Q_{T''}(\mathcal{D}')$, meaning that μ is a homomorphism from p to $\mathcal{D}$. Moreover, this mapping μ can not be extended to a mapping μ' with $\mu \subset \mu'$ and $\mu'|_{var(Q_{T'''})} \in Q_{T'''}(\mathcal{D}')$ for any subtree T''' of T'. Thus μ is a maximal homomorphism and therefore $\mu|_{\vec{x}'} \in p'(\mathcal{D}')$. This solution is not a maximal solution of $p'(\mathcal{D}')$: Consider the partial mapping $\mu_1 : var(p') \to dom$ with $\mu_1 = \{x \mapsto 2, y \mapsto 1, z_0 \mapsto 0, z_1, \mapsto 3\}$. For the subtree T^4 of T' consisting of R and N_1, we have the CQ $Q_{T^4}(x, y, z_0, z_1) \leftarrow S(x, y, z_0), P(x, z_1)$, and therefore $\mu_1|_{var(Q_{T^4})} \in Q_{T^4}(\mathcal{D}')$. Again, there is no homomorphism μ_1' such that $\mu_1 \subset \mu_1'$, thus μ_1 is a maximal homomorphism and $\mu_1|_{\{z_0,z_1,z_2\}}$ a solution in $p'(\mathcal{D}')$. The complete set of solutions is given by

$$p(\mathcal{D}) = \{\{z_0 \mapsto 0\}, \{z_0 \mapsto 0, z_1 \mapsto 3\}, \{z_0 \mapsto 0, z_2 \mapsto 4\}\{z_0 \mapsto 0, z_1 \mapsto 3, z_2 \mapsto 4\}\}.$$

□

2.2 Computational Complexity

This section gives a brief introduction to the basic notions and concepts in computational complexity theory. In particular, we will make use of a rich theory in classifying the hardness of decision problems, either in classical complexity or parameterized complexity, and introduce the notions of enumeration complexity. For an in-depth look on both classical complexity and parameterized complexity, see [Pap03] respectively [FG10].

Input, Size and Cardinality

In this work, we only consider finite structures. We thus always assume that an input w is a string over a (fixed) finite alphabet, and we denote by $||w||$ the size of w, which is the length of the string. When the input of a computational problem consists of a query Q and a database $\mathcal{D}$, we use two different approaches for measuring the size of the input: *data complexity* and *combined complexity* [Var82]. We use *data complexity* for the majority of our problems, meaning that the input is measured only by the size of the database instance $\mathcal{D}$. When using *combined complexity*, the input to a problem is measured by the combined input sizes of the query Q and the database $\mathcal{D}$. For any finite set S, we denote by $|S|$ the cardinality of S.

Let $\mathcal{D}$ be a database over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$. The cardinality of $\mathcal{D}$ is the number of tuples stored in $\mathcal{D}$, i.e. $|\mathcal{D}| = \sum_{R \in \mathcal{R}} |R^{\mathcal{D}}|$. Flum et al. describe a reasonable encoding of the database as a string over a finite alphabet [FFG02], such that the input size of $\mathcal{D}$ is given as $||\mathcal{D}|| = 1 + |dom| + |\mathcal{R}| + \sum_{R \in \mathcal{R}} \text{arity}(R)|R^{\mathcal{D}}|$.

When we evaluate the combined complexity of a problem, we also consider a query as part of an input. For a finite set of variables var, any query in this work can be encoded as a word over $var \cup \mathcal{R} \cup \{\exists, \wedge, (,), \cup, \mathsf{OPT}\}$, where OPT is needed to encode wdPTs.

Since this encoding is always straight-forward and not important for this thesis, we will always omit it.

When we say linear time for a problem with input I, we mean that the number of operations is within $\mathcal{O}(||I||)$.

Computational Model

We adopt the *Random Access Machine* (RAM) model with uniform cost measure. For an input of size n, every register is of length $\mathcal{O}(\log(n))$. Operations such as addition of the values of two registers or concatenation can be performed in constant time. In contrast to the Turing model of computation, the RAM model with uniform cost measure can retrieve the content of any register via its unique address in constant time. This enables the construction of large lookup tables that can be queried within constant time.

Throughout this work, we use a variant of the RAM model named DRAM [Gra96], where the values stored in registers are polynomially bounded by n. As a consequence, the amount of available memory is polynomial in n. Some previous work mentioned in this thesis use a more restrictive variant of the model, called a DLINRAM [Gra96], where the values are at most $cn/log(n)$, and therefore the available memory is linear in the size of the input.

Grandjean proved that in the DLINRAM model sorting strings can be done in time $\mathcal{O}(n/\log n)$ where n is the size of the input containing strings encoded in some fixed alphabet and separated by some special symbol [Gra96]. We can use this method to sort relations. To construct the input to the sorting algorithm, let *dom* be the domain of a relation $R^{\mathcal{D}}$. We first translate the values from *dom* to a possibly smaller domain dom_R, containing only the values that appear in $R^{\mathcal{D}}$. Note that $|dom_R| \leq \text{arity}(R)|R^{\mathcal{D}}|$. Then, we translate these values to binary (since we are required to use a fixed alphabet), where each value takes $\log(dom_R)$ bits. The size of the input to the sorting problem is $n = \text{arity}(R) \cdot |R^{\mathcal{D}}| \cdot \log(|dom_R|) + (|R^{\mathcal{D}}| - 1)$. Therefore, we can sort the tuples of a relation in time $\mathcal{O}(n/\log n) = \mathcal{O}(||R^{\mathcal{D}}||)$. That is, it is possible to sort relations within linear time [SV17].

2.2.1 Decision Complexity

Let Σ be a finite alphabet. A *decision problem* Decide is a language over Σ, which we often denote by the following problem setting.

Decide	
INSTANCE:	$x \in \Sigma^*$.
QUESTION:	Is $x \in$ Decide?

A *decision complexity class* is a class of decision problems. The class PTIME is the class of all decision problems that can be solved in polynomial time, i.e. there exists a deterministic algorithm $\mathcal{A}$ such that for every $L \in \mathsf{PTIME}$ and every $x \in \Sigma^*$, $\mathcal{A}$ decides whether $x \in L$ in polynomial time. The class NP is defined analogously, but with a non-deterministic algorithm $\mathcal{A}$.

Given a CQ or a UCQ Q and a database instance $\mathcal{D}$ over a schema without FDs, we denote by $\textsc{Decide}\langle Q\rangle$ respectively $\textsc{Decide}_{\emptyset}\langle Q\rangle$ the problem of deciding whether $Q(\mathcal{D}) \neq \emptyset$. When we consider the problem in the presence of a (possibly empty) set Δ of FDs, the database $\mathcal{D}$ is defined over a schema $(\mathcal{R}, \Delta)$, and we denote the decision problem by $\textsc{Decide}_{\Delta}\langle Q\rangle$. For any query Q and database $\mathcal{D}$, deciding whether there exists a solution of Q over $\mathcal{D}$ is called the *Boolean evaluation problem.* In the case of wdPTs, we are usually more interested in the *evaluation* problem, which is the problem of given a query (respectively wdPT) p, database $\mathcal{D}$ and candidate solution μ, to decide whether μ is in $p(\mathcal{D})$.

Parameterized Complexity

A *parameterization* of Σ^* is a polynomial time computable mapping $\kappa\colon \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a pair (L, κ) where $L \subseteq \Sigma^*$ and κ is a parameterization of Σ^*. We refer to $x \in \Sigma^*$ as the instances of a problem, and to the numbers $\kappa(x)$ as the parameters. The following well-known problems will play an important role in our parameterized-complexity analyses in Chapter 6, and show how we usually denote parameterized problems.

p-Clique	
INSTANCE:	A graph G and $k \in \mathbb{N}$.
PARAMETER:	k
QUESTION:	Does G contain a clique of size k?

p-Dominating Set	
INSTANCE:	A graph G and $k \in \mathbb{N}$.
PARAMETER:	k
QUESTION:	Does G contain a dominating set of size k?

A parameterized problem $E = (L, \kappa)$ belongs to the class FPT of *fixed parameter tractable* problems if there exists an algorithm $\mathcal{A}$ deciding L, a polynomial p, and a computable function $f\colon \mathbb{N} \to \mathbb{N}$ such that the running time of $\mathcal{A}$ is at most $f(\kappa(x)) \cdot p(|x|)$.

Parameterized complexity theory also provides notions of intractability. Towards this notion, we first recall the definition of fpt-reductions. Let $E = (L, \kappa)$ and $E' = (L', \kappa')$ be parameterized problems over the alphabets Σ and Σ', respectively. An *fpt-reduction* from E to E' is a mapping $R\colon \Sigma^* \to (\Sigma')^*$ such that

1. for all $x \in \Sigma^*$ we have $x \in L$ if and only if $R(x) \in L'$,

2. there is a computable function f and a polynomial p such that $R(x)$ can be computed in time $f(\kappa(x)) \cdot p(|x|)$, and
3. there is a computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that $\kappa'(R(x)) \leq g(\kappa(x))$ for all $x \in \Sigma^*$.

One notion of intractability for parameterized problems are the classes $\mathsf{W}[i]$ (for $i \geq 1$) of the W-hierarchy. Since we are here only interested in the classes $\mathsf{W}[1]$ and $\mathsf{W}[2]$, we omit a discussion of the W-hierarchy and only recall the following facts: A parameterized problem E is in $\mathsf{W}[1]$ or $\mathsf{W}[2]$, if there exists an fpt-reduction to the problems p-Clique (for $\mathsf{W}[1]$) and p-Dominating Set (for $\mathsf{W}[2]$), respectively. Similarly, E is $\mathsf{W}[1]$-hard or $\mathsf{W}[2]$ if there exists an fpt-reduction from p-Clique and p-Dominating Set, respectively. It is strongly believed that problems that are hard for $\mathsf{W}[1]$ or $\mathsf{W}[2]$ are not in FPT. For details, see [FG10].

Computational Hypotheses

In decision complexity, it is common that lower bounds are proven relative to some common complexity theoretic assumption such as $\mathsf{PTIME} \neq \mathsf{NP}$ or $\mathsf{FPT} \neq \mathsf{W}[1]$. Many of the assumptions in this work are of a different nature, namely they assume a lower bounds for the specific run-time of computational problems, usually within the class PTIME. In fact, using such lower bounds for certain problems has gained a lot of interest in recent years [VW15, ABW18, LWW18, AW14, Pat10]. We will use the following well-established hypotheses for lower bounds:

- mat-mul: two Boolean $n \times n$ matrices A and B cannot be multiplied in time $\mathcal{O}(n^2)$.
  This problem is equivalent to the enumeration of all answers to the query $\Pi(x, y) \leftarrow A(x, z), B(z, y)$ over the schema $(\{A, B\}, \emptyset)$ and domain $\{1, \ldots, n\}$. It is strongly conjectured that this problem cannot be solved in $\mathcal{O}(n^2)$ time, and the best algorithms today require $\mathcal{O}(n^\omega)$ time for some $2.37 < \omega < 2.38$ [LG14, AW14].
- hyperclique: finding a k-hyperclique in a $(k-1)$-uniform graph is not possible in time $\mathcal{O}(n^{k-1})$ for all $k \geq 3$.
  This is a special case of the $(\ell, k)-$ Hyperclique Hypothesis [LWW18], which states that, in a k-uniform hypergraph of n vertices, $n^{k-o(1)}$ time is required to find a set of ℓ vertices such that each of it subsets of size k forms a hyperedge. We also denote the hyperclique hypothesis by Tetra$\langle k \rangle$, which is an equivalent problem introduced in [BB13].
- 4-clique: it is not possible to determine whether a 4-clique exists in a graph with n nodes in time $\mathcal{O}(n^3)$.
  This is a special case of the k-Clique Hypothesis [LWW18], which states that detecting a clique in a graph with n nodes requires $n^{\frac{\omega k}{3} - o(1)}$ time, where $\omega < 2.373$ is the matrix multiplication exponent.

- TRIANGLE: it is not possible to determine whether a graph with n nodes contains a triangle in time $\mathcal{O}(n^2)$. This is a special case of HYPERCLIQUE as well as the k-Clique Hypothesis and more common in the literature than the general versions of this problem, see [Lat08].

2.2.2 Enumeration Complexity

Given a finite alphabet Σ and binary relation $R \subseteq \Sigma^* \times \Sigma^*$, we denote by $\textsc{Enum}\langle R\rangle$ the *enumeration problem* of given an instance $x \in \Sigma^*$, to output all $y \in \Sigma^*$ such that $(x, y) \in R$. We will also denote this enumeration problem as follows:

$\textsc{Enum}\langle R\rangle$	
INSTANCE:	$x \in \Sigma^*$.
OUTPUT:	All $y \in \Sigma^*$ with $(x, y) \in R$.

For a query Q over a schema $S = (\mathcal{R}, \Delta)$, we denote by $\textsc{Enum}_\Delta\langle Q\rangle$ (or $\textsc{Enum}\langle Q\rangle$ in case $\Delta = \emptyset$) the enumeration problem $\textsc{Enum}\langle R\rangle$, where R is the binary relation between instances $\mathcal{D}$ over $\mathcal{S}$ and sets of answers $Q(\mathcal{D})$.

The set of all $y \in \Sigma^*$ forming the output of $\textsc{Enum}\langle R\rangle$ is defined as $\mathsf{Sol}_R(x)$ (or $\mathsf{Sol}(x)$ if R is clear from the context), which we also call set of all solutions to $\textsc{Enum}\langle R\rangle$ for x. A *parameterized enumeration problem* is a pair $(\textsc{Enum}\langle R\rangle, \kappa)$ such that κ is a parameterization of Σ^*.

An *enumeration algorithm* $\mathcal{A}$ for a (parameterized) enumeration problem $E = \textsc{Enum}\langle R\rangle$ (resp., $E = (\textsc{Enum}\langle R\rangle, \kappa)$) is an algorithm which, on input x, outputs exactly the elements from $\mathsf{Sol}(x)$ without duplicates. We denote the output of $\mathcal{A}$ on x by $\mathcal{A}(x)$.

Let $\mathcal{A}$ be an enumeration algorithm for some problem E. For an input x, let $n = |\mathcal{A}(x)|$. For $0 \leq i \leq n$, we define the *delay* of $\mathcal{A}$, denoted by $\mathit{delay}(i)$, as follows: $\mathit{delay}(0)$ ("preprocessing") is the time between the start of the algorithm and the (beginning of the) first output (or termination of $\mathcal{A}$, if $n = 0$). For $0 < i < n$, $\mathit{delay}(i)$ is the time between outputting solution i and $(i + 1)$. Finally, $\mathit{delay}(n)$ is the time between the last output and the termination of $\mathcal{A}$.

Enumeration Classes

The concept of the delay between outputs allows us to define several classes of enumeration problems that capture different notions of tractability for these problems [JPY88, CMM+17, Str10]. In the sequel, let $E = \textsc{Enum}\langle R\rangle$ (resp., $E = (\textsc{Enum}\langle R\rangle, \kappa)$) be a (parameterized) enumeration problem. For a class $\mathcal{C}$, we say that $E \in \mathcal{C}$ ($E' \in \mathcal{C}$, respectively), if there exists an enumeration algorithm $\mathcal{A}$ for E, some $m \in \mathbb{N}$, and a computable function f such that on every input x the following holds:

- $E \in \mathsf{OutputP}$: $\mathcal{A}$ terminates in time $\mathcal{O}((|x| + |\mathsf{Sol}(x)|)^m)$.
  Answers to E are enumerated within *output polynomial time*, or *polynomial total time* [JPY88].
- $E' \in \mathsf{OutputFPT}$: $\mathcal{A}$ terminates in time $\mathcal{O}(f(\kappa(x)) \cdot (|x| + |\mathsf{Sol}(x)|)^m)$.
  Answers to E are enumerated within *output fixed-parameter tractable time* [CMM+17].
- $E \in \mathsf{DelayP}$: *delay*(i) is in $\mathcal{O}(|x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
  Answers to E are enumerated with a *polynomial delay* [JPY88].
- $E \in \mathsf{DelayLin}$: *delay*(i) is in $\mathcal{O}(|x|)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
  Answers to E are enumerated with a *linear delay* [Bag06].
- $E' \in \mathsf{DelayFPT}$: *delay*(i) is in $\mathcal{O}(f(\kappa(x)) \cdot |x|^m)$ for every $0 \leq i \leq |\mathsf{Sol}(x)|$.
  Answers to E are enumerated with a *fixed-parameter tractable delay* [CMM+17].
- $E \in \mathsf{DelayC_{lin}}$: *delay*$(0)$ is in $\mathcal{O}(|x|)$ and for $0 < i \leq |\mathsf{Sol}(x)|$, *delay*$(i)$ is in $\mathcal{O}(1)$.
  Answers to E are enumerated with a *constant delay* after a *linear preprocessing* [DG07].

A stricter notion of allowing for polynomial delay between two solutions is captured by the class $\mathsf{SDelayP}$ (strict polynomial delay, [Str10]): An enumeration problem E is in $\mathsf{SDelayP}$ if there exists a total order $<$ on $\mathsf{Sol}(x)$, some $m \in \mathbb{N}$, and an algorithm $\mathcal{B}$ terminating in time $\mathcal{O}(|x|^m)$ with the following output: On input x, it returns the first element of $\mathsf{Sol}(x)$ (according to $<$). On input (x, y) for any $y \in \mathsf{Sol}(x)$, it either returns the next element according to $<$ if there is some, or halts otherwise. Also note that no restriction on the memory is used. In particular, for $\mathsf{DelayC_{lin}}$, any enumeration algorithm may use additional constant memory for writing between two consecutive answers. Only using the definition for the enumeration complexity classes, we immediately see that

$$\mathsf{DelayC_{lin}} \subseteq \mathsf{DelayLin} \subseteq \mathsf{DelayP} \subseteq \mathsf{OutputP}$$

respectively $\mathsf{DelayFPT} \subseteq \mathsf{OutputFPT}$ in the case of parameterized enumeration classes.

When aiming for enumeration complexity within $\mathsf{DelayC_{lin}}$ (that is, in Chapter 4 and 5 and parts of Chapter 6), all of our results hold under *data complexity*, i.e. for an input to a computational problem we assume the query to be fixed. When aiming for DelayP or any class containing DelayP, or when we consider any decision problems in Chapter 6, we will usually consider the *combined complexity* of a problem, thus considering the query as a variable part of the input.

Reduction between enumeration problems

Let Σ be a finite alphabet and R_1, R_2 be relations with $R_1, R_2 \subseteq \Sigma^* \times \Sigma^*$. There is an *exact reduction* from $\textsc{Enum}\langle R_1\rangle$ to $\textsc{Enum}\langle R_2\rangle$, denoted $\textsc{Enum}\langle R_1\rangle \leq_e \textsc{Enum}\langle R_2\rangle$, if there are mappings $\sigma : \Sigma^* \to \Sigma^*$ and $\tau : \Sigma^* \to \Sigma^*$ such that the following conditions hold.

- for every $x \in \Sigma^*$, the mapping $\sigma(x)$ is computable in $\mathcal{O}(|x|)$;
- for every $y \in \Sigma^*$ with $(\sigma(x), y) \in R_2$, $\tau(y)$ is computable in constant time;
- For every $x \in \Sigma^*$, the mapping $\pi : \mathsf{Sol}_{R_2}(\sigma(x)) \to \mathsf{Sol}_{R_1}(x)$ with $y \mapsto \tau(\sigma(x))$ is injective. Equivalently, the multiset $\{\tau(y) \mid y \in \Sigma^* \text{ with } (\sigma(x), y) \in R_2\}$ is equal to the multiset $\{y' \in \Sigma^* \mid (x, y') \in R_1\}$.

Intuitively, σ is used to map instances of $\textsc{Enum}\langle R_1\rangle$ to instances of $\textsc{Enum}\langle R_2\rangle$, and τ is used to map solutions to $\textsc{Enum}\langle R_2\rangle$ to solutions of $\textsc{Enum}\langle R_1\rangle$. The notation $\textsc{Enum}\langle R_1\rangle \equiv_e \textsc{Enum}\langle R_2\rangle$ means that $\textsc{Enum}\langle R_1\rangle \leq_e \textsc{Enum}\langle R_2\rangle$ and $\textsc{Enum}\langle R_2\rangle \leq_e \textsc{Enum}\langle R_1\rangle$. An enumeration class $\mathcal{C}$ is said to be *closed under exact reduction* if for every $\textsc{Enum}\langle R_1\rangle$ and $\textsc{Enum}\langle R_2\rangle$ with $\textsc{Enum}\langle R_2\rangle \in \mathcal{C}$ and $\textsc{Enum}\langle R_1\rangle \leq_e \textsc{Enum}\langle R_2\rangle$, we have that $\textsc{Enum}\langle R_1\rangle \in \mathcal{C}$. The third condition of the definition of exact reductions ensures that we can compute the solutions to an instance $x \in \Sigma^*$ of $\textsc{Enum}\langle R_1\rangle$ without duplicates by computing solutions $y \in \Sigma^*$ to the instance $\sigma(x)$ of $\textsc{Enum}\langle R_2\rangle$, and then computing $\tau(y)$ for every such $y \in \Sigma^*$. Using this approach, Bagan et al. [BDG07] proved the following:

Proposition 2.11 (Bagan et al. [BDG07])**.** $\mathsf{DelayC}_{\mathsf{lin}}$ *is closed under exact reduction.*

The proof for this proposition also holds for any meaningful enumeration complexity class that guarantees generating all unique answers with at least linear preprocessing time and at least constant delay between answers.

CHAPTER 3

State of the Art

In this chapter, we will give an overview of relevant parts of enumeration complexity and discuss further work related to this thesis. As large parts of this thesis are motivated by the dichotomy by Bagan et al. [BDG07], we will sketch the basic idea of the proof, both the negative as well as the positive part of it, and present other results on CQ enumeration. After that, we will discuss previous as well as recent work on constant delay enumeration related to this thesis. Moreover, we will give an overview on current work on wdPTs.
For an excellent survey on the topic of constant delay enumeration for a variety of queries, see [Seg15].

3.1 Enumeration Complexity

While decision problems often ask for the existence of a solution to some problem instance, enumeration problems aim at outputting all solutions. To capture the intuition of easy to enumerate problems – despite a possibly exponential number of output values – various notions of tractable enumeration classes have been proposed over the years. The first such notion has been introduced by Johnson et al. [JPY88], defining the classes of DelayP and $\mathsf{OutputP}$ among others. When evaluating the enumeration complexity of a query, it is common to assume data complexity, i.e. to assume that the size of the query is fixed. We make the same assumption in Chapters 4, 5 and parts of Chapter 6. When making this assumption, Durand and Grandjean [DG07] introduced the class of $\mathsf{DelayC}_{\mathsf{lin}}$, to have a more fine-grained measurement for the enumeration complexity. This can be seen as the optimal class for nontrivial enumeration problems.
Another way of evaluating the complexity of an enumeration problem is to make use of parameterized complexity [CMM$^+$17, Fer02, Dam06, CV15]. Among other classes, parameterized enumeration lifts the classes of DelayP and $\mathsf{OutputP}$ in a natural way to the classes $\mathsf{DelayFPT}$ and $\mathsf{OutputFPT}$. These classes are defined analogously to DelayP

and OutputP, but with a fixed-parameter polynomial time delay between answers instead of just a polynomial delay, respectively with a fixed-parameter polynomial total time to output all answers instead of just polynomial total time.
A new trend in enumeration complexity is to allow for updates of the underlying data during the enumeration process [LM14, BKS17, IUV17, NS18]. Although only loosely related to the topic of this thesis, we will briefly discuss enumeration with updates in Section 3.3.

When evaluating the efficiency of an enumeration algorithm $\mathcal{A}$, one is usually not content with showing membership in an enumeration complexity class E. The goal is to show that the enumeration in any class $E' \subsetneq E$ is not possible, making algorithm $\mathcal{A}$ in some sense optimal. However, for showing such lower bounds, enumeration complexity has very few tools to offer.
The most common way in the literature is to make use of decision complexity, and show that outputting a first solution is already hard, under reasonable complexity assumptions such as PTIME $\neq$ NP. A more general version of this approach is to show that the following problem is hard: Given a set of solutions, is there another one? This result is discussed in [Str10, Lemma 2.11], and can also be found in [KSS00].
To prove lower bounds for constant delay enumeration, a more fine grained approach for decision complexity is needed as well. Since common complexity assumptions such as PTIME $\neq$ NP are not helpful when trying to distinguish the hardness of solving problems within PTIME, certain well-established and well-studied problems such as the Boolean matrix multiplication problem or the 3-Sum problem are used to introduce conditional lower bounds within the class PTIME [VW15, WW10, LWW18, AW14]. Several results on constant delay enumeration rely on such hypotheses, see [BDG07, BB13, BKS17].
Other approaches in enumeration complexity for showing lower bounds, which are not within the scope of this thesis, are to use the ETH [LMS+13] to separate classes [CS18], to reduce problems to the transversal-hypergraph problem [EG02, KLMN14] or to use reductions for enumeration problems to show hardness for enumeration complexity classes analogously to that of the polynomial hierarchy for decision complexity classes [CKP+17].

3.2 Enumerating the Answers to a CQs

We next turn our attention to the enumeration problem for CQs. For combined complexity, we can use a standard technique for enumeration (see Section 6.3.1 for a short discussion) to enumerate the answers to a CQ with a linear delay, see [BDG07, Theorem 13]. When aiming for the class $\mathsf{DelayC}_{\mathsf{lin}}$ under data complexity, we first note that tractable enumeration is not possible for the unrestricted class of CQs. This is due to results in parameterized complexity. Indeed, the problem of answering Boolean CQs, taking the query size as the parameter, is W[1]-complete [PY99]. Thus, if we could enumerate all answers with linear preprocessing and constant delay, we can produce a potential first output or decide that no such output exists in FPT time. Assuming that $FPT \neq \mathsf{W}[1]$, this is not possible. For the class of acyclic CQs, the Boolean query evaluation problem is in FPT [Yan81]. However, due to the dichotomy by Bagan et al., even making the restriction to acyclic CQs is not enough:

Theorem 3.1 ([BDG07, Theorem 27])**.** *Let Q be an acyclic CQ without self-joins.*

1. *If Q is free-connex, then* $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.
2. *If Q is not free-connex, then* $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$*, assuming that two Boolean $n \times n$ matrices cannot be multiplied in time $\mathcal{O}(n^2)$.*

This result holds assuming a mode of computation that is slightly more restrictive than the one presented in this thesis: Memory is only allocated during the preprocessing step of an enumeration algorithm, leading to at most linear space overall. In this work, we keep the time bounds of constant delay and linear preprocessing, while an enumeration algorithm may use additional constant memory for writing between two consecutive answers. For a discussion on the different models of constant delay enumeration, see [Kaz13].

The positive part of the dichotomy for CQs was presented as a result on acyclic conjunctive functional queries with disequalities. A more intuitive way of understanding the enumeration algorithm is given by an extended version of the well-known Yannakakis algorithm [Yan81]. This extended algorithm, presented in [IUV17], exploits the free-connex structure of a CQ Q as follows: Since Q is free-connex, there exists a join-tree T of a hypergraph only containing sub-edges of $\mathcal{H}(Q)$ (which is an *inclusive extension* of $\mathcal{H}(Q)$, see Section 2.1), such that a subtree T' of T contains all free-variables. Choosing a node in this subtree as a root node paired with an index structure built in the preprocessing phase thus allows for constant delay enumeration by basically applying the Yannakakis algorithm.

The lower bound of Theorem 3.1 is proven by using a characterization of free-connexity, which we also frequently use in chapters 4 and 5:

Theorem 3.2 ([BDG07, Lemma 24])**.** *Let $\mathcal{H} = (V, E)$ be an acyclic hypergraph and $S \subseteq V$. Then H is ext-S-connex if and only if it does not contain an S-path.*

By using this result, Bagan et al. show that if an acyclic CQ Q is not free-connex, then we can take some of the relations that correspond to the first and last part of the free(Q)-path in $\mathcal{H}(Q)$, and encode the matrix multiplication problem to these two relations. This idea is crucial for some of the lower bounds we show in this work as well.

Cyclic CQs

The dichotomy above only gives insight to the enumeration complexity of acyclic CQs. Concerning unrestricted CQs, providing even a first solution of a query in linear time is impossible in general, as we have pointed out before. This does not imply, however, that there are no cyclic queries with the corresponding enumeration problems in $\mathsf{DelayC_{lin}}$.

The fact that no such queries exist requires an additional proof, which was presented by Brault-Baron [BB13]. For a definition of the hypothesis HYPERCLIQUEsee Section 2.2.1.

Theorem 3.3 ([BB13, Theorem 12, p.78])**.** *Let Q be a cyclic CQ without self-joins. Then* $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$*, assuming* HYPERCLIQUE.

A way to still achieve a constant delay enumeration algorithm is to aim for a less restrictive preprocessing phase, as the theorem above only works for a linear preprocessing phase. This approach was already taken by Bagan et al. [BDG07], where they make the natural generalization from free-connex acyclicity to that of bounded free-connex treewidth, see 6.1.1, Definition 6.6.

3.3 Constant Delay Enumeration of Related Query Languages

Following the results by Bagan et al. [BDG07], there has been much work on query enumeration in $\mathsf{DelayC_{lin}}$ over the last few years. Most importantly, there are results for query languages more general than UCQs. The restrictions made to achieve those results however are made on the database. In this thesis, we do not impose any restrictions on the database at all, thus none of these results directly imply results on UCQ enumeration. For the sake of completeness, we briefly mention some of the results.

Restrictions to the Database

Restrictions on a database I are imposed similar to the restrictions on a query Q: by building a graph G that corresponds to the database I. That is, the Gaifman graph of I is defined as $G = (V, E)$, where V is the set of domain elements over which I is defined, and (a, b) is an edge in E if and only if both a and b occur in a tuple of a relation R^I in the database.
The class of databases for which constant delay results are achieved are usually either sparse graphs, or graphs with bounded treewidth. Among other results, for first-order (FO) queries, enumeration within $\mathsf{DelayC_{lin}}$ is possible over classes of databases of bounded degree [DG07] or over classes of databases of (local) bounded expansion [KS13a, SV17]. Moreover, constant delay enumeration with a preprocessing in $\mathcal{O}(n^{1+\epsilon})$ is possible for FO queries over nowhere dense graphs[SSV18]. For monadic second-order (MSO) queries, we have tractability over classes of databases of bounded tree-width[KS13b].

Answering Queries under Updates

A slightly different approach for enumeration complexity has gained a lot of recent interest, as it takes advantage of the dynamic nature of enumeration itself. Enumerating the answers of a query under updates considers dynamic databases, where during the enumeration process, tuples can be inserted as well as deleted [BKS17, BKS18, IUV17, NS18]. As in the original dichotomy on CQ enumeration, there have been results on answering CQs [BKS17] as well as UCQs [BKS18] under updates, where the restriction is made on the structure of the query itself, and not the database. For CQs, as the enumeration problem itself is more general, the class of tractable queries is a proper subclass of free-connex queries.

For UCQ enumeration under updates, a recent work [BKS18] by Berkholz et al. considers enumeration under integrity constraints. The three different integrity constraints that are considered are small domain constraints, inclusion dependencies as well as FDs. For FDs, Berkholz et al. note that an intractable query can become tractable using the dependency. The search for an equivalent form of a CQ or UCQ under FDs is mentioned as an open problem.

The same work also claims a dichotomy for UCQ enumeration under updates, with a general proof for the lower bound that can also be applied for the setting of non-dynamical UCQ enumeration. Unfortunately, this proof is not correct, and we briefly point out why. In fact, Example 1.3 is already a counter example to the claim made in [BKS18, Theorem 4.2b]. Intuitively, the claim is that if a UCQ contains an intractable CQ and does not contain redundant CQs (a CQ contained in another CQ in the union), then the union itself is already intractable. In contrast, none of the CQs in Example 1.3 is redundant, Q_1 is intractable, and yet the UCQ is tractable.

The intuition behind the proof of the past claim is reducing the hard CQ Q_1 to Q. This can be done by assigning each variable of Q_1 with a different and disjoint domain (e.g., by concatenating the variable names to the values in the relations corresponding to the atoms), and leaving the relations that do not appear in the atoms of Q_1 empty. It is well known that $Q_1 \subseteq Q_2$ if and only if there exists a homomorphism from Q_2 to Q_1. The claim is that since there is no homomorphism from another CQ in the union to Q_1, then there are no answers to the other CQs with this reduction. However, it is possible that there is a body-homomorphism from another CQ to Q_1 even if it is not a full homomorphism (the free variables do not map to each other). Therefore, in cases of a body-homomorphism, the reduction from the Q_1 to Q does not work. In such cases, the union may be tractable, as we will show in Chapter 5.

3.4 Complexity of wdPTs

With the steadily increasing amount of inaccurate and incomplete data on the web, the need for partial matching as an extension of CQs is gaining more and more importance. Therefore, in the semantic web query language SPARQL, the OPT operator is a crucial feature. This operator has also been studied for arbitrary relational vocabulary [BPS15]. Pérez et al. [PAG09] pointed out that unconstrained SPARQL queries containing both the AND and OPT operator may cause some undesired behavior. This led to the definition well-designed AND,OPT-SPARQL. Among other benefits such as better complexity results, this language allows for a natural tree-representation of the queries, known as well-designed pattern trees [LPPS13].

Several aspects of the complexity of wdPTs have been studied in previous works. The query evaluation problem was shown coNP-complete for wdPTs without projection [PAG09] and Σ_2^P-complete for wdPTs with projection [LPPS13]. Wrapping wdPTs into the CONSTRUCT operator of SPARQL makes the evaluation problem NP-complete [KRU15]. In [AFP+15, AU16], the max-evaluation problem was introduced as an important variant of the evaluation problem: it asks if a given mapping is a *maximal* solution (i.e., it cannot be properly extended to another solution). This problem is DP-complete [AFP+15]. Important query analysis tasks such as the containment and equivalence problems in various settings were studied in [PS14]. It was shown that the complexity of these tasks ranges from NP-completeness (if we disallow projection) to undecidability (if projection is allowed in both queries). The enumeration problem however, as mentioned in the introduction, has been hardly considered so far with the exception of [LPPS13].

CHAPTER 4

Conjuntive Queries in the Presence of Functional Dependencies

In this chapter, we aim to lift known results on enumeration complexity of CQs to the setting of schemas with functional dependencies.

4.1 FD-Extended CQs

We start this section with formally defining the extended query Q^+. We then discuss the relationship between Q and Q^+: their equivalence w.r.t. enumeration and the possible structural differences between them. As a result, we obtain that if Q^+ is in a class of queries that allows for tractable enumeration, then Q is tractable as well.

We first define Q^+. The *extension* of an atom $R(\vec{v})$ according to an FD $S\colon A \to b$ and an atom $S(\vec{u})$ is possible if $\vec{u}[A] \subseteq \vec{v}$ but $\vec{u}[b] \notin \vec{v}$. In this case, $\vec{u}[b]$ is added to the variables of R. The *FD-extension* of a query is defined by iteratively extending all atoms as well as the head according to every possible dependency in the schema, until a fix-point is reached. The schema extends accordingly: the arities of the relations increase as their corresponding atoms extend, and the FDs apply in every relation that contains all relevant variables. Dummy variables are added to adjust to the change in arity in case of self-joins.

Definition 4.1 (FD-Extended Query)**.** Let $Q(\vec{p}) \leftarrow R_1(\vec{v_1}), \dots, R_m(\vec{v_m})$ be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$. We define two types of extension steps:

- The extension of an atom $R_i(\vec{v_i})$ according to an FD $R_j\colon A \to b$.
  Prerequisites: $\vec{v_j}[A] \subseteq \vec{v_i}$ and $\vec{v_j}[b] \notin \vec{v_i}$.

Effect: The arity of R_i increases by one, $R_i(\vec{v_i})$ is replaced by $R_i(\vec{v_i}, \vec{v_j}[b])$. In addition, every $R_k(\vec{v_k})$ such that $R_k{=}R_i$ and $k \neq i$ is replaced with $R_k(\vec{v_k}, t_k)$, where t_k is a fresh variable in every such step.

- The extension of the head $Q(\vec{p})$ according to an FD $R_j \colon A \to b$.
Prerequisites: $\vec{v_j}[A] \subseteq \vec{p}$ and $\vec{v_j}[b] \notin \vec{p}$.
Effect: The head is replaced by $Q(\vec{p}, \vec{v_j}[b])$.

The *FD-extension* of Q is the query $Q^+(\vec{q}) \leftarrow R_1^+(\vec{u_m}), \ldots, R_m^+(\vec{u_m})$, obtained by performing all possible extension steps on Q according to FDs of Δ until a fix-point is reached. The extension is defined over the schema $\mathcal{S}^+ = (\mathcal{R}^+, \Delta_{Q^+})$, where $\mathcal{R}^+$ is $\mathcal{R}$ with the extended arities, and Δ_{Q^+} is

$$\{R_i^+ \colon C \to d \mid \exists (R_j \colon A \to b) \in \Delta, \exists R_i^+(\vec{u_i}) \in \text{atoms}(Q^+), \exists R_j(\vec{v_j}) \in \text{atoms}(Q),$$
$$\text{s.t. } \vec{u_i}[C] = \vec{v_j}[A] \text{ and } \vec{u_i}[d] = \vec{v_j}[b]\}.$$

Given a query, its FD-extension is unique up to a permutation of the added variables, and renaming of the new variables. As the order of the variables and the naming make no difference w.r.t. enumeration, we can treat the FD-extension as unique.

Example 4.2. Consider a schema with $\Delta = \{R_1 \colon 1 \to 2, R_3 \colon 2, 3 \to 1\}$, and the query $Q(x) \leftarrow R_1(x, y), R_2(x, z), R_2(u, z), R_3(w, y, z)$. As the FDs are $x \to y$ and $yz \to w$, the FD-extension is

$$Q^+(x, y) \leftarrow R_1^+(x, y), R_2^+(x, z, y, w), R_2^+(u, z, t_1, t_2), R_3^+(w, y, z).$$

We first apply $x \to y$ on the head, and then $x \to y$ and consequently $yz \to w$ on $R_2(x, z)$. These two FDs are now in the schema also for R_2, and the FDs of the extension are $\Delta_{Q^+} = \{R_1^+ \colon 1 \to 2, R_2^+ \colon 1 \to 3, R_2^+ \colon 3, 2 \to 4, R_3^+ \colon 2, 3 \to 1\}$. □

We later show that the enumeration complexity of a CQ Q over a schema with FDs only depends on the structure of Q^+, which is implicitly given by Q and its schema. Therefore, we introduce the notions of acyclic and free-connex queries for FD-extensions:

Definition 4.3. Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, and let Q^+ be its FD-extension.

- We say that Q is *FD-acyclic*, if Q^+ is acyclic.
- We say that Q is *FD-free-connex*, if Q^+ is free-connex.
- We say that Q is *FD-cyclic*, if Q^+ is cyclic.

The following proposition shows that the classes of acyclic queries and free-connex queries are both closed under constructing FD-extensions.

Proposition 4.4. *Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$.*

- *If the query Q is acyclic, then it is FD-acyclic.*
- *If the query Q is free-connex, then it is FD-free-connex.*

Proof. We prove that if Q is acyclic, then Q^+ is also acyclic, by constructing a join-tree of $\mathcal{H}(Q^+)$ given one of $\mathcal{H}(Q)$. The same proof can be applied to a join tree containing the head to show that if Q is free-connex, then so is Q^+. Denote by $Q = Q_0, Q_1, \dots, Q_n = Q^+$ a sequence of queries such that Q_{i+1} is the result of extending all possible relations of Q_i according to a single FD $\delta \in \Delta$. By induction, it suffices to show that if $\mathcal{H}(Q_i)$ has a join tree, then $\mathcal{H}(Q_{i+1})$ has one too. So consider an acyclic query $Q_i(\vec{p}) \leftarrow R_1(\vec{v}_1), \dots, R_m(\vec{v}_m)$ extended to the query $Q_{i+1}(\vec{q}) \leftarrow R_1(\vec{u}_1), \dots, R_m(\vec{u}_m)$ according to the FD $\delta = R_j : \vec{x} \to y$, and let $T_i = (V_i, E_i)$ be a join tree of $\mathcal{H}(Q_i)$. We claim that the same tree (but with the extended atoms), is a join tree for Q_{i+1}. Formally, define $T_{i+1} = (V_{i+1}, E_{i+1})$ such that $V_{i+1} = \{R_k(\vec{u}_k) \mid 1 \leq k \leq m\}$ and $E_{i+1} = \{(R_k(\vec{u}_k), R_l(\vec{u}_l)) \mid (R_k(\vec{v}_k), R_l(\vec{v}_l)) \in E_i\}$. Next we show that the running intersection property holds in T_{i+1}, and therefore it is a join tree of $\mathcal{H}(Q_{i+1})$.

Every new variable introduced in the extension appears only in one atom, so the subtree of T_{i+1} containing such a variable contains one node and is trivially connected. For any other variable $w \neq y$, the attribute w appears in the same atoms in Q and Q^+. Therefore, the subgraph of T_{i+1} containing w is isomorphic to the subgraph of T_i containing w, and since T_i is a join tree, it is connected. It is left to show that the subtree of T_{i+1} containing y is connected. Since R_j is an atom in Q containing δ, it corresponds to vertices in T_i and T_{i+1} containing $\vec{x} \cup \{y\}$. Let R_k be some vertex in T_{i+1} containing y. We will show that all vertices $S_1, \dots, S_r$ on the path between R_k and R_j contain y. If y appears in the vertex R_k in T_i, then it also appears in $S_1, \dots, S_r$ since T_i is a join tree. Since the extension does not remove occurrences of variables, y appears in these vertices in T_{i+1} as well. Otherwise, y was added to R_k via δ, so R_k contains $\vec{x}$. Since T_i is a join tree, the vertices $S_1, \dots, S_r$ all contain the variables $\vec{x}$. Thus by the definition of Q_{i+1}, y is added to each of $S_1, \dots, S_r$ (if it was not already there) in T_{i+1}. Thus also the subtree of T_{i+1} containing y is connected. Therefore T_{i+1} is indeed a join tree. $\square$

By recalling Example 1.2, with new notation for readability, we see that the converse of the proposition above does not hold.

Example 4.5. Consider the query $Q(x, y) \leftarrow R_1(x, z), R_2(z, y)$ over a schema with the FD $R_2 : z \to y$. The CQ Q is acyclic and not free-connex. The FD-extension of this query is given by $Q^+(x, y) \leftarrow R_1^+(x, z, y), R_2^+(z, y)$, thus Q^+ is free-connex and Q is FD-free-connex.

This means that, by Theorem 3.1, there are queries Q such that enumerating the answers to Q^+ is in $\mathsf{DelayC_{lin}}$, but the same task cannot be done for Q with the same complexity if we do not assume the existence of FDs. The following theorem shows that, when relying on the FDs, enumerating the answers to Q^+ is equally hard as enumerating the answers to Q.

Theorem 4.6. *Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, and let Q^+ be its FD-extended query. Then,* $\textsc{Enum}_{\Delta}\langle Q\rangle \equiv_e \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$.

Proof. Consider a query $Q(\vec{p}) \leftarrow R_1(\vec{v}_1), \dots, R_m(\vec{v}_m)$ and its FD-extension $Q^+(\vec{q}) \leftarrow R_1^+(\vec{u}_1), \dots, R_m^+(\vec{u}_m)$. We show the two parts of the equivalence.

Claim. $\textsc{Enum}_{\Delta}\langle Q\rangle \leq_e \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$.

Construction. Given an instance I for $\textsc{Enum}_{\Delta}\langle Q\rangle$, we construct an instance $\sigma(I)$ for $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$ with two phases: cleaning and extension. In the cleaning phase, we remove tuples that interfere with the extended dependencies. For every dependency $\delta = R_j\colon X \to y$ and every atom $R_k(\vec{v}_k)$ that contains the corresponding variables (i.e., $X \cup \{y\} \subseteq \vec{v}_k$), we correct R_k according to δ: we only keep tuples of R_k^I that agree with some tuple of R_j^I over the values of $X \cup \{y\}$. We denote the cleaned instance by I_0. The cleaning phase can be done in linear time by first sorting both R_j^I and R_k^I according to $X \cup \{y\}$, and then performing one scan over both of them. Next, we perform the extension phase. We follow the extension of the schema as described in Definition 4.1 and extend the instance accordingly. This phase results in a sequence of instances $I_0, I_1, \dots, I_n = \sigma(I)$ that correspond to a sequence of queries $Q = Q_0, Q_1, \dots, Q_n = Q^+$ such that each query is the result of extending an atom or the head of the previous query according to an FD. If in step i the head was extended, we set $I_{i+1} = I_i$. Now assume some relation R_k is extended according to some FD $R_j\colon X \to y$. For each tuple $t \in R_k^{I_i}$, if there is no tuple $s \in R_j^{I_i}$ that agrees with t over the values of X, then we remove t altogether. Otherwise, we copy t to $R_k^{I_{i+1}}$ and assign y with the same value that s assigns it. The extension phase takes linear time for each step. Since the number of FDs is constant in data complexity, the overall construction takes linear time. Note that this construction ensures that the extended dependencies hold in $\sigma(I)$. Given an answer $\mu|_{\text{free}(Q^+)} \in Q^+(\sigma(I))$, we set $\tau(\mu) = \mu|_{\text{free}(Q)}$. This projection only requires constant time.

Correctness. We now show that $Q(I) = \{\mu|_{\text{free}(Q)} : \mu|_{\text{free}(Q^+)} \in Q^+(I^+)\}$. First, if $\mu|_{\text{free}(Q^+)}$ is an answer of $Q^+(I^+)$, then μ is a homomorphism from Q^+ to I^+. Since all tuples of I^+ appear (perhaps projected) in I, then μ is also a homomorphism from Q to I, and $\mu|_{\text{free}(Q)} \in Q(I)$. It is left to show the opposite direction: if $\mu|_{\text{free}(Q)} \in Q(I)$ then $\mu|_{\text{free}(Q^+)} \in Q^+(I^+)$. We show by induction on $Q = Q_0, Q_1, \dots, Q_n = Q^+$ that $\mu|_{\text{free}(Q_i)} \in Q_i(I_i)$. The induction base holds since in the cleaning phase we did not remove "useful" tuples. Since $\mu|_{\text{free}(Q)} \in Q(I)$, there exist tuples, one of each relation of the query, that agree on the values of $X \cup \{y\}$ (they all assign them with the values μ assigns them). These tuples were not removed in the cleaning phase, and therefore $\mu|_{\text{free}(Q)} \in Q(I_0)$. Next assume that $\mu|_{\text{free}(Q_i)} \in Q_i(I_i)$, and we want to show that $\mu|_{\text{free}(Q_{i+1})} \in Q_{i+1}(I_{i+1})$.

This claim is trivial in case the head was extended. Note also that there cannot be two distinct answers $\mu|_{\text{free}(Q_{i+1})}$ and $\mu'|_{\text{free}(Q_{i+1})}$ in $Q_{i+1}(I_{i+1})$ such that $\mu|_{\text{free}(Q_i)} = \mu'|_{\text{free}(Q_i)}$, as the added variable is bound by the FD to have only one possible value. Now consider the case where an atom $R_k(\vec{v}_k)$ was extended according to an FD $R_j\colon X \to y$ since $X \subseteq \vec{v}_k$. The tuple $\mu(\vec{v}_k) \in R_k^{I_i}$ was extended with the value $\mu(y)$ due to the tuple $\mu(\vec{v}_j) \in R_j^{I_i}$ that agrees with it on the values of X, and so $\mu(\vec{v}_k, y) \in R_k^{I_{i+1}}$. In case of self-joins, other atoms with the relation R_k are extended with a new and distinct variable. Such variables will be mapped to this value $\mu(y)$ as well. Overall, we have that μ (extended by mappings of the fresh variables) is also a homomorphism in $Q_{i+1}(I_{i+1})$.

Claim. $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \leq_e \text{Enum}_{\Delta}\langle Q\rangle$.

Construction. Given an instance I^+ for $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$, we construct an instance $\sigma(I^+)$ for $\text{Enum}_{\Delta}\langle Q\rangle$ with three phases: cleaning, building a lookup table and projection. In the cleaning phase, we remove tuples that do not contribute to the answer set of Q^+ in order to prevent additional answers from appearing in Q after the projection. This can be seen as unifying the restrictions of different FDs in Δ_{Q^+} that originate in the same FD in Δ. For every FD $R_j\colon X \to y$ in Δ and every atom $R_k^+(\vec{u}_k)$ such that $X \cup \{y\} \subseteq \vec{u}_k$, we remove all tuples $t \in R_k^{+I^+}$ that agree with some tuple $s \in R_j^{+I^+}$ over X but disagree with s over y. The cleaning phase can be done in linear time by first sorting both $R_k^{+I^+}$ and $R_j^{+I^+}$ according to X. Next, we construct a lookup table T to later reconstruct the assignments to $\text{free}(Q^+) \setminus \text{free}(Q)$. For every $y \in \text{free}(Q^+) \setminus \text{free}(Q)$ added to the head due to an FD $R_j\colon X \to y$, denote by $\vec{x}$ a vector containing the variables of X in lexicographic order. For every tuple in $R_j^{+I^+}$ that assigns y and $\vec{x}$ with the values y_0 and $\vec{x}_0$ respectively, we set $T(\vec{x}, \vec{x}_0, y) = y_0$. Note that due to the FD, a key cannot map to two different values. We conclude the construction by projecting the relations of I^+ according to the schema of Q. These steps result in the construction of an instance $\sigma(I^+)$ and a lookup table T in linear time. Note that Δ hold in $\sigma(I^+)$ since Δ_{Q^+} contains them.

Given $\mu|_{\text{free}(Q)} \in Q(I)$, we define $\tau(\mu|_{\text{free}(Q)}) = \mu|_{\text{free}(Q)} \cup \nu_\mu$, where the mapping ν_μ : $\text{free}(Q^+) \setminus \text{free}(Q) \to \textit{dom}$ uses the lookup table: For every $y \in \text{free}(Q^+) \setminus \text{free}(Q)$ added due to some FD $R_j\colon X \to y$, we set $\nu_\mu(y) = T[(\vec{x}, \mu(\vec{x}), y)]$. Note that τ is computable in constant time since we use the lookup table $|\,\text{free}(Q^+) \setminus \text{free}(Q)|$ times, and each access takes constant time.

Correctness. We first claim that the lookup table succeeds in reconstructing the values for the missing head variables: if $\mu|_{\text{free}(Q)} \in Q(\sigma(I^+))$, then $\tau(\mu|_{\text{free}(Q)}) = \mu|_{\text{free}(Q^+)}$. By definition, for every $y \in \text{free}(Q)$, $\tau(\mu(y)) = \mu(y)$. We need to show the same for $y \in \text{free}(Q^+) \setminus \text{free}(Q)$. In this case, y was added to the head due to some FD $R_j\colon X \to y$, and $\tau(\mu(y))$ is defined to be $\nu_\mu(y) = T[(\vec{x}, \mu(\vec{x}), y)]$. Since μ is a homomorphism into $\sigma(I^+)$, there exists some tuple in $R_j^{\sigma(I^+)}$ that assigns $\vec{x}$ and y with $\mu(\vec{x})$ and respectively $\mu(y)$. This tuple is a projection of a tuple in $R_j^{I^+}$ that assigns $\vec{x}$ and y with the same values. Due to this tuple, when constructing the lookup table, we set $T[(\vec{x}, \mu(\vec{x}), y)] = \mu(y)$.

We now show that $Q^+(I^+) = \{\mu|_{\text{free}(Q^+)} : \mu|_{\text{free}(Q)} \in Q(\sigma(I^+))\}$. The first direction is that if $\mu|_{\text{free}(Q^+)} \in Q^+(I^+)$, then $\mu|_{\text{free}(Q)} \in Q(\sigma(I^+))$. Indeed, if μ maps the relations $R_i^+(\vec{u}_i)$ to the tuples s_i respectively, these tuples agree on the value of y for every FD $X \to y$ in Δ_{Q^+}, and so none of them were removed in the cleaning phase. After the projection, μ maps $R_i(\vec{v}_i)$ to the same (perhaps projected) tuples in $R_i^{\sigma(I^+)}$, and so it is still a homomorphism in Q. The second direction is that if $\mu|_{\text{free}(Q)} \in Q(\sigma(I^+))$, then $\mu|_{\text{free}(Q^+)} \in Q^+(I^+)$. We know that μ is a homomorphism from Q to $\sigma(I^+)$. For every $1 \leq i \leq m$, denote by t_i the tuple $t_i = \mu(\vec{v}_i)$. These tuples are projections of tuples in I^+. Denote by s_i the tuple in $R_i^{I^+}$ whose projection is t_i. If an atom R_k was extended due to an FD $R_j \colon X \to y$, then t_j and s_k must agree on y, otherwise this s_k would have been deleted in the cleaning phase. Therefore, s_k assigns y with $\mu(y)$. We conclude that $\mu|_{\text{free}(Q^+)} \in Q^+(I^+)$. □

As an immediate result of Theorem 4.6, FD-extensions can be used to expand tractable enumeration classes.

Corollary 4.7. *Let $\mathcal{C}$ be an enumeration class that is closed under exact reduction. Let Q be a CQ and let Q^+ be its FD-extension. If* $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \in \mathcal{C}$*, then* $\text{Enum}_{\Delta}\langle Q\rangle \in \mathcal{C}$.

Proof. According to Theorem 4.6, $\text{Enum}_{\Delta}\langle Q\rangle \leq_e \text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$. Since $\mathcal{C}$ is closed under exact reduction, if $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \in \mathcal{C}$, then $\text{Enum}_{\Delta}\langle Q\rangle \in \mathcal{C}$. □

Since free-connex queries are in $\mathsf{DelayC_{lin}}$ and $\mathsf{DelayC_{lin}}$ is closed under exact reduction, we get the following corollary.

Corollary 4.8. *Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$. If Q is FD-free-connex, then* $\text{Enum}_{\Delta}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.

Proof. By definition of FD-free-connexity, Q^+ is free-connex, and thus according to Theorem 3.1 we have that $\text{Enum}_{\emptyset}\langle Q^+\rangle \in \mathsf{DelayC_{lin}}$. Given an instance over the schema $(\mathcal{R}, \Delta_{Q^+})$, the same instance is also valid over $(\mathcal{R}, \emptyset)$, so using the identity mapping shows that $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \leq_e \text{Enum}_{\emptyset}\langle Q^+\rangle$. Therefore, $\text{Enum}_{\Delta_{Q^+}}\langle Q\rangle \in \mathsf{DelayC_{lin}}$, and from Corollary 4.7 we get that $\text{Enum}_{\Delta}\langle Q\rangle \in \mathsf{DelayC_{lin}}$. □

We can now revisit Example 1.2. The query $Q(x, y) \leftarrow R_1(z, x), R_2(z, y)$ is not free-connex. Therefore, ignoring the FDs, according to Theorem 3.1 it is not in $\mathsf{DelayC_{lin}}$. However, given $R_2 \colon z \to y$, the FD-extended query is $Q^+(x, y) \leftarrow R_1^+(z, y, x), R_2^+(z, y)$. As it is free-connex, enumerating the answers to Q^+ is in $\mathsf{DelayC_{lin}}$ by Corollary 4.8.

4.2 A Dichotomy for Acyclic CQs

In this section, we characterize which self-join-free FD-acyclic queries are in $\mathsf{DelayC_{lin}}$. We use the notion of FD-extended queries defined in the previous section to establish a dichotomy stating that enumerating the answers to an acyclic query is in $\mathsf{DelayC_{lin}}$ if and only if the query is FD-free-connex. The positive case for the dichotomy is described in Corollary 4.8, and this section concludes the negative case. We prove the following theorem:

Theorem 4.9. *Let Q be a self-join-free FD-acyclic CQ over the schema $\mathcal{S} = (\mathcal{R}, \Delta)$.*

- *If Q is FD-free-connex, then* $\textsc{Enum}_\Delta\langle Q\rangle \in \mathsf{DelayC_{lin}}$.
- *If Q is not FD-free-connex, then* $\textsc{Enum}_\Delta\langle Q\rangle \notin \mathsf{DelayC_{lin}}$, *assuming* MAT-MUL.

Note that the restriction of considering only self-joins-free queries is required only for the negative side. This assumption is standard [BDG07, BB13, Kim12], as it allows to assign different atoms with different relations independently. The hardness result described here builds on that of Bagan et al. [BDG07] for databases that are assumed not to have FDs, and it relies on the computational hypothesis MAT-MUL, which amounts to solving the enumeration problem $\textsc{Enum}_\emptyset\langle\Pi\rangle$ for the query $\Pi(x, y) \leftarrow A(x, z), B(z, y)$, see Section 2.2.1.

The original proof describes an exact reduction $\textsc{Enum}_\emptyset\langle\Pi\rangle \leq_e \textsc{Enum}_\emptyset\langle Q\rangle$. Since Q is acyclic but not free-connex, it contains a free-path $(x, z_1, \dots, z_k, y)$. For a given an instance of the matrix multiplication problem, an instance of $\textsc{Enum}_\emptyset\langle Q\rangle$ is constructed, where the variables x,y and $z_1, \dots, z_k$ of the free-path encode the variables x, y and z of Π, respectively. All other variables of Q are assigned constants. This way, A is encoded by an atom containing x and z_1, and B is encoded by an atom containing z_k and y. Atoms containing some z_i and z_{i+1} propagate the value of z. Since x and y are in free(Q), but z_i are not, the answers to Q correspond to those of Π. As no atom of Q contains both x and y, the instance can be constructed in linear time. Constant delay enumeration for Q following a linear time preprocessing would result in the computation of the answers of Π in $\mathcal{O}(n^2)$ time.

FDs restrict the relations that can be assigned to atoms. This means that the reduction cannot be freely performed on databases with FDs, and the proof no longer holds. The following example illustrates where the reduction fails in the presence of FDs.

Example 4.10. The CQ from Example 1.2 has the form $Q(x, y) \leftarrow R_1(z, x), R_2(z, y)$ with the single FD $\Delta = \{R_2 \colon z \to y\}$. In the previous section, we show that it is in $\mathsf{DelayC_{lin}}$, so the reduction should fail. Indeed, it would assign R_2 with the same relation as B of the matrix multiplication problem, but this may have two tuples with the same z value and different y values. Therefore, the construction does not yield a valid instance of $\textsc{Enum}_\Delta\langle Q\rangle$. □

We now provide a modification of this construction to show an exact reduction from $\textsc{Enum}_{\emptyset}\langle\Pi\rangle$ to $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$. Any violations of the FDs are fixed by carefully picking more variables other than those of the free-path to take the roles of x,y and z of the matrix multiplication problem. This is done by introducing the sets V_x,V_y and V_z which are subsets of $var(Q)$. We say that a variable β *plays the role* of α, if $\beta \in V_\alpha$. To clarify the reduction, we start by describing a restricted case, where all FDs are unary. The basic idea in the case of general FDs remains the same, but it requires a more involved construction of the sets V_α.

4.2.1 Unary Functional Dependencies

For the unary case, we define the sets V_x, V_y and V_{z_i} to hold the variables that iteratively imply x, y and some z_i, respectively. That is, for $\alpha \in \{x, y, z_1, \ldots, z_k\}$ we set $V_\alpha := \{\alpha\}$ and apply $V_\alpha := V_\alpha \cup \{\gamma \in var(Q) \mid \gamma \to \beta \in \Delta_{Q^+} \wedge \beta \in V_\alpha\}$ until a fix-point is reached. We then define $V_z := V_{z_1} \cup \cdots \cup V_{z_k}$.

The Reduction. Let $\mathcal{D} = (A^{\mathcal{D}}, B^{\mathcal{D}})$ be an instance of $\textsc{Enum}_{\emptyset}\langle\Pi\rangle$. We define $\sigma(\mathcal{D})$ by describing the relation $R^{\sigma(\mathcal{D})}$ for every atom $R(\vec{v}) \in \text{atoms}(Q^+)$. If $var(R) \cap V_y = \emptyset$, then every tuple $(a, c) \in A^{\mathcal{D}}$ is copied to a tuple in $R^{\sigma(\mathcal{D})}$. Variables in V_x get the value a, variables in V_z get the value c, and variables that play no role are assigned a constant $\perp$. That is, we define $R^{\sigma(\mathcal{D})}$ to be $\{(f(v_1, a, c), \ldots, f(v_k, a, c)) \mid (a, c) \in A^{\mathcal{D}}\}$, where:

$$f(v_i, a, c) = \begin{cases} a & \text{if } v_i \in V_x \setminus V_z, \\ c & \text{if } v_i \in V_z \setminus V_x, \\ (a, c) & \text{if } v_i \in V_x \cap V_z, \\ \perp & \text{otherwise.} \end{cases}$$

If $var(R) \cap V_y \neq \emptyset$, we show that $var(R) \cap V_x = \emptyset$, see the sketch for the proof of the well-definedness below. In this case we define the relation similarly with $B^{\mathcal{D}}$. Given a tuple $(c, b) \in B^{\mathcal{D}}$, the variables of V_y get the value b, and those of V_z are assigned with c.

Example 4.11. Consider the query $Q^+(x, y, v) \leftarrow R(u, x, z), S(v, y, z)$ with FDs $\Delta_{Q^+} = \{R\colon u \to x, R\colon u \to z, S\colon y \to v\}$. Using the free-path (x, z, y), the reduction sets $V_x = \{x, u\}$, $V_y = \{y\}$ and $V_z = \{z, u\}$. Given an instance $\mathcal{D}$ of the matrix multiplication problem with relations $A^{\mathcal{D}}$ and $B^{\mathcal{D}}$, every tuple $(a, c) \in A^{\mathcal{D}}$ results in a tuple $((a, c), a, c) \in R^{\sigma(\mathcal{D})}$, and every tuple $(c, b) \in B^{\mathcal{D}}$ results in a tuple $(\perp, b, c) \in S^{\mathcal{D}}$. $\square$

We now outline the correctness of this reduction.

Well-defined reduction: For an atom R, either we have $var(R) \cap V_y = \emptyset$ or $var(R) \cap V_x = \emptyset$. That is, no atom contains variables from both V_x and V_y. Due to the definition of Q^+, this atom would otherwise also contain both x and y. However, they cannot appear in the same relation according to the definition of a free-path. The reduction is therefore well defined, and it can be constructed in linear time via copy and projection.

Preserving FDs: The construction ensures that if an FD $\gamma \to \alpha$ exists, then γ has all the roles of α. Therefore, either α has no role and corresponds to the constant $\perp$, or every value that appears in α also appears in γ. In any case, all FDs are preserved.

1-1 mapping of answers: If a variable of V_z had appeared in the head of Q^+, then by the definition of Q^+, some z_i would have been in the head as well. This cannot happen according to the definition of a free-path. Therefore, $V_z \cap \mathrm{free}(Q^+) = \emptyset$, and the head only encodes the x and y values of the matrix multiplication problem, so two different solutions to $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$ must differ in either x or y, and correspond to different solutions of $\textsc{Enum}_{\emptyset}\langle\Pi\rangle$. For the other direction, the head necessarily contains the variables x and y. Therefore, two different solutions to $\textsc{Enum}_{\emptyset}\langle\Pi\rangle$ correspond to different solutions of $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$.

4.2.2 General Functional Dependencies

Next we show how to lift the idea of this reduction to the case of general FDs. In the case of unary FDs, we ensure that the construction does not violate a given FD $\gamma \to \alpha$, by simply encoding the values of α to γ. In the general case, when allowing more than one variable on the left-hand side of an FD $\gamma_1, \dots, \gamma_k \to \alpha$, we must be careful when choosing the variables γ_j to which we copy the values of α. Otherwise, as the following example shows, we will not be able to construct the instance in linear time.

Example 4.12. Consider the query $Q(x, y) \leftarrow R_1(x, z, t_1), R_2(z, y, t_1, t_2)$ over a schema with the FD $R_2\colon t_1t_2 \to y$. Note that $Q = Q^+$ is acyclic but not free-connex, and that (x, z, y) is a free-path in $\mathcal{H}(Q^+)$. To repeat the idea shown in the unary case and ensure that the FDs still hold, the variable on the right-hand side of every FD is encoded to the variables on the left-hand side. If we encode y to t_1, then R_1 would contain the encodings of x, y and z. This means that its size will not be linear in that of the matrix multiplication instance, and we cannot hope for linear time construction. On the other hand, if we choose to encode y only to t_2, the reduction works. $\square$

In the following central lemma, we describe how to carefully pick the variables to which we assign roles in a way that meets the requirements we need for the reduction. We prove requirements 1 and 2 to guarantee a one-to-one mapping between the results of the two problems. Requirement 3 enables linear time construction, while requirement 4 is used to show that all FDs are preserved. The idea is that we consider the join-tree of Q^+ and define a partition of its atoms. We then define V_x and V_y to hold variables that appear only in different parts of the tree, ensuring that no atom contains variables of each. The running intersection property of a join-tree is then used to guarantee that the sets are inclusive enough to correct all FD violations.

Lemma 4.13. *Let Q be a self-join-free CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, such that Q^+ is acyclic but not free-connex. Further let $(x, z_1, \dots, z_k, y)$ be a free-path of Q^+. Then, there exist sets of variables V_x, V_y and V_z such that:*

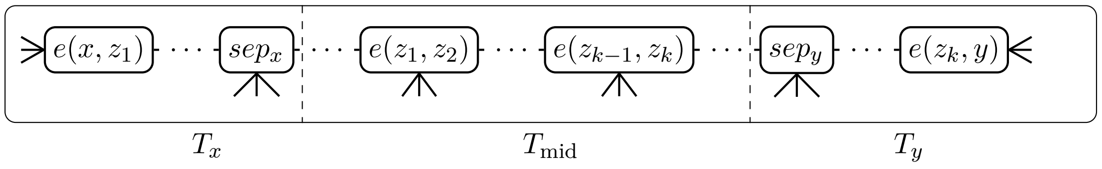


Figure 4.1: Join tree T of $\mathcal{H}(Q^+)$ for free-paths of length greater than 2. The subtrees T_x, T_y and T_{mid} are disjoint, and are separated by the nodes sep_x and sep_y.

1. $x \in V_x$*,* $y \in V_y$*,* $\{z_1, \dots z_k\} \subseteq V_z$*.*

2. $V_z \cap \text{free}(Q^+) = \emptyset$*.*

3. For every $R \in \text{atoms}(Q^+)$*:* $var(R) \cap V_y = \emptyset$ *or* $var(R) \cap V_x = \emptyset$*.*

4. For every $U \to v \in \Delta_{Q^+}$ *s.t.* $v \in V_\alpha$ *with* $\alpha \in \{x, y, z\}$*:* $U \cap V_\alpha \neq \emptyset$*.*

Proof. We first define a partition of the atoms of Q into two or three sets: T_x, T_y and possibly T_{mid}. Let T be a join tree of $\mathcal{H}(Q^+)$, and denote the hyperedges on the free-path by $e(x, z_1), \dots, e(z_k, y)$. Note that, by definition, each hyperedge of the free-path is a vertex of T and an atom of Q^+. By the running intersection property of T and since the path is cordless, we can conclude that there is a simple path P from $e(x, z_1)$ to $e(z_k, y)$ in T, such that $e(z_1, z_2), \dots, e(z_{k-1}, z_k)$ lie on that path in the order induced by the free-path. Let sep_x be the first atom on the path P that does not contain x. This exists because $e(z_k, y)$ does not contain x, as the free-path is chordless. Similarly, let sep_y be the last atom on P that does not contain y. Let T_x be the set of atoms v such that the unique path from v to $e(x, z_1)$ in T does not go through sep_x. Similarly, let T_y be the set of atoms w such that the unique path from w to $e(z_k, y)$ in T does not go through sep_y. Next set $T_{\text{mid}} = V(T) \setminus (T_x \cup T_y)$. Note that $e(x, z_1) \in T_x$ and $e(z_k, y) \in T_y$, but T_{mid} may be empty if the free-path is of length two. By definition, the atoms of Q^+ are exactly $T_x \cup T_{\text{mid}} \cup T_y$, and next we show that this union is disjoint. Figure 4.1 depicts the established partition.

Claim. *The sets* T_x *and* T_y *are disjoint.*

Proof of the Claim. Assume by contradiction that there is some $v \in T_x \cap T_y$. Let P_x be the unique simple path from v to $e(x, z_1)$, and recall that since $v \in T_x$ it does not go through sep_x. Similarly let P_y be the unique simple path from v to $e(z_k, y)$ that does not go through sep_y.

We first claim that there exists some atoms w that appears in all three paths P, P_x and P_y. Take w to be the first atom on P_x that is also in P and set P_x^w to be the simple path from v to w. Such an atom w exists because the last atom of P_x is $e(x, z_1)$ which is in P. Further set P^w to be the simple path from w to $e(z_k, y)$. Concatenating the paths P_x^w

and P^w, we obtain a simple path from v to $e(z_k, y)$. Since the simple paths in a tree are unique, this is exactly P_y, and so w is also in P_y.

Our second claim is that if an atom u is in both P and P_x, then it contains the variable x. Assume by contradiction that such an atom u does not contain x. Then u is an atom on P not containing x, and by definition of sep_x, the simple path from u to $e(x, z_1)$ contains sep_x. As this path is a subpath of P_x, P_x contains sep_x, in contradiction to the fact that $v \in T_x$. Similarly, if an atom is in both P and P_y, then it contains y.

Combining the two claims, we have an atom w containing both x and y, in contradiction to the fact that a free-path is chordless by definition. Therefore we conclude that T_x and T_y are indeed disjoint. □

Now we are ready to define the sets of variables V_x, V_y and V_z. We define V_x recursively to contain x and variables that imply those of V_x, but without variables that appear outside of T_x. V_y is defined symmetrically. V_z contains $z_1, \dots, z_k$ and variables that imply those of V_z but without free variables. Formally, $\mathsf{Implies}(V) = \{u \in var(Q) \mid \exists U \to w \in \Delta_{Q^+}$ with $w \in V$ and $u \in U\}$ for $V \subseteq var(Q)$, and we define via fix-point iteration the following:

- $\boldsymbol{V_x}$: base $V_x := \{x\}$; rule $V_x := (V_x \cup \mathsf{Implies}(V_x)) \setminus var(T_y \cup T_{\text{mid}})$
- $\boldsymbol{V_y}$: base $V_y := \{y\}$; rule $V_y := (V_y \cup \mathsf{Implies}(V_y)) \setminus var(T_x \cup T_{\text{mid}})$
- $\boldsymbol{V_z}$: base $V_z := \{z_1, \dots z_k\}$; rule $V_z := (V_z \cup \mathsf{Implies}(V_z)) \setminus \text{free}(Q^+)$

We now prove that V_x, V_y and V_z meet the requirements of the lemma. Requirements 1 and 2 follow immediately from the definition of the sets. To prove requirement 3, let $R \in \text{atoms}(Q^+)$. If $R \in T_x$, then by definition of V_y we have that $var(R) \cap V_y = \emptyset$. Otherwise, $R \in T_y \cup T_{\text{mid}}$, and similarly $var(R) \cap V_x = \emptyset$. It is left to show requirement 4.

Let $\delta = U \to v \in \Delta_{Q^+}$ where $v \in V_\alpha$. We first show the case of $\alpha = z$. If $U \cap V_z = \emptyset$, then $U \subseteq \text{free}(Q^+)$, and by the definition of Q^+, $v \in \text{free}(Q^+)$, which is a contradiction to the definition of V_z. Now we prove the case where $\alpha = x$. The case $\alpha = y$ is symmetric. Denote by $e(U, v)$ an atom containing all variables of δ. As $v \in V_x$, we know that $e(U, v) \notin T_y \cup T_{\text{mid}}$, therefore $e(U, v) \in T_x$. Assume by contradiction that $U \cap V_x = \emptyset$. Let $u \in U$. By definition of V_x, this means that $u \in var(e_u)$ for some $e_u \in T_y \cup T_{\text{mid}}$. As T_x, T_y and T_{mid} are disjoint, we have that $e_u \notin T_x$, which means that the path between e_u and $e(x, z_1)$ goes through sep_x. This means that the path from e_u to $e(U, v)$ goes through sep_x too, otherwise the concatenation of this path with the path from $e(U, v)$ to $e(x, z_1)$ would result in a path from e_u to $e(x, z_1)$ not going through sep_x. By the running intersection property, $u \in var(sep_x)$. Since this is true for all for all $u \in U$, it follows that $v \in var(sep_x)$ by definition of Q^+, contradicting the fact that $v \in V_x$. □

With the sets V_x, V_y, V_z at hand, we can now perform the reduction in the presence of general FDs.

Lemma 4.14. *Let Q be a self-join-free CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, and Π be the CQ $\Pi(x, y) \leftarrow A(x, z), B(z, y)$ over $\mathcal{S}' = (\{A, B\}, \emptyset)$. If Q^+ is acyclic and not free-connex, then* $\text{Enum}_{\emptyset}\langle\Pi\rangle \leq_e \text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$.

Proof. Let $\mathcal{D}_{A,B} = (A^{\mathcal{D}}, B^{\mathcal{D}})$ be an instance of $\text{Enum}_{\emptyset}\langle\Pi\rangle$ over the domain $dom = \{1, \ldots, n\}$. We define an instance $\sigma(\mathcal{D}_{A,B})$ of $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$ based on the sets V_x, V_y and V_z from Lemma 4.13 and the relations $A^{\mathcal{D}}$ and $B^{\mathcal{D}}$. Since Q^+ is acyclic but not free-connex, it contains some free-path $(x, z_1, \ldots, z_k, y)$.

To define the instance $\sigma(\mathcal{D}_{A,B})$, we first fix the functions f_A and f_B:

$$f_A(v, a, c) = \begin{cases} a & : v \in V_x \setminus V_z \\ c & : v \in V_z \setminus V_x \\ (a, c) & : v \in V_x \cap V_z \\ \perp & : \text{otherwise} \end{cases}, \qquad f_B(v, b, c) = \begin{cases} b & : v \in V_y \setminus V_z \\ c & : v \in V_z \setminus V_y \\ (b, c) & : v \in V_y \cap V_z \\ \perp & : \text{otherwise} \end{cases}$$

We partition all relational atoms of Q^+ into two sets: $\mathcal{R}^+_A$ and $\mathcal{R}^+_B$. The set $\mathcal{R}^+_A$ is defined as $\{R^+ \in \text{atoms}(Q^+) \mid var(R^+) \cap V_y = \emptyset\}$ and $\mathcal{R}^+_B$ is $\text{atoms}(Q^+) \setminus \mathcal{R}^+_A$. To obtain an instance $\sigma(\mathcal{D}_{A,B})$ of $\text{Enum}_{\Delta^+}\langle Q^+\rangle$, we apply f_A to the atoms in $\mathcal{R}^+_A$ using the values of $A^{\mathcal{D}}$, while the atoms in $\mathcal{R}^+_B$ use f_B and $B^{\mathcal{D}}$. That is, if $R^+(u_1, \ldots, u_m) \in \mathcal{R}^+_A$, then $(R^+)^{\sigma(\mathcal{D}_{A,B})}$ is defined to be $\{(f_A(u_1, a, c), \ldots, f_A(u_m, a, c)) \mid (a, c) \in A^{\mathcal{D}}\}$. Otherwise, $R^+(u_1, \ldots, u_m)$ is in $\mathcal{R}^+_B$, and $(R^+)^{\sigma(\mathcal{D}_{A,B})} = \{(f_B(v_1, b, c), \ldots, f_B(v_p, b, c)) \mid (c, b) \in B^{\mathcal{D}}\}$. The mapping τ is defined as the projection onto the variables x and y. Note that the instance can be constructed in linear time, and the projection can be computed in constant time.

We now claim that $\sigma(\mathcal{D}_{A,B})$ is a database over the schema $(\mathcal{R}^+, \Delta_{Q^+})$, as all the FDs of Δ_{Q^+} are satisfied. Let $\delta = R^+_j : U \to v \in \Delta_{Q^+}$. If $v \notin V_x \cup V_y \cup V_z$, then δ holds as v is assigned the value $\perp$ in every tuple in $(R^+_j)^{\sigma(\mathcal{D}_{A,B})}$. Next assume that $v \in V_x \setminus V_z$. By point 4 of Lemma 4.13, there is some $u \in U$ such that $u \in V_x$. Thus in every tuple in $(R^+_j)^{\sigma(\mathcal{D}_{A,B})}$, if v is assigned the value a, then u is either assigned the value a or (a, c) for some $c \in \{1, \ldots, n\}$ and in either case δ is satisfied. The proof for the cases where $v \in V_z \setminus (V_x \cup V_y)$ and $v \in V_y \setminus V_z$ is similar. Next assume that $v \in V_x \cap V_z$. By point 4 of Lemma 4.13, there are some $u_1, u_2 \in U$ such that $u_1 \in V_x$ and $u_2 \in V_z$ and for every tuple in $(R^+_j)^{\sigma(\mathcal{D}_{A,B})}$, if v is assigned the value (a, c), then u_1 is either assigned the value a or (a, c), u_2 is either assigned the value c or (a, c), and so δ is satisfied. The case $v \in V_y \cap V_z$ is similar. Note that the case where $v \in V_x \cap V_y$ cannot occur due to point 3 of Lemma 4.13.

The structure of the free-path $(x, z_1 \ldots, z_k, y)$ guarantees that all answers to the enumeration problem $\text{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$ correspond to those of $\text{Enum}_{\emptyset}\langle\Pi\rangle$ and vice versa. Indeed, let $\mu|_{\{x,y\}} \in \Pi(\mathcal{D}_{A,B})$. We define $\nu_\mu : var(Q^+) \to dom$ as follows.

$$\nu_\mu(v) = \begin{cases} \mu(x) & : v \in V_x \setminus V_z, \\ \mu(y) & : v \in V_y \setminus V_z, \\ \mu(z) & : v \in V_z \setminus (V_x \cup V_y), \\ (\mu(x), \mu(z)) & : v \in V_z \cap V_x, \\ (\mu(y), \mu(z)) & : v \in V_z \cap V_y, \\ \perp & : \text{otherwise.} \end{cases}$$

By definition, $f_A(v, \mu(x), \mu(z)) = \nu_\mu(v)$ and $f_B(v, \mu(y), \mu(z)) = \nu_\mu(v)$. As $\mu|_{\{x,y\}} \in \Pi(\mathcal{D}_{A,B})$, we have that $(\mu(x), \mu(z)) \in A^{\mathcal{D}}$ and $(\mu(z), \mu(y)) \in B^{\mathcal{D}}$. Consider any atom $R^+(u_1, \dots, u_m)$ of Q^+. If $R^+(u_1, \dots, u_m) \in \mathcal{R}_A^+$, then

$$\nu_\mu(u_1, \dots, u_m) = (f_A(u_1, \mu(x), \mu(z)), \dots, f_A(u_m, \mu(x), \mu(z))) \in (R^+)^{\sigma(\mathcal{D}_{A,B})}.$$

If $R^+(u_1, \dots, u_m) \in \mathcal{R}_B^+$, we have that $var(R^+) \cap V_x = \emptyset$ by point 3 of Lemma 4.13. Then, $\nu_\mu(u_1, \dots, u_m) = (f_B(u_1, \mu(z), \mu(y)), \dots, f_B(u_m, \mu(z), \mu(y)))$ is in $(R^+)^{\sigma(\mathcal{D}_{A,B})}$. Therefore, $\nu_\mu|_{\text{free}(Q^+)} \in Q^+(\sigma(\mathcal{D}_{A,B}))$. Since $x \in V_x \setminus V_z$ and $y \in V_y \setminus V_z$, we have that $\nu_\mu(x) = \mu(x)$ and $\nu_\mu(y) = \mu(y)$, and so $\tau(\nu_\mu) = \mu|_{\{x,y\}}$. Moreover, any answer $\mu'|_{\text{free}(Q^+)} \in Q^+(\sigma(\mathcal{D}_{A,B}))$ that has $\tau(\mu'|_{\text{free}(Q^+)}) = \mu|_{\{x,y\}}$ assigns $\mu(x)$ to variables in $V_x \setminus V_z$, assigns $\mu(y)$ to variables in $V_y \setminus V_z$ and assigns $\perp$ to variables not in $V_x \cup V_y \cup V_z$. By points 2 and 3 of Lemma 4.13, $V_z \cap \text{free}(Q^+) = \emptyset$ and $V_x \cap V_y = \emptyset$, so these are the only variables in free(Q^+). Therefore $\mu'|_{\text{free}(Q^+)} = \nu_\mu|_{\text{free}(Q^+)}$.

Next assume that $\mu'|_{\text{free}(Q^+)} \in Q^+(\sigma(\mathcal{D}_{A,B}))$. Let R_x^+ be an atom containing x and z_1 (such an atom exists by the definition of the free-path). By point 3 of Lemma 4.13 we have $var(R^+) \cap V_y = \emptyset$, and $R^+ \in \mathcal{R}_A^+$. By points 1 and 2 of Lemma 4.13 and since x is a free variable, we have $x \in V_x \setminus V_z$ and $z_1 \in V_z$. Thus there exists some $(a, c) \in A^{\mathcal{D}}$ such that $\mu'(x) = f_A(x, a, c) = a$ and $\mu'(z_1) = f_A(z_1, a, c) \in \{a, (a, c)\}$. Similarly, there exists some $(c', b) \in B^{\mathcal{D}}$ such that $\mu'(y) = f_B(y, b, c') = b$ and $\mu'(z_k) = f_B(z_k, b, c') \in \{c', (b, c')\}$. It remains to show that $c = c'$. We show by induction on i that $\mu'(z_i)$ is either c or of the form (t, c) with some value t. We know this fact for $i = 1$ since this is how we define c. If $k > 1$, then consider an atom $R_i^+(\vec{v}_i)$ containing $\{z_{i-1}, z_i\}$. Then μ' maps $\vec{v}_i$ to some tuple $t \in R_i^+$ that assigns z_{i-1} with a value of the form c or (t, c). Since $z_i \in V_z$, t also assigns z_i with a value of such a form, so $\mu'(z_i)$ is of the form c or (t, c) too. This show that $c = c'$. Therefore, $\tau(\mu') \in \Pi(\mathcal{D}_{A,B})$. Moreover, since τ is simply a projection, $\tau(\mu')$ is uniquely defined. □

By combining Theorem 4.6 and Lemma 4.14, we have the exact reduction $\textsc{Enum}_\emptyset\langle\Pi\rangle \leq_e \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \leq_e \textsc{Enum}_\Delta\langle Q\rangle$. Therefore having $\textsc{Enum}_\Delta\langle Q\rangle$ in $\mathsf{DelayC_{lin}}$ would mean that $\textsc{Enum}_\emptyset\langle\Pi\rangle \in \mathsf{DelayC_{lin}}$, which contradicts the hypothesis MAT-MUL. This concludes the proof of Theorem 4.9. Note that Theorem 4.9 does not contradict the dichotomy of Theorem 3.1: if for a given query Q we have that Q^+ is acyclic but not free-connex, then Q is not free-connex by Proposition 4.4.

4.3 Cyclic CQs

In the previous section, we established a classification of FD-acyclic CQs, but we did not consider FD-cyclic queries. A known result states that, under certain assumptions, self-join-free cyclic queries are not in $\mathsf{DelayC_{lin}}$ [BB13]. In this section, we therefore explore how FD-extensions can be used to obtain some insight on the implications of this result in the presence of FDs. We show that (under the same assumptions) self-join-free FD-cyclic queries that contain only unary FDs cannot be evaluated in linear time. For schemas containing only unary FDs, this extends the dichotomy presented in the previous section to all CQs, and also proves a dichotomy for the queries that can be enumerated in linear delay. We will prove the following theorem:

Theorem 4.15. *Let Q be a self-join-free CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ only contains unary FDs. If Q is FD-cyclic, then* $\textsc{Decide}_\Delta\langle Q\rangle$ *cannot be solved in linear time, assuming that the* $\textsc{Tetra}(k)$ *problem cannot be solved in linear time for any k.*

As before, the initial hardness proof for cyclic queries no longer holds in the presence of FDs, and we present a modified reduction that satisfies the FDs. We start by describing the hypothesis used to obtain the conditional lower bounds. We define $\textsc{Tetra}(k)$ to be the hypergraph with the vertices $\{1, \dots, k\}$ and the edges $\{\{1, \dots, k\} \setminus \{i\} \mid i \in \{1, \dots, k\}\}$. Let $\mathcal{H}$ be a hypergraph. With a slight abuse of notation, we also denote by $\textsc{Tetra}(k)$ the decision problem of whether $\mathcal{H}$ contains a subhypergraph isomorphic to $\textsc{Tetra}(k)$.

$\textsc{Tetra}(3)$ is the problem of deciding whether a graph contains a triangle, and it is strongly believed not to be solvable within time linear in the size of the graph [WW10]. The generalization of this assumption is that $\textsc{Tetra}(k)$ cannot be solved in time linear in the size of the graph for any $k \geq 3$. This assumption is stronger than the one used in Section 4.2, as $\textsc{Tetra}(3)$ can be reduced to the matrix multiplication problem [WW10]. However, we do have reasons to also believe the $\textsc{Tetra}(k)$ assumption for any $k \geq 4$. The (ℓ, k)-Hyperclique Hypothesis [LWW18] states that, in a k-uniform hypergraph of n vertices, $n^{k-o(1)}$ time is required to find a set of ℓ vertices such that each of it subsets of size k forms a hyperedge. Solving $\textsc{Tetra}(k)$ in linear time would contradict the (ℓ, k)-Hyperclique Hypothesis. Indeed, an algorithm that decides whether a hypergraph contains a $\textsc{Tetra}(k)$ in linear time runs in at most n^{k-1} many steps, and thus can also detect a $(k-1)$-hyperclique in a k-uniform graph in time n^{k-1}, which is less than $n^{k-o(1)}$. Thus $\textsc{Tetra}(k)$ is equivalent to the computational hypothesis hyperclique, see Section 2.2.1. As Brault-Baron used the notion of $\textsc{Tetra}(k)$ in order to show [BB13, Theorem 11], we will use the same notation for this hypothesis on order to generalize the result in the presence of FDs.

We will show that if Q^+ is cyclic and only unary FDs are present, the problem $\textsc{Tetra}(k)$ for some k can be reduced to $\textsc{Decide}_{\Delta_{Q^+}}\langle Q^+\rangle$. In the following definition, $\mathcal{H}^b$ is a pseudo-minor isomorphic to some $\textsc{Tetra}(k)$. To perform the said reduction, we will use this pseudo-minor on a graph describing our query.

Definition 4.16. Let $\mathcal{H}$ be a cyclic hypergraph. We denote by $\mathsf{Tet}_{\mathsf{pm}}(\mathcal{H})$ the pairs of pseudo-minors $(\mathcal{H}^a, \mathcal{H}^b)$ of $\mathcal{H}$ such that:

1. $\mathcal{H}^a$ is obtained by a (possibly empty) set of vertex removal and edge removal operations on $\mathcal{H}$.
2. $\mathcal{H}^b$ is obtained by a (possibly empty) set of edge contraction and edge removal operations on $\mathcal{H}^a$.
3. $\mathcal{H}^b$ is isomorphic to $\textsc{Tetra}(k)$ for some $k \geq 3$.
4. Either $\mathcal{H}^a = \mathcal{H}^b$ or $\mathcal{H}^a$ is a chordless cycle.

Given a query Q, we define $\mathsf{Tet}_{\mathsf{pm}}(Q) = \mathsf{Tet}_{\mathsf{pm}}(\mathcal{H}(Q))$.

Brault-Baron [BB13, Theorem 11] showed that a cyclic hypergraph $\mathcal{H}$ admits some $\textsc{Tetra}(k)$ as a pseudo-minor. We describe the proof here in our terminology.

Lemma 4.17 ([BB13], Theorem 11)**.** *Let $\mathcal{H}$ be a hypergraph. If $\mathcal{H}$ is cyclic, then $\mathsf{Tet}_{\mathsf{pm}}(\mathcal{H})$ is non-empty.*

Proof. If $\mathcal{H}$ has a chordless cycle C as an induced subgraph, then removing vertices not in C followed by performing all possible edge removals results in a chordless cycle $\mathcal{H}^a$. Then, $\mathcal{H}^b$ isomorphic to $\textsc{Tetra}(3)$ is obtained by a repeated use of edge-contraction followed by performing all possible edge removals. In this case, $(\mathcal{H}^a, \mathcal{H}^b) \in \mathsf{Tet}_{\mathsf{pm}}(\mathcal{H})$. If $\mathcal{H}$ does not contain a chordless cycle, since it is not acyclic, it is non-conformal. Consider its smallest non-conformal clique. The clique is not contained in any edge (since it is non-conformal), and it is a $\textsc{Tetra}(k)$ because of its minimality. Therefore, removing all vertices other than the clique, and then performing all possible edge removals, results in a graph $\mathcal{H}^a = \mathcal{H}^b$ isomorphic to some $\textsc{Tetra}(k)$. Again, $(\mathcal{H}^a, \mathcal{H}^b) \in \mathsf{Tet}_{\mathsf{pm}}(\mathcal{H})$. $\square$

For the reduction we present next, we first need to show that for an FD-cyclic query Q, no pseudo-minor in $\mathsf{Tet}_{\mathsf{pm}}(Q^+)$ contains all variables of any FD $X \to y$.

In the following we assume that Δ only contains non-trivial FDs, meaning $y \notin X$.

Lemma 4.18. *Let Q be a self-join-free FD-cyclic CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$. Let $(\mathcal{H}^a, \mathcal{H}^b) \in \mathsf{Tet}_{\mathsf{pm}}(Q^+)$ and $\mathcal{H}^a = (V, E)$. For every non-trivial $X \to y \in \Delta_{Q^+}$, we have $X \cup \{y\} \nsubseteq V$.*

Proof. We start with an observation regarding the FDs. Note that in $\mathcal{H}(Q^+)$ some edge contains the vertices $X \cup \{y\}$, and by the construction of Q^+, every edge that contains X must also contain y. These properties still hold after any sequence of vertex removals and edge removals as long as none of the vertices $X \cup \{y\}$ are removed. Therefore if none of the vertices $X \cup \{y\}$ were removed, there must be an edge in $\mathcal{H}^a$ containing $X \cup \{y\}$, and every edge in $\mathcal{H}^a$ containing X also contains y.

We distinguish two cases. In the first case, $\mathcal{H}^b = \mathcal{H}^a$ is a $\textsc{Tetra}(k)$ obtained from $\mathcal{H}(Q^+)$ by a sequence of vertex and edge removals. If $X \cup \{y\} \subseteq V$, then by the definition of $\textsc{Tetra}(k)$ it should contain the edge $V \setminus \{y\}$. Such an edge cannot exist since it contains all of X but not y. Therefore, such a $\textsc{Tetra}(k)$ cannot contain all of $X \cup \{y\}$, and in this case we conclude that $X \cup \{y\} \not\subseteq V$. In the second case, $\mathcal{H}^a$ is a cycle, and $\mathcal{H}^b$ is a $\textsc{Tetra}(3)$ obtained by performing edge contraction steps on it. If none of $X \cup \{y\}$ were removed, some edge of $\mathcal{H}^a$ must contain all of them. Since all edges are of size 2 it must be that $|X| = 1$. Denote $X = \{x\}$. Since we consider a cycle containing both x and y, there should be at least one edge containing x but not containing y. Since we showed it is not possible, such a cycle cannot contain all of $X \cup \{y\}$. □

We are now ready to establish the reduction. Given a pseudo-minor of $\mathsf{Tet}_{\mathsf{pm}}(Q^+)$ isomorphic to some $\textsc{Tetra}(k)$, we can reduce the problem of checking whether a hypergraph contains a subhypergraph isomorphic to $\textsc{Tetra}(k)$ to finding a boolean answer to Q^+.

Lemma 4.19. *Let Q be a self-join-free FD-cyclic CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ only contains unary FDs. Let $(\mathcal{H}^a, \mathcal{H}^b) \in \mathsf{Tet}_{\mathsf{pm}}(Q^+)$ such that $\mathcal{H}^b$ is isomorphic to* $\textsc{Tetra}(k)$ *for some k. Then, there is a linear time reduction* $\textsc{Tetra}(k) \leq_m \textsc{Decide}_{\Delta_{Q^+}}\langle Q^+\rangle$.

Proof. Given an input hypergraph $\mathcal{G}$ for the $\textsc{Tetra}(k)$ problem, we define an instance I of $\textsc{Decide}_{\Delta_{Q^+}}\langle Q^+\rangle$. We consider a sequence of pseudo-minors $\mathcal{H}(Q^+) = \mathcal{H}_1, \mathcal{H}_2, \dots, \mathcal{H}_m = \mathcal{H}^b$, each pseudo-minor is obtained by performing one operation over the previous one, where $\mathcal{H}_j = \mathcal{H}^a$ for some $1 \leq j \leq m$. We treat hypergraphs as describing CQs. That is, to the hypergraph $\mathcal{H}_i$ we associate a query Q_i such that $\mathcal{H}(Q_i) = \mathcal{H}_i$. Every edge e of $\mathcal{H}$ corresponds to an atom in Q_i with a relational symbol R^i_e, and the vertices of e are its variables. We assume that the variables in every atom are sorted by some total order. In the following, we construct instances I_i to these queries. It is possible to define a instance I_m such that deciding $Q_m(I_m)$ solves $\textsc{Tetra}(k)$ [BB13, Lemma 20]. We describe how to inductively build an instance $I_1 = I$ such that deciding $Q^+(I)$ solves the same problem.

Constructing I. We first define I_m. For every edge e of $\mathcal{H}_t$, we define a relation R^m_e that contains all edges of $\mathcal{G}$ that have the same size as e. A tuple of R^m_e consists of the vertices of such an edge sorted by some total order on the vertices of $\mathcal{G}$. We now define I_i given I_{i+1}. We distinguish three cases according to the type of pseudo-minor operation that leads from $\mathcal{H}_i$ to $\mathcal{H}_{i+1}$.

- *edge removal*: For every $e'' \in \mathcal{H}_{i+1}$, set $R^i_{e''} = R^{i+1}_{e''}$. Then, let e be the edge removed, and let e' be an edge containing it. Set R^i_e to be a copy of $R^{i+1}_{e'}$ projected accordingly.

- *edge contraction*: Let v be the vertex replaced by its neighbor u. For any edge $e \in \mathcal{H}_i$ contracting to an edge $e' \in \mathcal{H}_{i+1}$, set R^i_e to be a copy of $R^{i+1}_{e'}$, and assign the attribute v a copy of the value of u in every tuple. Then, if $u \notin e$, project u out of R^i_e. For every other edge $e'' \in \mathcal{H}_i$, set $R^i_{e''} = R^{i+1}_{e''}$.
- *vertex removal*: Let v be the vertex removed, and let $e \in \mathcal{H}_i$ be an edge containing v resulting in an edge $e' \in \mathcal{H}_{i+1}$. Expand $R^{i+1}_{e'}$ to R^i_e by copying $R^{i+1}_{e'}$, and assign v with a constant $\perp$ in every tuple. Next apply the following FD-correction steps on v:

  1. In every tuple, concatenate to the value of v the values of variables it implies. These variables are defined recursively by ImpliedBy$(v) = \{v\}$ and ImpliedBy$(v) = \{w \mid t \to w \in \Delta_{Q^+}, t \in \text{ImpliedBy}(v)\}$. For each $w \in \text{ImpliedBy}(v) \setminus \{v\}$, if $R^i(\vec{u})$ is an atom such that $\vec{u}[k] = v$ and $\vec{u}[j] = w$, then in every tuple $t \in R^i$, replace $t[k]$ with $(t[k], t[j])$.
  2. After the value of v is determined, concatenate the value of v to the variables implying it. These variable are defined by Implies$(v) = \{v\}$ and Implies$(v) = \{u \mid u \to t \in \Delta_{Q^+}, t \in \text{Implies}(v)\}$. For each variable $u \in \text{Implies}(v) \setminus \{v\}$, if $R^i(\vec{u})$ is an atom such that $\vec{u}[k] = v$ and $\vec{u}[j] = u$, then in every tuple $t \in R^i$, replace $t[j]$ with $(t[j], t[k])$.

  For every edge $e'' \in \mathcal{H}_i$ not containing v, set $R^i_{e''} = R^{i+1}_{e''}$.

The overall construction of the instance I can be done in linear time, since there is a constant number of pseudo-minor operations, each requiring a linear number of computational steps.

***I* is an instance of $\mathcal{S}$.** We show that I satisfies the FDs in Δ_{Q^+} by induction: we claim that for each $\mathcal{H}_i$ all FDs $x \to y$ such that $x, y \in V(\mathcal{H}_i)$ are satisfied. According to Lemma 4.18, $\mathcal{H}^a$ and therefore all of $\mathcal{H}_j, \dots, \mathcal{H}_m$ do not contain all variables of any FD. Therefore our claim trivially holds for $\mathcal{H}_j, \dots, \mathcal{H}_m$. We now prove our claim for $\mathcal{H}_i$ where $i \le j-1$. Consider an FD $\delta = x \to y$ such that $x, y \in V(\mathcal{H}_i)$. There are three cases:

- If $x, y \in V(\mathcal{H}_{i+1})$, then by the induction assumption δ is satisfied in $\mathcal{H}_{i+1}$. If $\mathcal{H}_{i+1}$ is obtained by edge removal, then the only new relation in $\mathcal{H}_i$ is a projection of a relation of $\mathcal{H}_{i+1}$, and therefore δ is satisfied in all relations. Otherwise, $\mathcal{H}_{i+1}$ is obtained by vertex removal. If the value of y is the same in R^i_e as in R^{i+1}_e, we are done by the induction assumption. Otherwise, y is changed due to the second FD-correction step, and the vertex removed is some z such that $y \to z$. In this case, since x transitively implies z, both x and y are concatenated with the same values, and δ is still satisfied.
- If $x \notin V(\mathcal{H}_{i+1})$, then $\mathcal{H}_{i+1}$ is obtained by the removal of the vertex x, and the first FD-correction step ensures that x contains a copy of the values of y in every tuple where they both appear. Therefore δ is satisfied.

- If $y \notin V(\mathcal{H}_{i+1})$, then $\mathcal{H}_{i+1}$ is obtained by the removal of the vertex y. The second FD-correction step ensures that x contains a copy of the values of y, and δ is satisfied.

Correctness. We know [BB13, Lemma 20] that there is a solution to $Q_m(I_m)$ iff there exists a subhypergraph of $\mathcal{G}$ isomorphic to $\mathcal{H}_t$, and in fact every mapping μ that can be used for the evaluation corresponds to such a subhypergraph. We claim that every mapping used for evaluating $Q_{i+1}(I_{i+1})$ corresponds to a mapping that can be used for $Q_i(I_i)$, and vice versa. This was already shown in case $\mathcal{H}_{i+1}$ is obtained by $\mathcal{H}_i$ via edge contraction [BB13, Lemma 15] or edge removal [BB13, Lemma 14], and it was shown for vertex removal [BB13, Lemma 13] if we simply assign the new vertex with a constant and skip the FD-correction steps. Let $\mathcal{H}_{i+1}$ be a pseudo-minor obtained from $\mathcal{H}_i$ via vertex removal, and denote by I_i^0 the instance constructed from I_{i+1} as described but without the FD-correction steps. It is left to show that a mapping μ^0 that satisfies $Q_i(I_i^0)$ corresponds to a mapping μ that satisfies $Q_i(I_i)$, and vice versa. This will conclude that $\mathcal{G}$ has a subhypergraph isomorphic to $\mathcal{H}_t$ iff $Q^+(I) \neq \emptyset$.

First we claim that whenever we perform an FD-correction step, if x implies y and x is in some atom, then y is in this atom too. This will help us show that we perform the same changes over x in all relations. Since Q^+ is an FD-extension and only unary FDs are present, we are guaranteed that if x implies y, then y is present in every edge of $\mathcal{H}(Q^+)$ where x appears. This property is preserved under vertex removal and edge removal operations (as long as x and y are not removed), which are the only operations that can be performed between $\mathcal{H}(Q^+)$ and any pseudo-minor on which we perform vertex-removal.

We now show that given μ^0 that satisfies $Q_i(I_i^0)$ there is a mapping μ that satisfies $Q_i(I_i)$, and vice versa. We show this by induction, considering one FD-correction step involving one variable at a time. Consider the first FD-correction step on a vertex v implying w. In any atom $R(\vec{u})$ such that $\vec{u}[k] = v$, we showed that there exists an index j such that $\vec{u}[j] = w$. For every tuple $t_0 \in R^0$, there is a similar tuple $t \in R$ with the only difference being $t[k] = (t_0[k], t_0[j])$. Therefore, by defining $\mu(v) = (\mu_0(v), \mu_0(w))$ and $\mu(u) = \mu_0(u)$ for all other variables $u \neq v$, every tuple that is used in the evaluation of μ^0 in I_i^0 results in a tuple that can be used in the evaluation of μ in I_i. Indeed, μ is a valid evaluation of $Q_i(I_i)$. A similar argument holds similarly for the opposite direction and for the second FD-correction step. For the opposite direction for example, if $\mu(v) = (a_v, a_w)$, we define $\mu^0(v) = a_v$. □

Theorem 4.15 is an immediate consequence of Lemma 4.19:

Proof of Theorem 4.15. For the sake of a contradiction assume that Q is FD-cyclic, and $\textsc{Decide}_\Delta\langle Q\rangle$ is solvable in linear time. Theorem 4.6 implies a linear time reduction $\textsc{Decide}_{\Delta_{Q^+}}\langle Q^+\rangle \leq_m \textsc{Decide}_\Delta\langle Q\rangle$. Therefore, it is possible to solve $\textsc{Decide}_{\Delta_{Q^+}}\langle Q^+\rangle$ in linear time as well. As Q^+ is cyclic, there exists a pseudo-minor $\mathcal{H}_{pm} \in \mathsf{Tet}_{\mathsf{pm}}(Q^+)$

isomorphic to $\textsc{Tetra}(k)$ for some $k \geq 3$. According to Lemma 4.19, this $\textsc{Tetra}(k)$ problem is also solvable in linear time. $\square$

In terms of enumeration complexity, Theorem 4.15 means that any enumeration algorithm for such a query cannot output a first solution (or decide that there is none) within linear time, and we get the following corollary.

Corollary 4.20. *Let Q be a self-join-free CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ only contains unary FDs. If Q is FD-cyclic, then $\textsc{Enum}_\Delta\langle Q\rangle \notin \mathsf{DelayC_{lin}}$, assuming that the $\textsc{Tetra}(k)$ problem is not solvable in linear time for any k.*

Less restrictive than constant delay enumeration, the class $\mathsf{DelayLin}$ consists of enumeration problems that can be solved with a linear delay between solutions. Acyclic CQs are known to be in $\mathsf{DelayLin}$ [BDG07], and Corollary 4.7 shows that FD-acyclic CQs are in this class as well. Theorem 4.15 implies a lower bound for $\mathsf{DelayLin}$ as well. Thus, we obtain a dichotomy stating that CQs are in $\mathsf{DelayLin}$ if and only if they are FD-acyclic.

Theorem 4.21. *Let Q be a CQ with no self-joins over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ only contains unary FDs.*

- *If Q is FD-acyclic, then $\textsc{Enum}_\Delta\langle Q\rangle \in \mathsf{DelayLin}$.*
- *Otherwise (if Q is FD-cyclic), $\textsc{Enum}_\Delta\langle Q\rangle \notin \mathsf{DelayLin}$, assuming that the $\textsc{Tetra}(k)$ problem cannot be solved in linear time for any k.*

Proof. If Q^+ is acyclic, then $\textsc{Enum}_\emptyset\langle Q^+\rangle \in \mathsf{DelayLin}$ [BDG07]. According to Theorem 4.6, $\textsc{Enum}_\Delta\langle Q\rangle \leq_e \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$. Since every instance that satisfies Δ_{Q^+} also satisfies $\emptyset$, we conclude that $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \leq_e \textsc{Enum}_\emptyset\langle Q^+\rangle$ using the identity mapping. Therefore, $\textsc{Enum}_\Delta\langle Q\rangle \in \mathsf{DelayLin}$, since $\mathsf{DelayLin}$ is closed under exact reductions.

If Q^+ is cyclic, assume by contradiction that $\textsc{Enum}_\Delta\langle Q\rangle \in \mathsf{DelayLin}$. According to Corollary 4.20, $\textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle \in \mathsf{DelayLin}$ as well. This means that finding a first answer to Q^+ or deciding that there is none can be done in linear time, in contradiction to Theorem 4.15. $\square$

We conclude this section with a short discussion about its extension to general FDs. The following example shows that the proof for Theorem 4.15 that was provided here cannot be lifted to general FDs. Exploring this extension is left for future work.

Example 4.22. Consider $Q() \leftarrow R_1(x, y, u), R_2(x, w, z), R_3(y, v, z), R_4(u, v, w)$ over a schema with all possible two-to-one FDs in R_1, R_2 and R_3. That is,

$$\begin{aligned}\Delta = \{&xy \to u, yu \to x, ux \to y, zy \to v,\\ &yv \to z, vz \to y, xz \to w, zw \to x, wx \to z\}.\end{aligned}$$

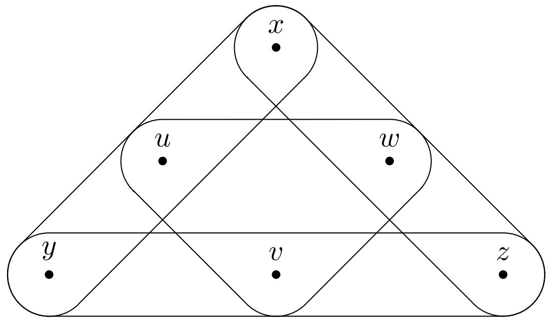


Figure 4.2: The hypergraph $\mathcal{H}(Q^+)$ for Example 4.22

Note that $Q^+ = Q$. The hypergraph $\mathcal{H}(Q^+)$ is cyclic, see Figure 4.2, yet it is unclear whether Q can be solved in linear time, and whether Tetra(3) can be reduced to answering Q^+. Using Lemma 4.18, $\mathcal{H}(Q^+)$ has triangle pseudo-minors that do not contain all variables of any FD. Consider for example the one obtained by removing all vertices other than x, y, z. A construction similar to that of Lemma 4.19 would assign u with the values of x and y, assign v with the values of y and z, and assign w with the values of x and z. This results in the edge $\{u, v, w\}$ containing all three values of any possible triangle, meaning that this edge cannot be constructed in linear time. Other choices of triangle pseudo-minors lead to similar encoding problems. $\square$

4.4 Cardinality Dependencies

In the last section of this chapter, we show that our results also apply to CQs over schemas with cardinality dependencies. *Cardinality Dependencies* (CDs) [AFG16, CFWY14] are a generalization of FDs, where the left-hand side does not uniquely determine the right-hand side, but rather provides a bound on the number of distinct values it can have. Formally, Δ is the set of *CDs* of a schema $\mathcal{S} = (\mathcal{R}, \Delta)$. Every $\delta \in \Delta$ has the form $(R_i\colon A \to B, c)$, where $R_i\colon A \to B$ is an FD and c is a positive integer. A CD δ is *satisfied* by an instance $\mathcal{D}$ over $\mathcal{S}$, if every set of tuples $S \subseteq (R_i)^{\mathcal{D}}$ that agrees on the indices of A, but no pair of them agrees on all indices of B, contains at most c tuples. It follows from the definition that δ is an FD if $c = 1$.

Denote by Δ^{FD} the FDs obtained from a set of CDs Δ by setting all c values to one. Given a query Q over $\mathcal{S} = (\mathcal{R}, \Delta)$, we define the *CD-extended query* Q^+ of Q to be the FD-extended query of Q over $\mathcal{S} = (\mathcal{R}, \Delta^{\text{FD}})$. The schema $\mathcal{S}^+$ is defined with the original c values, and the extended CDs are $\Delta_{Q^+} = \{(R_i^+\colon A \to b, c) \mid \exists (R_j\colon A \to B, c) \in \Delta, b \in B, A \cup \{b\} \subseteq \mathit{var}(R_i^+)\}$. Note that FD-extensions are indeed a special case of CD-extensions.

The hardness results extend to CDs because FDs are their special case. Since every instance that preserves the FDs Δ^{FD} also preserves the CDs Δ, we can conclude that

$\textsc{Enum}_{\Delta^{\mathrm{FD}}}\langle Q\rangle \leq_e \textsc{Enum}_{\Delta}\langle Q\rangle$. When only FDs are present we can apply Theorem 4.6, and get $\textsc{Enum}_{\Delta^{\mathrm{FD}}_{Q^+}}\langle Q^+\rangle \leq_e \textsc{Enum}_{\Delta^{\mathrm{FD}}}\langle Q\rangle$. Combining the two we get the following lemma.

Lemma 4.23. *Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ is a set of CDs, and let Q^+ be its CD-extension. Then $\textsc{Enum}_{\Delta^{FD}_{Q^+}}\langle Q^+\rangle \leq_e \textsc{Enum}_{\Delta}\langle Q\rangle$.*

Defining the classes of *CD-acyclic* and *CD-free-connex* queries similarly to the case with FDs, Lemma 4.23 implies that all lower bounds presented in this chapter hold for CDs. If Q is self-join-free and CD-acyclic but not CD-free-connex and $\textsc{Enum}_{\Delta}\langle Q\rangle \in \mathsf{DelayC_{lin}}$, then by Lemma 4.23 we have that $\textsc{Enum}_{\Delta^{\mathrm{FD}}_{Q^+}}\langle Q^+\rangle \in \mathsf{DelayC_{lin}}$ as well. According to Lemma 4.14 this means that $\textsc{Enum}_{\emptyset}\langle \Pi\rangle \in \mathsf{DelayC_{lin}}$, and the matrix multiplication problem can be solved in quadratic time. So $\textsc{Enum}_{\Delta}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$, assuming the Boolean matrix multiplication assumption. Similarly, we conclude the hardness of self-join-free CD-cyclic CQs over schemas that contain only unary CDs, of the form $(A \to B, c)$ with $|A| = 1$. Combining Lemma 4.23 with Theorem 4.15, we have that such queries cannot be evaluated in linear time, assuming that the $\textsc{Tetra}(k)$ problem cannot be solved in linear time for any k.

In order to extend the positive results, we need to show that the CD-extension is at least as hard as the original query w.r.t. enumeration. We use a slight relaxation of exact reductions: For $\textsc{Enum}\langle R_1\rangle \leq_{e'} \textsc{Enum}\langle R_2\rangle$, instead of a bijection between the sets of outputs, one output of $\textsc{Enum}\langle R_1\rangle$ corresponds to at most a constant number of outputs of $\textsc{Enum}\langle R_2\rangle$.

Lemma 4.24. *Let Q be a CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ is a set of CDs, and let Q^+ be its CD-extension. Then $\textsc{Enum}_{\Delta}\langle Q\rangle \leq_{e'} \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$.*

Proof. When dealing with FDs, we assume that the right-hand side has only one variable, as we can use such FDs to describe all possible ones. With CDs this no longer holds. Nonetheless, every instance of the schema $\mathcal{S} = (\mathcal{R}, \Delta)$ satisfies $\Delta^1 = \{(R_i\colon A \to b, c) \mid (R_i\colon A \to B, c) \in \Delta, b \in B\}$, so is also an instance of $\mathcal{S}^1 = (\mathcal{R}, \Delta^1)$. Therefore, $\textsc{Enum}_{\Delta}\langle Q\rangle \leq_e \textsc{Enum}_{\Delta^1}\langle Q\rangle$ using the identity mapping. It is left to show that $\textsc{Enum}_{\Delta^1}\langle Q\rangle \leq_{e'} \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$. The proof idea is the same as in Theorem 4.6, except now, for each tuple extended from $R_i^{\mathcal{D}}$ to $R_i^{\mathcal{D}^+}$ we can have at most c new tuples. Since this process is only done a constant number of times, the construction still only requires linear time, and the rest of the proof holds. Note that now one solution of $\textsc{Enum}_{\Delta^+}\langle Q^+\rangle$ may correspond to several solutions of $\textsc{Enum}_{\Delta^1}\langle Q\rangle$, as some variables were possibly added to the head. However, as the possible values of the added head variables are bounded by CDs, the number of solutions of Q^+ that correspond to one solution of Q is bounded by a constant.

We now formally prove that $\textsc{Enum}_{\Delta^1}\langle Q\rangle \leq_{e'} \textsc{Enum}_{\Delta_{Q^+}}\langle Q^+\rangle$. Denote Q by $Q(\vec{p}) \leftarrow R_1(\vec{v}_1), \dots, R_m(\vec{v}_m)$. Given an instance $\mathcal{D}$ of $\textsc{Enum}_{\Delta}\langle Q\rangle$, we define $\sigma(\mathcal{D})$. We start by removing tuples that interfere with the extended dependencies. For every dependency $\delta = (R_j \colon X \to y, c) \in \Delta^1$ and every atom $R_k(\vec{v}_k)$ that contains $X \cup \{y\}$, we correct R_k according to δ: we only keep tuples of $R_k^{\mathcal{D}}$ that agree with some tuple of $R_j^{\mathcal{D}}$ over the values of $X \cup \{y\}$. Next, we follow the extension of the schema, and in each step we extend some R_i to R_i' according to some CD $(R_j \colon X \to y, c)$. For each tuple $t \in R_i^{\mathcal{D}}$, if there is no tuple $s \in R_j^{\mathcal{D}}$ that agrees with t over the values of X, then we remove t altogether. Otherwise, we consider all values such tuples assign y. Denote those values by $a_1, \dots, a_k$, and note that due to the CD, $k \leq c$. We copy t to $R_i^{\sigma(\mathcal{D})}$ m times, each time assigning y with a different value of $a_1, \dots, a_k$. Given an answer $\mu \in Q^+(\sigma(\mathcal{D}))$, we define $\tau(\mu)$ to be the projection of μ to free(Q). We need to show that $Q(\mathcal{D}) = \{\mu|_{\text{free}(Q)} : \mu|_{\text{free}(Q^+)} \in Q^+(\sigma(\mathcal{D}))\}$, and that an element of the left-hand side may only appear a constant amount of times on the right-hand side. First, if $\mu|_{\text{free}(Q^+)} \in Q^+(\mathcal{D}^+)$, since all tuples of $\mathcal{D}^+$ appear (perhaps projected) in $\mathcal{D}$, then $\mu|_{\text{free}(Q)} \in Q(\mathcal{D})$. We now show the opposite direction. Let $\mu|_{\text{free}(Q)} \in Q(\mathcal{D})$, and consider a sequence of queries $Q = Q_0, Q_1, \dots, Q_n = Q^+$ such that each one is the result of extending an atom or the head of the previous query according to a CD $(R_j \colon X \to y, c)$. We claim that if $\mu|_{\vec{p}_i} \in Q_i(\vec{p}_i)$, then $\mu|_{\vec{p}_{i+1}} \in Q_{i+1}(\vec{p}_{i+1})$. This claim is trivial in case the head was extended. Note that there can be at most $c - 1$ different answers $\mu'|_{\vec{p}_{i+1}}$ to Q_{i+1} such that $\mu|_{\vec{p}_{i+1}} \neq \mu'|_{\vec{p}_{i+1}}$ but $\mu|_{\vec{p}_i} = \mu'|_{\vec{p}_i}$, as the added variable y is bound by the CD to have at most c possible values. In the case an atom $R_k(\vec{v}_k)$ was extended, denote $t_k = \mu(\vec{v}_k)$ and $t_j = \mu(\vec{v}_j)$. The construction guarantees that t_j and some copy of t_k in $\sigma(\mathcal{D})$ agree on the value of y, so $\mu(u_k) \in R_k$. By induction we get that $\mu|_{\text{free}(Q^+)} \in Q^+(\sigma(\mathcal{D}))$. $\square$

We can now extend our positive results to accommodate CDs. Let Q be a CD-free-connex CQ over a schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ contains CDs. According to Lemma 4.24, $\text{Enum}_{\Delta}\langle Q \rangle \leq_{e'} \text{Enum}_{\Delta_{Q^+}}\langle Q^+ \rangle \leq_e \text{Enum}_{\emptyset}\langle Q^+ \rangle$, and due to Theorem 3.1, $\text{Enum}_{\emptyset}\langle Q^+ \rangle \in \mathsf{DelayC_{lin}}$. The class $\mathsf{DelayC_{lin}}$ is closed under this type of reduction. To avoid printing duplicates, we need to store previous results in a lookup table, and verify that a generated result is new before printing it. This alone is not enough, as we can have a long sequence of generating known results, and then the delay between generating new results can be larger than constant. For this reason, we use the known technique of delaying the results, also used by Capelli et al. [CS18, Proposition 12]. If every answer to $\text{Enum}_{\Delta}\langle Q \rangle$ corresponds to at most c answers to $\text{Enum}_{\emptyset}\langle Q^+ \rangle$, we save the newly generated results in a queue, and after generating c results we pop and print a result from the queue. This guarantees that the queue is never empty when accessed, and the results are printed with constant delay. Therefore, $\text{Enum}_{\Delta}\langle Q \rangle \in \mathsf{DelayC_{lin}}$, and we deduce the following:

Theorem 4.25. *Let Q be a CD-acyclic CQ with no self-joins over the schema $\mathcal{S} = (\mathcal{R}, \Delta)$, where Δ is a set of CDs.*

- *If Q is CD-free-connex, then* $\text{Enum}_{\Delta}\langle Q \rangle \in \mathsf{DelayC_{lin}}$.
- *If Q is not CD-free-connex, then* $\text{Enum}_{\Delta}\langle Q \rangle \notin \mathsf{DelayC_{lin}}$, *assuming* MAT-MUL.

CHAPTER 5

Unions of Conjunctive Queries

In this chapter, we investigate the enumeration complexity of UCQs, and aim to understand which queries allow for an enumeration within $\mathsf{DelayC_{lin}}$.
We first establish a notion of free-connexity for UCQs, and show that such UCQs are always tractable. Towards a lower bound, we distinguish between UCQs only containing acyclic queries and UCQs containing at least one cyclic CQ. For the first case, queries containing so-called body-isomorphic queries are of special interest. We prove a dichotomy for UCQs consisting of two body-isomorphic queries in Section 5.2.3, as well as a result close to a dichotomy for several body-isomorphic queries. We then finish the chapter with an investigation of UCQs containing cyclic queries.

5.1 Upper Bounds via Union Extensions

Our first goal is to derive an upper bound for enumerating all answers. To do so, we generalize the notion of free-connexity from CQs to UCQs and show that queries with this property are in $\mathsf{DelayC_{lin}}$. Free-connexity of UCQs is defined via *union extensions* of a UCQ. This novel concept formalizes the idea that enumerating the answers to one CQ in a union can be used to enumerate the answers for another CQ in the same union. Before introducing any of these new notions, we first give a brief insight on a known result on UCQ enumeration.

5.1.1 Enumerating the Union of Tractable CQs

Although the enumeration complexity of UCQs is widely unknown, the complexity of enumerating a union of only tractable CQs is well understood [BKS18]. Using known techniques [Str10, Proposition 2.38], the answers to a union of tractable CQs can be enumerated with a constant delay after linear preprocessing.

Theorem 5.1 ([Str10]). *Let $Q = Q_1 \cup \ldots \cup Q_n$ be a UCQ for some fixed $n \geq 1$. If for all $1 \leq i \leq n$ the CQ Q_i is free-connex, then* $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.

Proof. Consider Algorithm 1 to enumerate the answers of a union of two CQs. In case of a union $Q = \bigcup_{i=1}^{\ell} Q_i$ of more CQs, we can use this algorithm recursively by treating the second query as $Q_2 \cup \cdots \cup Q_\ell$.

Algorithm 1 Enumerating the answers to a tractable union of two CQs

```
while $a \leftarrow Q_1(\mathcal{D}).next()$ do
    if $a \notin Q_2(\mathcal{D})$ then
        print $a$
    else
        print $Q_2(\mathcal{D}).next()$
    end if
end while
while $a \leftarrow Q_2(\mathcal{D}).next()$ do
    print $a$
end while
```

By the end of the run, the algorithm prints $Q_1(\mathcal{D}) \setminus Q_2(\mathcal{D})$ over all iterations of line 3, and it prints $Q_2(\mathcal{D})$ in lines 5 and 9. Line 5 is called $Q_1(\mathcal{D}) \cap Q_2(\mathcal{D})$ times, so the command $Q_2(\mathcal{D}).next()$ always succeeds there. Since free-connex CQs can be enumerated in constant delay and tested in constant time after a linear time preprocessing phase, this algorithm runs within the required time bounds. □

The technique presented in the proof of 5.1 has the advantage that it does not require more than constant memory available for writing in the enumeration phase. Theorem 5.1 is also a consequence of the following lemma, which gives us a general approach to compile several enumeration algorithms into one. The result of Lemma 5.2 can be used to derive upper bounds for UCQs even in cases not covered by Theorem 5.1, as we will see later in this section.

Lemma 5.2. *Let $R \subseteq \Sigma^* \times \Sigma^*$ and let n, m be positive integers. Further let $d, p : \mathbb{N} \to \mathbb{N}$ be mappings and $\mathcal{A}$ be an algorithm that outputs the solutions to every instance x of* $\textsc{Enum}\langle R\rangle$ *such that:*

- *the delay of $\mathcal{A}$ is bounded by $p(|x|)$ at most n times and bound by $d(|x|)$ otherwise;*
- *every result is produced at most m times.*

Then, there exists an enumeration algorithm $\mathcal{A}'$ for $\textsc{Enum}\langle R\rangle$*, that enumerates all solutions for an instance x with $np(|x|) + md(|x|)$ preprocessing time and $md(|x|)$ delay.*

Proof. Algorithm $\mathcal{A}'$ simulates $\mathcal{A}$ and maintains a lookup table and a queue that are initialized as empty. When $\mathcal{A}$ returns a result, $\mathcal{A}'$ checks the lookup table to determine whether it was found before. If it was not, the result is added to both the lookup table and the queue. Algorithm $\mathcal{A}'$ first performs $np(|x|)$ computation steps, and then after every $md(x)$ computation steps, it outputs a result from the queue. Moreover, $\mathcal{A}'$ returns its i-th result after $np(|x|) + imd(|x|)$ computation steps. At this time, $\mathcal{A}$ produced at least mi results, which form at least i unique results, so the queue is never empty when accessed. When it is done simulating $\mathcal{A}$, algorithm $\mathcal{A}'$ outputs all remaining results in the queue. This way, all results of $\mathcal{A}$ are output with no duplicates since every result enters the queue exactly once. $\square$

A direct consequence of Lemma 5.2 is that to show that a problem is in $\mathsf{DelayC_{lin}}$, it suffices to find an algorithm for this problem where the delay is usually constant, but it may be linear a constant number of times, and the number of times every result is produced is bound by a constant. Note that for this result, we use the fact that our model for constant delay enumeration can allocate constant memory between two outputs, as we have pointed out in Section 3.2.

5.1.2 Provided Variables and Union Extensions

As Example 1.3 in the introduction of this thesis shows, Theorem 5.1 does not cover all tractable UCQs. We now address the other cases. i.e. UCQs potentially containing at least one intractable CQ. We start with some definitions. We define *body-homomorphisms* between CQs to have the standard meaning of homomorphism, but without the restriction on the heads of the queries.

Definition 5.3. Let Q_1, Q_2 be CQs.

- A *body-homomorphism* from Q_2 to Q_1 is a mapping $h : var(Q_2) \to var(Q_1)$ such that for every atom $R(\vec{v})$ of Q_2, we have $R(h(\vec{v})) \in Q_1$.
- If Q_1, Q_2 are self-join free and there is a body-homomorphism h from Q_2 to Q_1 and vice versa, we say that Q_2 and Q_1 are *body-isomorphic*, and h is called a *body-isomorphism.*

In Example 1.3, we saw that the answers to one of the queries in the union can be used to compute answers to another query in the same union. We now formalize this observation, that one CQ within a union Q can make a contribution to the enumeration of the answers of another CQ in Q, by *providing* some variables.

Definition 5.4. Let Q_1, Q_2 be CQs. We say that Q_2 *provides* a set of variables $V_1 \subseteq var(Q_1)$ to Q_1 if:

1. There is a body-homomorphism h from Q_2 to Q_1.

2. There is $V_2 \subseteq \text{free}(Q_2)$ such that $h(V_2) = V_1$.
3. There is $V_2 \subseteq S \subseteq \text{free}(Q_2)$ such that Q_2 is S-connex.

Providing variables among queries within a union is our central tool for obtaining upper bounds on UCQ enumeration complexity. The following lemma offers some insight to the importance of this tool: If Q_2 provides a set of variables to Q_1, then we can produce an auxiliary relation for Q_1 containing all possible value combinations of these variables. This can be done efficiently while already producing some answers to Q_2.

Lemma 5.5. *Let Q_1, Q_2 be CQs such that Q_2 provides V_1 to Q_1. Given a database instance $\mathcal{D}$, one can compute with linear time preprocessing and constant delay a set of mappings $M \subseteq Q_2(\mathcal{D})$, which can be translated to $Q_1(\mathcal{D})|_{V_1}$ in time $\mathcal{O}(|M|)$.*

Proof. For the proof of this lemma and the proof of Theorem 5.8, we use the constant delay version of the Yannakakis algorithm [IUV17], denoted by CDY algorithm, which we briefly discussed in Section 3.2.

Let h be a body-homomorphism, and let V_2 and S be sets of variables meeting the conditions of Definition 5.4. Take an ext-S-connex tree T for $\mathcal{H}(Q_2)$, and perform the CDY algorithm on Q_2 while treating S as the free-variables. This results in a set N of mappings from the variables of S to the domain such that $N = Q_2(\mathcal{D})|_S$.

For every mapping $\mu \in N$, extend it once to obtain a mapping from all variables of Q_2 as follows. Go over all vertices of T starting from the connected part containing S and treating a neighbor of an already treated vertex at every step. Consider a step where in its beginning μ is a homomorphism from a set S_1, and we are treating an atom $R_i(\vec{v}_i, \vec{u}_i)$ where $\vec{v}_i \subseteq S_1$ and $\vec{u}_i \cap S_1 = \emptyset$. We take some tuple in R_i of the form $(\mu(\vec{v}_i), t_i)$ and extend μ to also map $\mu(\vec{u}_i) = t_i$. Such a tuple exists since the CDY algorithm has a preprocessing step that removes any tuple with no extension. This extension takes constant time, and in the end we have that $\mu|_{free(Q)} \in Q_2(\mathcal{D})$. These extensions form $M \subseteq Q_2(\mathcal{D})$. When computing M, the delay for the first element may be linear due to the preprocessing phase of the CDY algorithm, but the delay after that is constant.

We now describe how M can be translated to $Q_1(\mathcal{D})|_{V_1}$. As $M|_{V_2} = Q_2(\mathcal{D})|_{V_2}$, we simply need to use the body-homomorphism in the opposite direction. For every variable $v_1 \in V_1$, define $h^{-1}(v_1)$ to be $\{v_2 \in V_2 \mid h(v_2) = v_1\}$. Given a mapping $\mu \in Q_2(\mathcal{D})$, if $\mu(v_2)$ is the same for all $v_2 \in h^{-1}(v_1)$, denote it by $\mu(h^{-1}(v_1))$. Otherwise, $\mu(h^{-1}(v_1))$ is undefined, and in the following μ is skipped. Since h is a body-homomorphism, we have that $M|_{V_2} \circ h^{-1} = Q(\mathcal{D})|_{V_1}$. Given $\mu \in M$, we can compute $\mu|_{V_2} \circ h^{-1}$ (or determine that it is undefined) in constant time. Doing this for every $\mu \in M$, we can compute $Q_1(\mathcal{D})|_{V_1}$ in time $\mathcal{O}(|M|)$. $\square$

During the process of enumeration, a set of variables provided to a CQ can be used as an auxiliary relation, accessible by an auxiliary atom. The CQ along with its auxiliary atoms is called a *union extension.* Intuitively, a union extension of a CQ within a UCQ captures the possible interaction among queries in the union w.r.t. the enumeration problem.

Definition 5.6. Let $Q = Q_1 \cup \ldots \cup Q_n$ be a UCQ.

- A *union extension* Q_1^+ of $Q_1(\vec{v}) \leftarrow R_1(\vec{v}_1), \ldots, R_s(\vec{v}_s)$ is given by

  $$Q_1^+(\vec{v}) \leftarrow R_1(\vec{v}_1), \ldots, R_s(\vec{v}_s), P_1(\vec{u}_1), \ldots, P_k(\vec{u}_k)$$

  where $k \geq 0$, each $\vec{u}_i$ with $1 \leq i \leq k$ is provided by some $Q_j \in Q$, and $P_1, \ldots, P_k$ are fresh relational symbols. By way of recursion, the variables $\vec{u}_i$ may alternatively be provided by a union extension of some $Q_j \in Q$.

- Atoms appearing in Q_1^+ but not in Q_1 are called *virtual atoms.*

Using the concept of union extensions, we are now ready to define the notion of free-connexity for UCQs.

Definition 5.7. Let $Q = Q_1 \cup \ldots \cup Q_n$ be a UCQ.

- Q_1 is said to be *union-free-connex* with respect to Q if it has a free-connex union extension.
- Q is *free-connex* if all CQs in Q are union-free-connex.

Note that the term free-connex for UCQs is a generalization of that for CQs: If a UCQ Q contains only one CQ, then Q is free-connex if and only if the CQ it contains is free-connex. In the original dichotomy given by Theorem 3.1, it was proven that free-connex CQs are tractable. We next show that this tractability of free-connex queries also carries over to UCQs.

Theorem 5.8. *Let Q be a UCQ. If Q is free-connex, then $Q \in \mathsf{DelayC_{lin}}$.*

Proof. For each query Q_1 in the union (in an order imposed by the recursive definition of union extensions), we first instantiate its free-connex union extension Q_1^+, and then evaluate the resulting free-connex CQ using the CDY algorithm: For every virtual atom containing some variables V_1, use Lemma 5.5 to generate a subset of $Q_2(\mathcal{D})$ while obtaining a relation $Q_1(\mathcal{D})|_{V_1}$ assigned to this atom. After instantiating all virtual relations, we have an instance $\mathcal{D}^+$ for Q_1^+, and we can evaluate it as usual using the CDY algorithm. We have that $Q_1(\mathcal{D}) = Q_1^+(\mathcal{D}^+)$ since all virtual atoms in Q_1^+ are assigned relations that contain merely a projection of the results.

Overall, there is a constant number of times where the delay is linear: once per query and once per virtual atom. Similarly, every result is produced at most a constant number of times: once per query and once per virtual atom. According to Lemma 5.2 this means that $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$. □

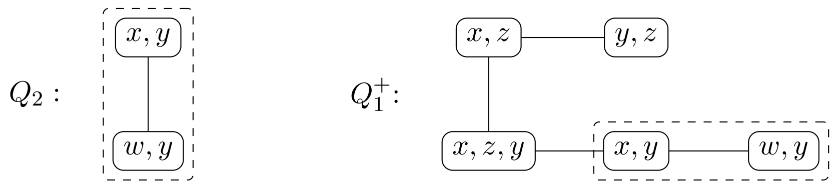


Figure 5.1: $\{x, y, w\}$-connex tree for Q_2 and $\{x, y, w\}$-connex tree for Q_1^+.

We can now revisit Example 1.3 from the introduction and explain its tractability using the terminology and results from this section. Note that we have chosen new names for the relational symbols to ensure readability.

Example 5.9. Let $Q = Q_1 \cup Q_2$ be the UCQ

$$Q_1(x, y, w) \leftarrow R_1(x, z), R_2(z, y), R_3(x, w) \text{ and}$$
$$Q_2(x, y, w) \leftarrow R_1(x, y), R_2(y, w).$$

There is a body-homomorphism h from Q_2 to Q_1 with $h((x, y, w)) = (x, z, y)$. The query Q_2 provides $\{x, z, y\}$ to Q_1, as $\{x, y, w\} \subseteq \text{free}(Q_2)$, and Q_2 is $\{x, y, w\}$-connex. Thus, we can add $R'(x, z, y)$ to Q_1, and the union extension

$$Q_1^+(x, y, w) \leftarrow R_1(x, z), R_2(z, y), R_3(y, w), R'(x, z, y)$$

is free-connex (see Figure 5.1). Since every query in Q is union-free-connex, we have that $\text{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$ by Theorem 5.8. □

The tractability result in Theorem 5.8 is based on the structure of the union extended queries. This means that the intractability of any query within a UCQ can be resolved as long as another query can provide the right variables. The following example shows that this can even be the case for a UCQ only consisting of non-free-connex CQs. Moreover, the example illustrates why we need the definition of union extensions to be recursive.

Example 5.10. Let $Q = Q_1, Q_2, Q_3$ with

$$Q_1(x, y, v, u) \leftarrow R_1(x, z_1), R_2(z_1, z_2), R_3(z_2, z_3), R_4(z_3, y), R_5(y, v, u),$$
$$Q_2(x, y, v, u) \leftarrow R_1(x, y), R_2(y, v), R_3(v, z_1), R_4(z_1, u), R_5(u, t_1, t_2),$$
$$Q_3(x, y, v, u) \leftarrow R_1(x, z_1), R_2(z_1, y), R_3(y, v), R_4(v, u), R_5(u, t_1, t_2).$$

Each of the three CQs is intractable on its own: Q_1 has the free-path (x, z_1, z_2, z_3, y), while Q_2 has the free-path (v, z_1, u), and Q_3 has the free-path (x, z_1, y). The CQ Q_2 provides the variables $\{x, z_1, y\}$ to Q_3, as it is $\{x, y, v\}$-connex, $\{x, y, v\} \subseteq \text{free}(Q)$, and there is a body-homomorphism h from Q_2 to Q_3 with $h((x, y, v)) = (x, z_1, y)$. Extending the body of Q_3 by the virtual atom $R'(x, z_1, y)$ yields the free-connex union extension Q_3^+. Similarly, we have that Q_3 provides $\{v, z_1, u\}$ to Q_2, and extending Q_2 by $R''(v, z_1, u)$ yields the free-connex union extension Q_2^+. Since Q_2^+ and Q_3^+ provide $\{x, z_1, z_2, y\}$ and respectively $\{x, z_2, z_3, y\}$ to Q_1, we obtain a free-connex union extension Q_1^+ by adding virtual atoms with the variables (x, z_1, z_2, y) and (x, z_2, z_3, y) to Q_1. Thus, Q is free-connex and can be enumerated efficiently by Theorem 5.8. □

5.2 Lower Bounds

In this section, we prove lower bounds for enumerating the answers to UCQs within the time bounds of $\mathsf{DelayC_{lin}}$. We begin with some general observations regarding cases where a single CQ is not harder than a union containing it, and then continue to handle other cases. In Section 5.2.2 we discuss unions containing only intractable CQs, and in Section 5.2.3 we discuss unions containing two body-isomorphic CQs. In both cases such UCQs may be tractable, and in case of such a union of size two, we show that our results from Section 5.1 capture all tractable unions.

5.2.1 General Observations

In order to provide some intuition for the choices we make throughout this section, we first explain where the approach used for proving the hardness of single CQs fails.

Example 5.11. Consider the UCQ $Q = Q_1 \cup Q_2$ from Example 5.9:

$$Q_1(x, y, w) \leftarrow R_1(x, z), R_2(z, y), R_3(x, w) \text{ and}$$
$$Q_2(x, y, w) \leftarrow R_1(x, y), R_2(y, w).$$

The original proof that shows that Q_1 is an intractable CQ describes a reduction from Boolean matrix multiplication [BDG07, Lemma 26]. Let A and B be binary representations of Boolean $n \times n$ matrices, i.e. $(a, b) \in A$ corresponds to a 1 in the first matrix at index (a, b). Define a database instance $\mathcal{D}$ as $R_1^{\mathcal{D}} = A$, $R_2^{\mathcal{D}} = B$, and $R_3^{\mathcal{D}} = \{1, \dots, n\} \times \{\perp\}$. One can show that $Q_1(\mathcal{D})$ corresponds to the answers of AB. If $\textsc{Enum}\langle Q_1 \rangle \in \mathsf{DelayC_{lin}}$, we can solve matrix multiplication in time $\mathcal{O}(n^2)$, in contradiction to $\textsc{mat-mul}$. Since Q_2 evaluates over the same relations as Q_1 does, Q_2 also produces answers over this construction. Since the number of results for Q_2 might reach up to n^3, enumerating Q in constant delay does not necessarily compute the answers to Q_1 in $\mathcal{O}(n^2)$ time, and does not contradict the complexity assumption. $\square$

The example above shows that in general, whenever we show a lower bound to a UCQ by computing a hard problem through answering one CQ in the union, we need to ensure that the other CQs cannot have too many answers over this construction. To resolve this issue, we describe cases where there is a way to encode any arbitrary instance of Q_1 to an instance of Q, such that no other CQ in the union returns results.

Lemma 5.12. *Let Q be a UCQ of self-join free CQs, and let $Q_1 \in Q$ such that for all $Q_i \in Q \setminus \{Q_1\}$ there is no body homomorphism from Q_i to Q_1. Then we have $\textsc{Enum}\langle Q_1 \rangle \leq_e \textsc{Enum}\langle Q \rangle$.*

Proof. Let $Q_1(\vec{p}) \leftarrow R_1(\vec{v}_1), \dots, R_m(\vec{v}_m)$. Given an instance $\mathcal{D}$ of $\textsc{Enum}\langle Q_1 \rangle$, the construction of $\sigma(\mathcal{D})$ assigns each variable of Q_1 with a different and disjoint domain by concatenating the variable names to the values in their corresponding relations. We leave the relations that do not appear in the atoms of Q_1 empty. Since there is no body-homomorphism from Q_i to Q_1, then there are now no answers to Q_i over this construction, and the answers to Q are exactly those of Q_1.

Formally, for every $i \in \{1, \dots, m\}$ and every tuple $(c_1, \dots, c_t) \in R_i^{\mathcal{D}}$ we have the tuple $((c_1, \vec{v}_i[1]), \dots, (c_t, \vec{v}_i[t]))$ in $R_i^{\sigma(\mathcal{D})}$. The relations that do not appear in Q_1 are left empty. We claim that the results of Q_1 over the original instance are exactly those of Q over our construction if we omit the variable names. That is, we define $\tau : \mathit{dom} \times \mathit{var}(Q_1) \to \mathit{dom}$ as $\tau((c, v)) = c$, and show that $Q_1(\mathcal{D}) = \tau(Q(\sigma(\mathcal{D})))$.

We first prove that $Q_1(\mathcal{D}) = \tau(Q_1(\sigma(\mathcal{D})))$. That is, we show that the results obtained due to the evaluation of Q_1 in both cases are the same. If $\mu|_{\vec{p}} \in Q_1(\mathcal{D})$, then for every atom $R_i(\vec{v}_i)$ in Q_1, $(\mu(\vec{v}_i[1]), \dots, \mu(\vec{v}_i[t])) \in R_i^{\mathcal{D}}$. By construction, $((\mu(\vec{v}_i[1]), \vec{v}_i[1]), \dots, (\mu(\vec{v}_i[t]), \vec{v}_i[t])) \in R_i^{\sigma(\mathcal{D})}$. By defining $f_\mu : \mathit{var}(Q_1) \to \mathit{dom} \times \mathit{var}(Q_1)$ as $f_\mu(u) = (\mu(u), u)$, we have $f_\mu \in Q_1(\sigma(\mathcal{D}))$. Since $\tau \circ f_\mu = \mu$, we have that $\mu|_{\vec{p}} \in \tau(Q_1(\sigma(\mathcal{D})))$, and this concludes that $Q_1(\mathcal{D}) \subseteq \tau(Q_1(\sigma(\mathcal{D})))$. The opposite direction is trivial: if $\nu|_{\vec{p}} \in Q_1(\sigma(\mathcal{D}))$, then for every atom $R_i(\vec{v}_i)$ in Q_1, $\nu(\vec{v}_i) \in R_i^{\sigma(\mathcal{D})}$. By construction, $\tau(\nu(\vec{v}_i)) \in R_i^{\mathcal{D}}$, and therefore $\tau \circ \nu|_{\vec{p}} \in Q_1(\mathcal{D})$.

We now know that $Q_1(\mathcal{D}) = \tau(Q_1(\sigma(\mathcal{D}))) \subseteq \tau(Q(\sigma(\mathcal{D})))$. It is left to show that $\tau(Q(\sigma(\mathcal{D}))) \subseteq Q_1(\mathcal{D})$. Assume by contradiction that there exists $\mu|_{\vec{p}} \in Q(\sigma(\mathcal{D}))$ such that $\tau \circ \mu|_{\vec{p}} \notin Q_1(\mathcal{D})$. Since $\mu|_{\vec{p}} \in Q(\sigma(\mathcal{D}))$, there exists some $Q_i \in Q$ such that $\mu|_{\vec{p}} \in Q_i(\sigma(\mathcal{D}))$. Since $\tau \circ \mu|_{\vec{p}} \notin Q_1(\mathcal{D})$ and $Q_1(\mathcal{D}) = \tau(Q_1(\sigma(\mathcal{D})))$, we know that $\mu|_{\vec{p}} \notin Q_1(\sigma(\mathcal{D}))$, and therefore $i \neq 1$. Define $\eta : \mathit{dom} \times \mathit{var}(Q_1) \to \mathit{var}(Q_1)$ as $\eta(c, v) = v$. Since $\mu|_{\vec{p}} \in Q_i(\sigma(\mathcal{D}))$, we know that for every atom $R_j(\vec{v}_j)$ in Q_i, $\mu(\vec{v}_j) \in R_j^{\sigma(\mathcal{D})}$. By construction, if $((c_1, v_1), \dots, (c_t, v_t)) \in R_j^{\sigma(\mathcal{D})}$ then $R_j(v_1, \dots, v_t)$ is an atom in Q_1. Consider $\mu \circ \eta : \mathit{var}(Q_i) \to \mathit{var}(Q_1)$. For every atom $R_j(\vec{v}_j)$ in Q_i, $R_j(\mu \circ \eta(\vec{v}_j))$ is an atom in Q_1. This means that there is a body-homomorphism from Q_i to Q_1, and achieves a contradiction. □

Lemma 5.12 implies that if there is an intractable CQ in a union where no other CQ maps to it via a body-homomorphism, then the entire union is intractable. This also captures cases such as a union of CQs where one of them is hard, and the others contain a relation that does not appear in the first.

Using the same reduction, a similar statement with relaxed requirements can be made in case it is sufficient to consider the Boolean evaluation problem instead of the enumeration problem.

Lemma 5.13. *Let Q be a UCQ of self-join free CQs, and let $Q_1 \in Q$ such that for all $Q_i \in Q$ either there is no body-homomorphism from Q_i to Q_1 or Q_1 and Q_i are body-isomorphic. Then,* Decide$\langle Q_1 \rangle \leq$ Decide$\langle Q \rangle$ *via a linear-time many-one reduction.*

Proof. Given a database instance $\mathcal{D}$ of Decide$\langle Q_1 \rangle$, we use the same function σ as in the proof of Lemma 5.12 to reduce this to a database instance $\sigma(\mathcal{D})$ of Decide$\langle Q \rangle$. As before, a CQ in the union with no body-homomorphism to Q_1 has no answers over

$\sigma(\mathcal{D})$. So it remains to show that for body-isomorphic CQs Q_1 and Q_2 and database $\mathcal{D}$, $Q_1(\mathcal{D}) \neq \emptyset$ if and only if $Q_2(\sigma(\mathcal{D})) \neq \emptyset$. First assume that $Q_1(\mathcal{D}) \neq \emptyset$, and let $h : var(Q_2) \to var(Q_1)$ be a body-homomorphism from Q_2 to Q_1. By construction of σ, we have that $Q_1(\mathcal{D}) \neq \emptyset$ if and only if $Q_1(\sigma(\mathcal{D})) \neq \emptyset$. For the homomorphism $\mu : var(Q_1) \to dom$ with $\mu|_{\mathsf{free}(Q)} \in Q_1(\sigma(\mathcal{D}))$, we have that $\mu \circ h|_{\mathsf{free}(Q)} \in Q_2(\sigma(\mathcal{D}))$: For every atom $R(\vec{x}) \in Q_2$, we have $R(h(\vec{x})) \in Q_1$ and thus $R(\mu \circ h(\vec{x})) \in R^{\sigma(\mathcal{D})}$. The other direction can be proven analogously. $\square$

Going back to the result by Brault-Baron [BB13] on cyclic queries, Theorem 3.3 states that deciding whether a cyclic CQ has any answers cannot be done in linear time (assuming HYPERCLIQUE). Thus according to Lemma 5.13, if a UCQ Q contains a cyclic Q_1 such that the conditions of Lemma 5.13 are satisfied, the entire union cannot be decided in linear time, and thus $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$. We will further discuss this case in Section 5.3.2.

5.2.2 Unions of Intractable CQs

We now discuss unions containing only CQs classified as *intractable* according to Theorem 3.1 and Theorem 3.3, that is, CQs that are either cyclic or acyclic and not free-connex. In addition to that, we assume intractable CQs to be self-join-free. Unions of intractable CQs are of special interest, because we have the means to derive some lower bounds according to Theorem 5.15 below. For a discussion on UCQs containing both tractable and intractable queries, see Section 5.3. The following lemma can be used to identify a CQ on which we can apply Lemma 5.12 or Lemma 5.13.

Lemma 5.14. *Let Q be a UCQ. There exists a query $Q_1 \in Q$ such that for all $Q_i \in Q$ either there is no body-homomorphism from Q_i to Q_1 or Q_1 and Q_i are body-isomorphic.*

Proof. Consider a longest sequence $(Q^1, \dots, Q^m)$ such that for every $2 \leq j \leq m$ there is a body-homomorphism from Q^j to Q^{j-1} denoted μ^j, but no body-homomorphism in the opposite direction. Note that it is not possible that the same query appears twice in the sequence: if $Q^k = Q^j$ where $j > k$, then there is a mapping $\mu_{k+2} \circ \dots \circ \mu_j$ from $Q^j = Q^k$ to Q^{k+1}, in contradiction to the definition of the sequence. Therefore, $m \leq |Q|$, and such a longest sequence exists.
We claim that $Q_1 = Q^m$ satisfies the conditions of the lemma. To see this, first consider some $Q^j \in \{Q^1, \dots, Q^{m-1}\}$. There is a body-homomorphism from Q^m to Q^j which is the concatenation of $\mu^{j+1} \circ \dots \circ \mu^m$. Therefore, either there is no body-homomorphism from Q^j to Q^m or Q^m and Q^j have isomorphic bodies. Now consider some $Q_i \notin \{Q^1, \dots, Q^m\}$. If there is no body-homomorphism from Q_i to Q^m, then we are done. Otherwise, assume that there is also no body-homomorphism from Q_i to Q^m. Then $(Q^1, \dots, Q^m, Q_i)$ is a longer sequence, contradicting the maximality. Therefore, we have that Q^m and Q_i have isomorphic bodies. $\square$

Using the results obtained so far, we deduce a characterization of all cases of a union of intractable CQs, except those that contain a pair of body-isomorphic acyclic CQs.

Theorem 5.15. *Let Q be a UCQ of intractable CQs that does not contain two body-isomorphic acyclic CQs. Then, $Q \notin \mathsf{DelayC_{lin}}$, assuming* MAT-MUL *and* HYPERCLIQUE.

Proof. Let Q_1 be a CQ in Q given by Lemma 5.14. We distinguish between two possible cases of the structure of Q_1. In case Q_1 is acyclic, since we know that Q does not contain body-isomorphic acyclic CQs, then for all $Q_i \in Q \setminus \{Q_1\}$ there is no body-homomorphism from Q_i to Q_1. According to Lemma 5.12, $\textsc{Enum}\langle Q_1\rangle \leq_e \textsc{Enum}\langle Q\rangle$. Since Q_1 is self-join free acyclic non-free-connex, we have that $\textsc{Enum}\langle Q_1\rangle \notin \mathsf{DelayC_{lin}}$ assuming MAT-MUL. Therefore $\textsc{Enum}\langle Q\rangle$ is not in $\mathsf{DelayC_{lin}}$ either. In case Q_1 is cyclic, we use Lemma 5.13 to conclude that $\textsc{Decide}\langle Q_1\rangle \leq \textsc{Decide}\langle Q\rangle$. According to Theorem 3.1, since Q_1 is self-join free cyclic, $\textsc{Decide}\langle Q_1\rangle$ cannot be solved in linear time assuming HYPERCLIQUE. Therefore $\textsc{Decide}\langle Q\rangle$ cannot be solved in linear time, and in particular $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$. □

In the next example, we demonstrate how the reductions from Lemma 5.13 and Theorem 3.1 combine in Theorem 5.15.

Example 5.16. Consider the UCQ $Q = Q_1 \cup Q_2 \cup Q_3$ with

$$\begin{aligned}
Q_1(x,y) &\leftarrow R_1(x,y), R_2(y,u), R_3(x,u),\\
Q_2(x,y) &\leftarrow R_1(y,v), R_2(v,x), R_3(y,x),\\
Q_3(x,y) &\leftarrow R_1(x,z), R_2(y,z).
\end{aligned}$$

The queries Q_1 and Q_2 are cyclic, and Q_3 is acyclic but not free-connex. This union is intractable according to Theorem 5.15. Note that Q_1 and Q_2 are body-isomorphic, but there is no body-homomorphism from Q_3 to Q_1. The proof of Theorem 3.1 states the following: If $\textsc{Enum}\langle Q_1\rangle \in \mathsf{DelayC_{lin}}$, then given an input graph G, we can use Q_1 to decide the existence of triangles in G in time $\mathcal{O}(n^2)$, in contradiction to HYPERCLIQUE. The same holds true for the $\textsc{Enum}\langle Q\rangle$. For every edge (u,v) in G with $u < v$ we add $((u,x),(v,y))$ to $R_1^{\mathcal{D}}$, $((u,y),(v,z))$ to $R_2^{\mathcal{D}}$ and $((u,x),(v,z))$ to $R_3^{\mathcal{D}}$. The query detects triangles: for every triangle a,b,c in G with $a < b < c$, the query Q_1 returns $((a,x),(b,y))$. The union only returns answers corresponding to triangles:

- For every answer $((d,x),(e,y))$ to Q_1, there exists some f such that d,e,f is a triangle in G with $d < e < f$.
- For every answer $((g,z),(h,x))$ to Q_2, there exists some i such that g,h,i is a triangle in G with $h < i < g$.
- The query Q_3 returns no answers over this construction.

□

Theorem 5.15 does not cover the case of a UCQ containing acyclic non-free-connex queries with isomorphic bodies. Since this requires a more intricate analysis, we first restrict ourselves to such unions of size two.

5.2.3 Unions of Two Body-Isomorphic CQs

In this section, we discuss unions of two body-isomorphic CQs in general, and show in Theorem 5.27 that such a UCQ is tractable if and only if the UCQ is free-connex. By combining this with Theorem 5.15, we have the following dichotomy for the case of unions containing exactly two intractable CQs.

Theorem 5.17. *Let $Q = Q_1 \cup Q_2$ be a union of intractable CQs.*

- *If Q is free-connex, then* $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.
- *If Q is not free-connex, then* $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$*, assuming* MAT-MUL*,* HYPERCLIQUE *and 4-*CLIQUE*.*

In order to show Theorem 5.27, we will start with some general observations and examples. First consider an arbitrary pair of body-isomorphic CQs. As both of them have the same structure, either the two CQ are cyclic, or both are acyclic. In the case of a union of two cyclic CQs, the UCQ is intractable according to Theorem 5.15. So in this section, we discuss the union of body-isomorphic acyclic CQs. Note that, unlike the previous section, we allow a CQ in the union to be free-connex. Dealing with only body-isomorphic CQs in a union allows us to introduce a new notation that we use hereafter.

Consider a UCQ of the form $Q_1 \cup Q_2$, where there exists a body-isomorphism h from Q_2 to Q_1. That is, the CQs have the structure:

$$\begin{aligned} &Q_1(\vec{v}_1) \leftarrow R_1(h(\vec{w}_1)), \ldots, R_n(h(\vec{w}_n)),\\ &Q_2(\vec{v}_2) \leftarrow R_1(\vec{w}_1), \ldots, R_n(\vec{w}_n). \end{aligned}$$

Applying h^{-1} to the variables of Q_1 does not affect evaluation. Thus, we can rewrite Q_1 as $Q_1(h^{-1}(\vec{v}_1)) \leftarrow R_1(\vec{w}_1), \ldots, R_n(\vec{w}_n)$. Since now the two CQs have exactly the same body, we can treat the UCQ as a query with one body and two heads:

$$Q_1(h^{-1}(\vec{v}_1)), Q_2(\vec{v}_2) \leftarrow R_1(\vec{w}_1), \ldots, R_n(\vec{w}_n)$$

This same notation can also be used when considering UCQs consisting of more than two body-isomorphic CQs. Note that when treating a UCQ as one CQ with several heads, the set atoms(Q) can be used instead of atoms(Q_i) for any Q_i in Q, as the atoms are the same for all CQs in the union. Similarly, the set free(Q_i) has a new meaning, as the free variables may differ between different queries Q_i in the union. With this notation at hand, we now inspect some examples of two body-isomorphic acyclic CQs.

Example 5.18. Consider $Q = Q_1 \cup Q_2$ with

$$Q_1(x,y,v) \leftarrow R_1(x,z), R_2(z,y), R_3(y,v), R_4(v,w) \text{ and}$$
$$Q_2(x,y,v) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x).$$

Since the CQs in Q are body-isomorphic, the query Q can be rewritten as

$$Q_1(w,y,z), Q_2(x,y,v) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x).$$

Using the new notation, we have that free$(Q_1) = \{w,y,z\}$ and free$(Q_2) = \{x,y,v\}$. Concerning the enumeration complexity of Q, we can use the same approach used for single CQs in Theorem 3.1, and show that this UCQ is not in $\mathsf{DelayC_{lin}}$ assuming MAT-MUL. Let A and B be binary representations of Boolean $n \times n$ matrices as explained in the beginning of this section. Define a database instance $\mathcal{D}$ as $R_1^{\mathcal{D}} = A$, $R_2^{\mathcal{D}} = B$, $R_3^{\mathcal{D}} = \{1,\dots,n\} \times \{\perp\}$ and $R_4^{\mathcal{D}} = \{(\perp,\perp)\}$. Since $Q_1(\mathcal{D})$ corresponds to the answers of AB, and $Q_2(\mathcal{D})$ is of size $\mathcal{O}(n^2)$, we cannot enumerate Q within the time bounds of $\mathsf{DelayC_{lin}}$ unless we can solve matrix multiplication in time $\mathcal{O}(n^2)$. $\square$

A union of two intractable body-isomorphic acyclic CQs may also be tractable. In fact, by only adding a single variable to the heads of the CQs in Example 5.18, we obtain a tractable UCQ.

Example 5.19. Let Q be the UCQ

$$Q_1(x,y,w,v) \leftarrow R_1(x,z), R_2(z,y), R_3(y,v), R_4(v,w) \text{ and}$$
$$Q_2(x,y,w,v) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x),$$

which can be rewritten as

$$Q_1(w,y,x,z), Q_2(x,y,w,v) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x).$$

Both CQs are acyclic non-free-connex. As Q_2 provides $\{v,w,y\}$ and Q_1 provides $\{x,y,z\}$, both CQs have free-connex union extension:

$$Q_1^+(w,y,x,z) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x), P_1(v,w,y) \text{ and}$$
$$Q_2^+(x,y,w,v) \leftarrow R_1(w,v), R_2(v,y), R_3(y,z), R_4(z,x), P_2(x,y,z).$$

By Theorem 5.8 it follows that $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$. $\square$

Intuitively, the reason why the reduction of Example 5.18 fails in Example 5.19 is the fact that all the variables of the free-paths in one CQ, which are used to encode matrix multiplication, are free in the other CQ. Indeed, if we encode matrices A and B to the relations of the free-path w,v,y in Q_1, there can be up to n^3 many answers to Q_3 in the worst case. The answer set in this case is too large to contradict the assumed lower bound of $\mathcal{O}(n^2)$ for matrix multiplication. As it turns out, there are cases where we cannot reduce matrix multiplication to a union in this manner, and yet we can show that it is intractable using an alternative computational problem.

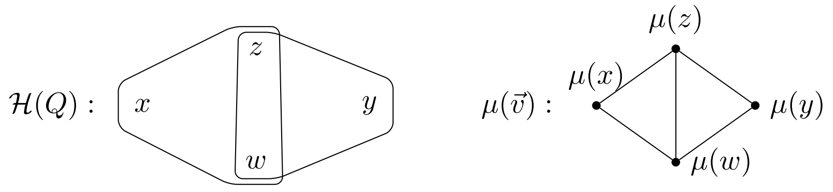


Figure 5.2: If $\mu|_{free(Q_i)} \in Q(\mathcal{D})$, the induced subgraph of $\mu(w, x, y, z)$ forms a clique where one edge might be missing.

Example 5.20. Let Q be the UCQ

$$Q_1(x, y, t), Q_2(x, y, w) \leftarrow R_1(x, w, t), R_2(y, w, t).$$

We can show that this union is intractable under the 4-CLIQUE assumption. For a given graph $G = (V, E)$ with $|V| = n$, we compute the set T of all triangles in G in time n^3. Define a database instance $\mathcal{D}$ as $R_1^{\mathcal{D}} = R_2^{\mathcal{D}} = T$. For every output $\mu|_{free(Q_i)} \in Q(\mathcal{D})$ with $i \in \{1, 2\}$, we know that $(\mu(x), \mu(z), \mu(w))$ and $(\mu(y), \mu(w), \mu(z))$ are triangles. If $\mu(x) \neq \mu(y)$, this means that $\mu((x, y)) \in E$ if and only if $(\mu(x), \mu(y), \mu(w), \mu(z))$ forms a 4-clique (see Figure 5.2). Since there are $\mathcal{O}(n^3)$ answers to Q, if $\text{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$, we can check whether $\mu((x, y)) \in E$ for every answer in $Q(\mathcal{D})$, and determine whether a 4-clique appears in G in time $\mathcal{O}(n^3)$. $\square$

Note that we can use the 4-CLIQUE assumption in Example 5.20, since, in addition to the free-path variables, there is another variable in both free-path relations. We now generalize the structural observations we made in Examples 5.20 and 5.18, that allowed us to reduce the enumeration problem to some hard computational decision problem.

Definition 5.21. Let $Q = Q_1 \cup Q_2$ be a UCQ where Q_1 and Q_2 are body-isomorphic.

- Q_1 is said to be *free-path guarded* if for every free-path P in Q_1, we have that $var(P) \subseteq \text{free}(Q_2)$.
- Let $P = (u_1, \ldots u_k)$ be a path in Q_1. Two atoms $R_1(\vec{v}_1)$ and $R_2(\vec{v}_2)$ of Q_1 are *subsequent P-atoms* if $\{u_{i-1}, u_i\} \subseteq \vec{v}_1$ and $\{u_i, u_{i+1}\} \subseteq \vec{v}_2$ for some $1 < i < k$.
- Q_1 is said to be *bypass guarded* if for every free-path P in Q_1 and variable u that appears in two subsequent P-atom, we have that $u \in \text{free}(Q_2)$.

Note that every free-connex CQ is trivially free-path guarded and bypass guarded.

Example 5.22. Consider the UCQ $Q = Q_1 \cup Q_2$ from Example 5.19. Using the notation for body-isomorphic queries, the only free-path of Q_1 is $P = (w, v, y)$. As $\{w, v, y\} \subseteq \text{free}(Q_2)$, the CQ Q_1 is free-path guarded. Moreover, we have that $R_1(w, v)$ and $R_2(v, y)$ in atoms(Q) are subsequent P-atoms. Since $v \in \text{free}(Q_2)$, the query Q_1 is bypass guarded as well.

The query Q_1 from Example 5.18 and Example 5.20 is not free-path respectively not bypass guarded. Indeed, denote by $Q = Q_1 \cup Q_2$ the UCQ from Example 5.18. The variables of the free-path $P' = (w, v, y)$ of Q_1 are not contained in free(Q_2), thus Q_1 is not free-path guarded. Finally, denoting by $Q = Q_1 \cup Q_2$ the query from Example 5.20, consider the free-path $P'' = (x, w, y)$ of Q_1. The atoms $R_1(x, w, t)$ and $R_2(y, w, t)$ in atoms(Q) are subsequent P'' atoms. Since t is contained in both atoms but not in free(Q_2), we have that Q_2 is not bypass guarded. $\square$

In the following two lemmas, we prove that if some CQ in a union is either not free-path guarded or bypass guarded, then the UCQ is intractable. The first lemma shows that the reduction in Example 5.18, where we can use the fact that Q_1 is not free-path guarded to compute matrix multiplication, can be constructed in the general case as well.

Lemma 5.23. *Let $Q = Q_1 \cup Q_2$ be a UCQ of self-join free body-isomorphic acyclic CQs. If Q_1 is not free-path guarded, then* Enum$\langle Q\rangle$ *is not in* $\mathsf{DelayC_{lin}}$*, assuming* mat-mul.

Proof. Let A and B be Boolean $n \times n$ matrices represented as binary relations, i.e. $A \subseteq \{1, \ldots, n\}^2$, where $(a, b) \in A$ means that the entry in the a-th row and b-th column is 1. Further let $P = (z_0, \ldots, z_{k+1})$ be a free-path in Q_1 that is not guarded, and let $0 \leq i \leq k + 1$ be minimal with $z_i \notin$ free(Q_2). For any $(a, b, c) \in (\{1, \ldots, n\} \cup \{\bot\})^3$ we define a function $\tau_{(a,b,c)} : \mathit{var}(Q) \to \{a, b, c, \bot\}$ as follows:

If $0 < i < k + 1$, for every $v \in \mathit{var}(Q)$ we define

$$\tau_{(a,b,c)}(v) = \begin{cases} a & \text{if } v \in \{z_0, \ldots, z_{i-1}\}, \\ b & \text{if } v = z_i, \\ c & \text{if } v \in \{z_{i+1}, \ldots, z_{k+1}\}, \\ \bot & \text{otherwise,} \end{cases}$$

and if $i = 0$ or $i = k + 1$ we set

$$\tau_{(a,b,c)}(v) = \begin{cases} a & \text{if } v = z_0, \\ b & \text{if } v \in \{z_1, \ldots, z_k\}, \\ c & \text{if } v = z_{k+1}, \\ \bot & \text{otherwise.} \end{cases}$$

Since P is chordless and $k \geq 1$, there is no atom in Q that contains both z_0 and z_{k+1} as variables. Thus we can partition the set atoms(Q) into non-empty sets $\mathcal{A}_1 = \{R(\vec{x}) \in$ atoms$(Q) \mid z_1 \in \vec{x}\}$ and $\mathcal{A}_2 =$ atoms$(Q) \setminus \mathcal{A}_1$. We define a database instance $\mathcal{D}$ over Q as follows: For every $R(\vec{x}) \in \mathit{atoms}(Q)$ with $\vec{x} = (v_1, \ldots, v_m)$, if $R(\vec{x}) \in \mathcal{A}_1$ we set

$$R^{\mathcal{D}} = \{(\tau_{(a,b,\bot)}(v_1), \ldots, \tau_{(a,b,\bot)}(v_m)) \mid (a, b) \in A\},$$

and if $R(\vec{x}) \in \mathcal{A}_2$ we set

$$R^{\mathcal{D}} = \{(\tau_{(\bot,b,c)}(v_1), \ldots, \tau_{(\bot,b,c)}(v_m)) \mid (b, c) \in B\}.$$

Next consider a homomorphism $\mu \in Q(\mathcal{D})$. In the case that $0 < i < k+1$, we have that $\mu(z_0) = \cdots = \mu(z_{i-1}) = a$, $\mu(z_i) = b$ and $\mu(z_{i+1}) = \cdots = \mu(z_{k+1}) = c$ for some $(a,b) \in A$ and $(b,c) \in B$, and in case that $i \in \{0, k+1\}$, we have that $\mu(z_0) = a$, $\mu(z_1) = \cdots = \mu(z_k) = b$ and $\mu(z_{k+1}) = c$ for some $(a,b) \in A$ and $(b,c) \in B$. This is the case since the variables z_i are connected via the path in both CQs. In either case, $\mu(\text{free}(Q_1))$ is a tuple only containing the values a, c and $\perp$. If $0 < i < k+1$, $\mu(\text{free}(Q_2))$ is a tuple only containing the values a, c and $\perp$ and if $i \in \{0, k+1\}$ then $\mu(\text{free}(Q_2))$ is a tuple only containing the values a, b and $\perp$ or b, c and $\perp$. Thus the overall output is at most of size $2n^2$. Since the (a,c) pairs in the output $Q_1(\mathcal{D})$ correspond to tuples of the matrix multiplication, we cannot enumerate the solutions of $Q(\mathcal{D})$ with linear preprocessing and constant delay, assuming MAT-MUL. □

In Example 5.20, we encounter a UCQ where both CQs are free-path guarded, but Q_1 is not bypass guarded. For every UCQ with this property, we can show that solving the enumeration problem in $\mathsf{DelayC_{lin}}$ also solves 4-CLIQUE.

Lemma 5.24. *Let $Q = Q_1 \cup Q_2$ be a UCQ of self-join free body-isomorphic acyclic CQs. If Q_1 and Q_2 are free-path guarded and Q_1 is not bypass guarded,*

then ENUM$\langle Q \rangle \notin \mathsf{DelayC_{lin}}$, *assuming 4-*CLIQUE.

Proof. Let $G = (V, E)$ be a graph with $|V| = n$. We show how to solve the 4-CLIQUE problem on G in time $\mathcal{O}(n^3)$ if ENUM$\langle Q \rangle$ is in $\mathsf{DelayC_{lin}}$. Let P be a free-path in Q_1 and let $u \notin \text{free}(Q_2)$ such that u appears in two subsequent P-atoms. We first show that P is of the form (z_0, z_1, z_2). To do so, let $P = (z_0, \ldots, z_m)$ for $n \geq 2$ and $1 \leq i \leq m-1$ such that $\{u, z_{i-1}, z_i\}$ and $\{u, z_i, z_{i+1}\}$ are contained in edges of $\mathcal{H}(Q)$. As P is chordless, there is no edge containing $\{z_{i-1}, z_{i+1}\}$, thus the path (z_{i-1}, u, z_{i+1}) is a chordless path. As Q_1 is free-path guarded, $z_{i-1}, z_{i+1} \in \text{free}(Q_2)$ and since $u \notin \text{free}(Q_2)$, this is a free-path of Q_2. Since Q_2 is free-path guarded we have $z_{i-1}, z_{i+1} \in \text{free}(Q_1)$ and since P is a free-path in Q_1 we have that $i = 1$ the path is of length two.

Let R_1 and R_2 be atoms with $\{z_0, z_1, u\} \subseteq var(R_1)$ and $\{z_1, z_2, u\} \subseteq var(R_1)$. Further let (a,b,c) be a triangle in G. We define a mapping $\tau_{(a,b,c)}$ on variables of Q in order to encode this triangle to tuples $r \in R^{\mathcal{D}}$ of a database over Q, such that $var(R)$ either contains $\{z_0, z_1, u\}$ or $\{z_1, z_2, u\}$. That is, given some $v \in var(Q)$, we define

$$\tau_{a,b,c}(v) = \begin{cases} a & \text{if } v = z_0 \text{ or } v = z_2, \\ b & \text{if } v = z_1, \\ c & \text{if } v = u, \\ \perp & \text{otherwise.} \end{cases}$$

For every atom $R(v_1, \ldots, v_s) \in \text{atoms}(Q)$, we define

$$R^{\mathcal{D}} = \{(\tau_{a,b,c}(v_1), \ldots, \tau_{a,b,c}(v_s)) \mid (a,b,c) \text{ is a triangle in } G\}.$$

Note that $|R^{\mathcal{D}}| \in \mathcal{O}(n^3)$, as there are at most n^3 triangles in G, and that we can construct $\mathcal{D}$ within $\mathcal{O}(n^3)$ steps. Now consider a homomorphism $\mu : \mathit{var}(Q) \to V$ mapping Q into the database. Since $(\mu(z_0), \mu(z_1), \mu(u))$ and $(\mu(z_1), \mu(z_2), \mu(u))$ form triangles in G, we have that G contains a 4-clique if and only if $(\mu(z_0), \mu(z_2)) \in E(Q)$. As $z_0, z_1 \in \text{free}(Q_1)$ it suffices to check every $\mu|_{free(Q_1)} \in Q(\mathcal{D})$ for this property. We have that $\{z_0, z_1, z_2, u\}$ is neither contained in free(Q_1) nor in free(Q_2). Thus, $|Q(\mathcal{D})| \in \mathcal{O}(n^3)$. If Enum$\langle Q\rangle$ is in $\mathsf{DelayC_{lin}}$, we can construct $\mathcal{D}$, output $Q(\mathcal{D})$ and check every output for an edge of the form $(\mu(z_0), \mu(z_2))$ in time $\mathcal{O}(n^3)$, which is a contradiction to 4-clique. □

For the rest of the section, we aim to show that any UCQ that is not covered by Lemma 5.23 and Lemma 5.24 is in fact union-free-connex. To prove this, we first need a structural property given as follows.

Lemma 5.25. *Let $Q = Q_1 \cup Q_2$ be a UCQ of body-isomorphic acyclic CQs, where Q_1 and Q_2 are free-path guarded, and Q_1 is bypass guarded, and let P be a free-path in Q_1. Then there exists a join-tree T for Q with a subtree T_P such that $\mathit{var}(P) \subseteq \mathit{var}(T_P)$, and every variable that appears in two different atoms of T_P is in* free(Q_2).

Proof. Consider a path $A_1, \dots, A_s$ between two atoms on a join tree. We define a *contraction step* for a path of length 2 or more: if there is A_j such that $A_j \cap A_{j+1} \subseteq A_1 \cap A_s$, then remove the edge (A_j, A_{j+1}) and add the edge (A_1, A_s). The new graph is still a join-tree since all of the atoms on the path between A_1 and A_s contain $A_1 \cap A_s$, and in particular they contain $A_j \cap A_{j+1}$. The unique path on the join-tree between the atoms A_1 and A_s is now of length one. A path on a join-tree is said to be *fully-contracted* if none of its subpaths can be contracted.

Now let T be a join-tree of Q, and let $P = (z_0, \dots, z_{k+1})$. We consider some path in T between an atom containing $\{z_0, z_1\}$ and an atom containing $\{z_k, z_{k+1}\}$. Take the unique subpath T_P of it containing only one atom with $\{z_0, z_1\}$ and one atom with $\{z_k, z_{k+1}\}$, and fully contract it. Note that T_P contains $\mathit{var}(P)$ due to the running intersection property.

First, we claim that every variable u that appears in two or more atoms of T_P is part of a chordless path from z_0 to z_{k+1}. We first show a chordless path from u to z_{k+1}. Denote the last atom on T_P containing u by A_i. If A_i contains z_{k+1}, we are done. Otherwise, consider the subpath A_{i-1}, A_i, A_{i+1}. Since it is fully contracted, $A_i \cap A_{i+1} \not\subseteq A_{i-1} \cap A_{i+1}$. This means that there is a variable v in A_i and in A_{i+1} that does not appear in A_{i-1}. Now consider the last atom containing v, and continue with the same process iteratively until reaching z_{k+1}. Do the same symmetrically to find a chordless path from u to z_0. Note that the concatenation of the two paths is chordless by construction and since z_0 and z_{k+1} are not neighbors.

Assume by contradiction that there is a variable $u \notin \text{free}(Q_2)$ that appears in two distinct atoms of T_P. There is a chordless path from z_0 to z_{k+1} that contains u. Take a subpath of it starting with the first variable before u which is in free(Q_2), and ending with the first variable after u which is in free(Q_2). This is a free-path in Q_2, and since Q_2 is free-path guarded, $u \in$ free(Q_1). Next consider two neighboring atoms on T_P that contain u. There exists some z_i that appears in both atoms. Note that $i > 0$ and $i < k+1$ since the path only contains one atom with z_0 and one atom with z_{k+1}. Since Q_1 is bypass guarded, u is not a neighbor of both z_{i-1} and z_{i+1}. Without loss of generality, assume it is not a neighbor of z_{i+1}. Then there is a chordless path $(u, z_i, z_{i+1}, \dots, z_{k+1})$. Since $u \in$ free(Q_1), it is a free-path. This contradicts the fact that Q_1 is free-path guarded since $u \notin$ free(Q_2). □

We are now ready to show that the properties free-path guardedness and bypass guardedness imply free-connexity.

Lemma 5.26. *Let $Q = Q_1 \cup Q_2$ be a UCQ of body-isomorphic acyclic CQs. If Q_1 and Q_2 are both free-path guarded and bypass guarded, then Q is free-connex.*

Proof. We describe how to iteratively build a union extension for each CQ. In every step we take one free-path among the queries in Q and add a virtual atom in order to eliminate this free-path. We show that this eventually leads to free-connex union extensions.

Let $P = (z_0, \dots, z_{k+1})$ be a free-path in Q_1. Take T_P according to Lemma 5.25, and denote by V_P the variables $var(P)$ and all variables that appear in more than one atom of T_P. First we claim that Q_2 provides V_P. It is guaranteed that $V_P \subseteq$ free(Q_2), so we only need to show that Q_2 is acyclic V_P-connex. Take the join-tree T from Lemma 5.25. For every vertex A_i in T_P, add another vertex with $A'_i = var(A_i) \cap V_P$ and an edge (A_i, A'_i). Then, for every edge (A_i, A_j) in T_P, add the edge (A'_i, A'_j) and remove (A_i, A_j). The running intersection property is maintained since for every edge (A_i, A_j) removed, $var(A_i) \cap var(A_j) \subseteq V_P$, meaning that all vertices on the path A_i, A'_i, A'_j, A_j contain $var(A_i) \cap var(A_j)$. The subtree containing the new vertices contains exactly the variables V_P.

We add the atom $R(V_P)$ to both Q_1 and Q_2 and obtain Q_1^+ and Q_2^+ respectively. After this extension there are no free-paths that start in z_0 and end in z_{k+1} since they are now neighbors. If both of the CQs are now free-connex, then we are done. Otherwise, we use the extension iteratively, as we will show below that for the UCQ $Q_1^+ \cup Q_2^+$, both Q_1^+ and Q_2^+ are free-path guarded and bypass guarded. Note that after a free-path from z_0 to z_{k+1} is treated, and even after future extension, there will never be a free-path from z_0 to z_{k+1} since they are now neighbors. Since there is a finite number of variable pairs, at some point all pairs that have a free-path between them are resolved, this process stops, and there are no free-paths.
It is left to prove that the conditions for this lemma hold also for the extension as long as at least one of the CQs is not free-connex.

Claim 1: Q_1^+ and Q_2^+ are body-isomorphic acyclic.

Proof of Claim 1. We show a join-tree for the extension. Take the join-tree T according to Lemma 5.25, and add a vertex v that contains exactly V_P. Remove all edges in T_P and add an edge between every atom in T_P and v. The running intersection property is preserved since V_P contains every variable that appears in more than one atom connected to v. $\square$

Claim 2: Q_1^+ and Q_2^+ are free-path guarded.

Proof of Claim 2. First note that the lemma holds for every free-path in the extension that is also a free-path in the original query. The only edge that was added in the extension contains V_P. Thus, a new free-path $(v_0, \ldots, v_{m+1})$ contains $v_j, v_{j+1} \in V_P \subseteq \text{free}(Q_2) = \text{free}(Q_2^+)$. In particular, Q_2^+ does not contain new free-paths.

Let $P' = (v_0, \ldots, v_{m+1})$ be a free-path in Q_1^+ but not in Q_1, and let $v_j, v_{j+1} \in V_P$. Note that no other variable in P' is in V_P, otherwise this path cannot be chordless in Q_1^+. So, $P_s = (v_0, \ldots, v_j)$ and $P_t = (v_{j+1}, \ldots, v_{m+1})$ are chordless paths in Q_1. Every two variables in V_P are connected in Q via a path containing only variables in V_P. Thus, there is a chordless path $P_{mid} = (v_j, t_1, \ldots, t_r, v_{j+1})$ in Q_1 with $r \geq 1$ and $\{t_1, \ldots, t_r\} \cap \{v_0, \ldots, v_{m+1}\} = \emptyset$. That is, $var(P_s)$, $var(P_{mid})$ and $var(P_t)$ are pairwise disjoint. It is possible to show that since the query is acyclic, the concatenation of these three paths after perhaps skipping the connection points (v_j, v_{j+1} or both) is a chordless path in Q_1. Denote this chordless subpath by P_0.

If $t_1, \ldots, t_r \notin \text{free}(Q_1)$, then P_0 is a free-path in Q_1. If there is some $t_l \in \text{free}(Q_1^+)$, let t_s and t_t be the first and last elements in $t_1, \ldots, t_r$ in $\text{free}(Q_1^+)$. Then the subpath of P_0 between v_0 and t_s and the path of P_0 between t_t and v_{m+1} are both free-paths in Q_1. In both cases we get that $\{v_0, \ldots, v_{j-1}, v_{j+1}, \ldots, v_{m+1}\}$ appear in free-paths in Q_1. Since Q_1 is free-path guarded, these variables are in $\text{free}(Q_2)$. Since also $v_j, v_{j+1} \in V_P \subseteq \text{free}(Q_2)$ and $\text{free}(Q_2) = \text{free}(Q_2^+)$, we get that $var(P') \subseteq \text{free}(Q_2^+)$. $\square$

Claim 3: Q_1^+ and Q_2^+ are bypass guarded.

Proof of Claim 3. First let $P' = (t_0, \ldots, t_{m+1})$ in Q_2^+, and assume by contradiction that there exists some $u \notin \text{free}(Q_1^+)$, that appears with two subsequent P'-atoms. This means that Q_2^+ has an atom containing $\{t_{i-1}, t_i, u\}$ and an atom containing $\{t_i, t_{i+1}, u\}$ with $0 < i < m + 1$. As explained in the previous claim, Q_2^+ has no new free-paths, so P' is a free-path in Q_2 as well. Due to the conditions of the lemma, u does not appear in two subsequent P'-atoms in Q_2, so one of these atoms is new in Q_2^+. Assume without loss of generality that it is $\{t_i, t_{i+1}, u\}$. Then $\{t_i, t_{i+1}, u\} \subseteq V_P \subseteq \text{free}(Q_2)$. But this contradicts the fact that $t_i \notin \text{free}(Q_2)$ since P' is a free-path.

Now let $P' = (t_0, \ldots, t_{m+1})$ in Q_1^+. According to Claim 2, $var(P') \subseteq \text{free}(Q_2^+)$. Assume by contradiction that there exists some $u \notin \text{free}(Q_2^+)$, that appears with two subsequent P'-atoms. This means that Q^+ has an atom containing $\{t_{i-1}, t_i, u\}$ and an atom containing

$\{t_i, t_{i+1}, u\}$. Since P' is chordless, t_{i-1} and t_{i+1} are not neighbors, then (t_{i-1}, u, t_{i+1}) is chordless, and is in fact a free-path in Q_2^+. According to Claim 2 again, $\{t_{i-1}, u, t_{i+1}\} \subseteq \text{free}(Q_1^+)$. This means that $i = 1$ and $P' = (t_0, t_1, t_2)$. Since $u \notin \text{free}(Q_2)$ and $V_P \subseteq \text{free}(Q_2)$, we have that the atoms containing $\{t_0, t_1, u\}$ and $\{t_1, t_2, u\}$ appear also in Q_1. So $u \notin \text{free}(Q_2)$ appears in two subsequent atoms of P' in Q_1, in contradiction to the lemma's conditions. □

Since all conditions of the lemma hold after a step of extension, we can iteratively perform more steps until we reach a free-connex extension. □

Since Lemma 5.23, Lemma 5.24 and Lemma 5.26 cover all cases of a union of two self-join-free body-isomorphic acyclic CQs, we have a dichotomy that characterizes the UCQs discussed in this section.

Theorem 5.27. *Let $Q = Q_1 \cup Q_2$ be a UCQ of self-join free body-isomorphic CQs.*

- *If Q_1 and Q_2 are both free-path guarded and bypass guarded,then Q is free-connex and* $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.
- *Otherwise, Q is not free-connex and* $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$*, assuming* HYPERCLIQUE, MAT-MUL *and 4*-CLIQUE.

Proof. By Theorem 5.15, if the CQs are cyclic, $\textsc{Enum}\langle Q\rangle$ is not in $\mathsf{DelayC_{lin}}$ assuming HYPERCLIQUE. Now assume that the CQs are acyclic. By Lemma 5.26, if Q_1 and Q_2 are both free-path guarded and bypass guarded, then Q is free-connex, and $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$ by Theorem 5.8. By Lemma 5.23 and Lemma 5.24, if one of Q_1 and Q_2 are not free-path guarded or not bypass guarded, then $\textsc{Enum}\langle Q\rangle$ is not in $\mathsf{DelayC_{lin}}$ (assuming MAT-MUL and 4-CLIQUE), and then Q is not free-connex by Theorem 5.8. □

5.3 Towards a Dichotomy

So far, regarding lower bounds, we have characterized the case of two body-isomorphic CQs. In this section we examine the next steps that are required to fully characterize which UCQs are in $\mathsf{DelayC_{lin}}$. We pinpoint some of the difficulties that must be tackled when formulating such a dichotomy, accompanied by examples. In Section 5.3.1 we discuss unions containing only acyclic CQs, and in Section 5.3.2 we discuss those that contain at least one cyclic CQ.

5.3.1 Unions of Acyclic CQs

We inspect two ways of extending the results of Section 5.2.3. The first is to a union of two CQs that are not body-isomorphic. If there is an intractable CQ Q_1 in the union where for every other CQ Q_i in the union there is no body-homomorphism from Q_i to Q_1, we can reduce Q_1 to the union as described in Lemma 5.12. Since Q_1 is intractable, the union is intractable too. In case there is a body-homomorphism to the hard queries, one might think that it is sufficient for intractability to have unguarded intractable structures similarly to the previous section. The following example shows that this is not the case.

Example 5.28. Let Q be a UCQ $Q = Q_1 \cup Q_2$ with

$$Q_1(x, y, w) \leftarrow R_1(x, z), R_2(z, y), R_3(y, w) \text{ and}$$
$$Q_2(x, y, w) \leftarrow R_1(x, t_1), R_2(t_2, y), R_3(w, t_3).$$

We first note that the query Q_1 is acyclic but not free-connex, while Q_2 is free-connex. Moreover, we have that Q_1 is not contained in Q_2 but there is a body-homomorphism from Q_2 to Q_1. The variable z is part of the free-path (x, z, y) in Q_1, but the variables t_1 and t_2 that map to it via the body-homomorphism are not free in Q_2. If we extend the notion of guarding to non-body-isomorphic CQs in the natural way, the free-path (x, z, y) is not guarded. Nevertheless, we cannot compute matrix multiplication in $\mathcal{O}(n^2)$ time by encoding it to the free-path (x, z, y) and enumerating the answers to Q in constant delay like before. Indeed, for such a construction, there can be n^3 many results to Q_2 as w and y are not connected in Q_2, and they can have distinct values. This is not an issue when discussing body-isomorphic CQs. We do not know whether the enumeration problem for this query is in $\mathsf{DelayC_{lin}}$. □

A future characterization of the union of CQs that are not body-isomorphic would need an even more careful approach than the one used in Section 5.2.3 in order to handle the case that variables mapping to the free-path are not connected via other variables mapping to the free-path.

A second way of extending Section 5.2.3 is to consider more than two body-isomorphic acyclic CQs. This case may be tractable or intractable, and for some queries of this form, we do not have a classification yet. Example 5.10 shows a tractable union of such form, while the following example shows that free-paths that share edges can be especially problematic within a union.

Example 5.29. Let $k \geq 4$ and consider the UCQ Q containing k body-isomorphic CQs, and a set of atoms $R_i(x_i, z)$ for every $1 \leq i \leq k-1$. The free variables for each CQ are all possible combinations of $k-1$ out of the k variables of the query, $\{z, x_1, \dots, x_{k-1}\}$. In the case of $k = 4$, this amounts to the following query:

$$Q_1(x_1, x_2, x_3), Q_2(x_1, x_2, z), Q_3(x_1, x_3, z), Q_4(x_2, x_3, z)$$
$$\leftarrow R_1(x_1, z), R_2(x_2, z), R_3(x_3, z).$$

The CQ Q_1 has free-paths (x_i, z, x_j) between all possible pairs of i and j. In this case, enumerating the solutions to the UCQ is not in $\mathsf{DelayC_{lin}}$, assuming 4-clique: Given a graph $G = (V, E)$, encode to each relation the edges in the input graph, and concatenate the variable names. That is, for every edge $(u, v) \in E$ and $1 \leq i \leq 3$, add $((u, x_i), (v, z))$ to $R_i^{\mathcal{D}}$. By concatenating variable names, we can identify which solutions come from which CQ, and ignore the answers to all CQs other than Q_1. Any answers to Q_1 gives us 3 vertices that have a common neighbor. We can check in constant time if every pair of the 3 vertices are neighbors, and if so, we found a 4-clique. As there can be $\mathcal{O}(n^3)$ solutions the the UCQ, we solve 4-clique in $\mathcal{O}(n^3)$ time. This proves that the UCQ is intractable assuming 4-CLIQUE.

With the same strategy, we can solve k-clique in time $\mathcal{O}(n^{k-1})$, but this does not result in a lower bound for any $k \geq 5$: It is assumed that one can not find a k-clique in time $n^{\frac{\omega k}{3}-o(1)}$, which does not contradict an $\mathcal{O}(n^{k-1})$ algorithm. However, this reduction does not seem to fully capture the hardness of this query, as it encodes all relations with the same set of edges. We do not know if queries of the structure given here are hard in general, or if they become easy for larger k values, as any current approach that we know of for constant delay enumeration fails for them. ☐

We next describe some classification we can achieve when we exclude classes of UCQs that have a structure similar to that of Example 5.29. In some sense, we generalize the concepts of both free-path guarded and bypass guarded queries. First note that when generalizing the notion of guarding free-paths to unions of several CQs, a free-path does not have to be guarded by a single CQ.

Definition 5.30. Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of body-isomorphic CQs, and let $P = (z_0, \ldots, z_{k+1})$ be a free-path in Q_1. We say that a set $\mathcal{U} \subseteq 2^{var(P)}$ is a *union guard* for P if:

- $\{z_0, z_{k+1}\} \in \mathcal{U}$.
- For every $\{z_a, z_c\} \subseteq u \in \mathcal{U}$ with $a + 1 < c$, we have that $\{z_a, z_b, z_c\} \in \mathcal{U}$ for some $a < b < c$.
- For every $u \in \mathcal{U}$, we have $u \subseteq \text{free}(Q_i)$ for some $1 \leq i \leq n$.

We say that P is *union guarded* if there exists some $\mathcal{U} \subseteq 2^{var(P)}$ that is a union guard for P.

We sometimes refer to the set $\{z_a, z_b, z_c\}$ with $a < b < c$ from Definition 5.30 as (z_a, z_b, z_c). The idea of the definition is as follows: Consider a UCQ $Q = Q_1 \cup \ldots \cup Q_n$, and let $P = (z_0, \ldots, z_{k+1})$ be a free-path in Q_1. If we have that $\{z_0, z_j, z_{k+1}\}$ is not in some set $\text{free}(Q_i)$ for $0 < j < k + 1$ and some Q_i in the union, then we can encode the matrix multiplication problem to the UCQ similar to the construction of Example 5.18 or the proof of Lemma 5.23.
In case such a Q_i exists, we have to be aware of potential subpaths of P that are free-paths in Q_i : As P is a chordless path, there is a chordless paths between any z_s, z_r for $0 \leq s < r \leq k + 1$. Thus, if Q_i does not contain the rest of P as free variables as well, it contains some free-path P' which is a subpath of P. We can simply repeat the argument

of before with P' a free-path and again encode matrix multiplication to the variables of P'. Since any of the sub-paths of a partially covered free-path can become a free-path as well, we need to make sure that union guarded is defined as above in order to fully *guard* free-paths that would allow for such an encoding.

Example 5.31. Let Q be a UCQ $Q = Q_1 \cup Q_2$ with

$$Q_1(z_0, z_4, z_4)Q_2(z_0, z_1, z_3) \leftarrow R_1(z_0, z_1), R_2(z_1, z_2), R_3(z_2, z_3), R_4(z_3, z_4).$$

The query Q_1 contains the free-path $P = (z_0, z_1, z_2, z_3)$. This free-path is not union guarded, and we have seen in the previous section that Q_1 is not free-path guarded either, as $var(P)$ is not included in Q_2. Indeed, although the free variable *guard* the variables z_0, z_1, z_3 of this free-path, the subpath (z_1, z_2, z_3) of P is a free-path in Q_2, which is not guarded. So consider Q when we add another CQ Q_3:

$$Q_1(z_0, z_4, z_4)Q_2(z_0, z_1, z_3), Q_3(z_0, z_2, z_3) \leftarrow R_1(z_0, z_1), R_2(z_1, z_2), R_3(z_2, z_3), R_4(z_3, z_4).$$

Now the path P is fully contained in the union of free(Q_2) and free(Q_3), and we say that Q_1 is union guarded. Using the construction in Lemma 5.23 to encode matrix multiplication to any of the subpaths of P will not give us a contradiction to the hypothesis MAT-MUL. ☐

To formalize this idea, we show that if a UCQ contains a free-path with no union guard, then the entire union is intractable.

Lemma 5.32. *Let $Q = Q_1 \cup \ldots \cup Q_n$ be a UCQ of body-isomorphic acyclic CQs. If there exists a free-path in Q_1 that is not union guarded, then* Enum$\langle Q \rangle \notin \mathsf{DelayC_{lin}}$*, assuming* MAT-MUL.

Proof. Let A and B be two Boolean $n \times n$ matrices and let $P = (z_0, \ldots, z_{k+1})$ be a free-path of Q_1 that is not union guarded. We know that $\{z_0, z_{k+1}\} \subseteq$ free(Q_1). Since P is not union guarded, there exist some a and c such that $0 \le a + 1 < c \le k + 1$, and $\{z_a, z_c\} \subseteq$ free(Q_r) for some $Q_r \in Q$, but for all $Q_s \in Q$ and for all $a < b < c$ we have $\{z_a, z_b, z_c\} \notin$ free(Q_s). Note that $P' = (z_a, z_{a+1}, \ldots, z_c)$ is a free-path of Q_r.

For every $\alpha, \beta, \gamma \in \{1, \ldots, n\}$ we define a function $\tau_{(\alpha,\beta,\gamma)} : var(Q) \to \{\alpha, \beta, \gamma, \bot\}$. For every $v \in var(Q)$ we define

$$\tau_{(\alpha,\beta,\gamma)}(v) = \begin{cases} \alpha & \text{if } v = z_a, \\ \beta & \text{if } v \in \{z_{a+1}, \ldots, z_{c-1}\}, \\ \gamma & \text{if } v = z_c, \\ \bot & \text{otherwise.} \end{cases}$$

Since P' is a free-path, there is no atom in Q that contains both z_a and z_c as variables. Thus we can partition the set atoms(Q) into non-empty sets $\mathcal{A}_1 = \{R(\vec{x}) \in \text{atoms}(Q) \mid$

$z_a \in \vec{x}\}$ and $\mathcal{A}_2 = \text{atoms}(Q) \setminus \mathcal{A}_1$. We define a database instance $\mathcal{D}$ over Q as follows: For every $R(\vec{x}) \in atoms(Q)$ with $\vec{v} = (v_1, \dots, v_s)$, if $R(\vec{x}) \in \mathcal{A}_1$ we set

$$R^{\mathcal{D}} = \{(\tau_{(\alpha,\beta,\bot)}(v_1), \dots, \tau_{(\alpha,\beta,\bot)}(v_s)) \mid (\alpha, \beta) \in A\},$$

and if $R(\vec{x}) \in \mathcal{A}_2$ we set

$$R^{\mathcal{D}} = \{(\tau_{(\bot,\beta,\gamma)}(v_1), \dots, \tau_{(\bot,\beta,\gamma)}(v_s)) \mid (\beta, \gamma) \in B\}.$$

With argument similar to the ones in the proof of Lemma 5.23, we have that the product AB is encoded in $Q_r(\mathcal{D})$, and every $Q_s(\mathcal{D})$ is of size at most $\mathcal{O}(n^2)$. Thus $\text{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$ assuming MAT-MUL. $\square$

We also want to lift the concept of bypass guardedness to the setting of more than two queries in a union. However, as we do not know of a classification for Example 5.29, we restrict the class of UCQs we consider to those where the free-paths within each CQ do not share variables. Doing so, we manage to obtain a similar characterization to that of Section 5.2.3.

Definition 5.33. Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of body-isomorphic CQs, and let P be a free-path of Q_1. We say that P is *isolated* if the following two conditions hold:

- Q is $var(P)$-connex and
- for all free-paths $P' \neq P$ in Q_1 we have $var(P') \cap var(P) = \emptyset$.

Note that the structural property *isolated* is a stronger property than bypass guarded. Given a free-path P, a bypass guarded query can have a variable in two subsequent P-atoms as long as this variable is free in another CQ. An isolated free-path cannot have such a variable at all.

In the previous section, we saw that the union of two body-isomorphic queries that are free-path guarded and bypass guarded are tractable. In the following, we show Theorem 5.36, giving us a similar upper bound for the class of body-isomorphic acyclic CQs that fulfill the generalized notions of guardedness. To achieve this result, we first need to prove Lemma 5.34, which provides a structural property that is then used in order to show Lemma 5.35.

Lemma 5.34. *Let $Q = Q_1, \dots, Q_n$ be a UCQ and $P = (z_0, \dots, z_{k+1})$ be a free-path of Q_1. Further let $\mathcal{U}$ be a union guard of P. There exists some $\mathcal{U}' \subseteq \mathcal{U}$ such that:*

- *$\{z_0, z_{k+1}\} \subseteq v \in \mathcal{U}'$.*
- *for all $1 \leq i \leq m$, $\{z_{i-1}, z_i\} \subseteq v \in \mathcal{U}'$.*
- *there exists a join-tree T_P for $\mathcal{H} = (var(P), \mathcal{U}')$.*

Proof. We use the inductive definition of a union guard to define $\mathcal{U}'$ and T_P. Moreover, for the ease of explanations, we define T_P with a parent-child relation. By the first two points of Definition 5.30, $\{z_0, z_{k+1}\} \in \mathcal{U}$ and thus we have $(z_0, z_j, z_{k+1}) \in \mathcal{U}$ for some $1 \leq j \leq k$. We set (z_0, z_j, z_{k+1}) as a root vertex of T_P. The second condition of Definition 5.30 defines the parent-child condition of T_P with at most two children per vertex: If $v = (z_a, z_b, z_c) \in V(T_P)$, then

- if $b > a + 1$, there exists some $\{z_a, z_j, z_b\} \in \mathcal{U}$ with $a < j < b$. Add one such vertex to the set of children of v.
- if $c > b + 1$, there exists some $\{z_b, z_j, z_c\} \in \mathcal{U}$ with $b < j < c$. Add one such vertex to the set of children of v.

Note that given $\mathcal{U}$, the choice of T_P might not be unique, and the vertices in T_P correspond to a subset $\mathcal{U}'$ of $\mathcal{U}$.

Claim. *T_P is a join tree of $\mathcal{H}$.*

Proof of the Claim. For every $v \in V(T_P)$, denote by $T_P(v)$ the maximal subtree of T_P rooted in v. Note that for $v = (z_a, z_b, z_c)$, we have $\mathit{var}(T_P(v)) \subseteq \{z_a, \dots, z_c\}$ by construction.

We first prove that for every $v_p, v_c \in V(T_P)$, if v_c is a descendent of v_p and $z_i \in v_c \cap v_p$, then z_i is contained in every vertex on the path between v_p and v_c. Let $v_c \in T_P(v_p)$, and by way of contradiction let v_q be the first vertex the path not containing z_i. Then by construction of T_P, the parent of v_q must either be of the form (z_i, z_a, z_b), or (z_a, z_b, z_i). In both cases, v_q is of the form (z_a, z_j, z_b), and $\mathit{var}(T_P(v_q)) \subseteq \{z_a, \dots, z_b\}$. In the first case $a > i$, and in the second case $b < i$. In either case $z_i \notin T_P(v_q)$, which is a contradiction to the fact that $z_i \in v_c$.

Let v_1, v_2 be two distinct vertices in T_P and $z_i \in v_1 \cap v_2$ for some $0 \leq i \leq k + 1$. Let v_3 be a vertex on the unique path from v_2 to v_1. We make a case distinction by how v_1 and v_2 are connected. In case either $v_1 \in T_P(v_2)$ or $v_2 \in T_P(v_1)$, we already showed that v_3 contains z_i. So assume that $v_1 \notin T_P(v_2)$ and $v_2 \notin T_P(v_1)$. This means that there is some $v_p \in V(T_P) \setminus \{v_1, v_2\}$ with $v_1, v_2 \in T_P(v)$, and distinct children u_1, u_2 of v_p such that $v_1 \in T_P(u_1)$ and $v_2 \in T_P(u_2)$. For $v_p = (z_a, z_b, z_c)$ we have that $\mathit{var}(T_P(u_1)) \cap \mathit{var}(T_P(u_2)) = \{z_b\}$ by construction, and since $z_i \in v_1 \cap v_2$ it follows that $z_b = z_i$. Thus, either v_3 is on the unique path from v_p to v_1 or from v_p to v_2. Since v_1 and v_2 are descendents of v_p, we already showed that $z_b \in v_3$. □

Claim. *For every $1 \leq i \leq k + 1$, the set $\{z_{i-1}, z_i\}$ is contained in a vertex of T_P.*

Proof of the Claim. We describe a path from the root to a vertex containing $\{z_{i-1}, z_i\}$. We start with the root, and at each step we consider a vertex $v = (z_a, z_b, z_c)$ such that $a \leq i - 1, i \leq c$. We have that either $i \leq b$ or $i - 1 \geq b$. Assume that $i \leq b$. If $b = a + 1$, we have that $a = i - 1$ and $b = i$, so we found the vertex we need. Otherwise, v has a child (z_a, z_t, z_b) with $a \leq i - 1, i \leq b$. We consider this child next. The case that $i - 1 \geq b$ is symmetrical. Since the tree is finite, the process will end and we will find such vertex. $\square$

This concludes the proof of the Lemma. $\square$

We are now ready to show the positive result for a UCQ of body-isomorphic CQs, similar to that of Lemma 5.26 for the case of a union of two such queries.

Lemma 5.35. *Let $Q = Q_1 \cup \ldots \cup Q_n$ be a UCQ of body-isomorphic acyclic CQs. If every free-path in Q_1 is union guarded and isolated, then Q_1 is union-free-connex with respect to Q.*

Proof. We show how to eliminate a free-path in Q_1 by adding provided virtual atoms. This process can be applied repeatedly to treat all free-paths in Q_1. Let $P = (z_0, \ldots, z_{k+1})$ be a free-path in Q_1, and consider a join-tree T_P given by Lemma 5.34. We extend every CQ Q_j in Q to a union extension Q_j^+ as follows: For every $v \in T_P$, add the atom $R_v(\vec{v})$ to atoms(Q), where R_v is a fresh relational symbol. In Claim 2 we show that these variables sets are provided, so this is indeed a valid union extension. In Claim 3 we show that extension eliminates P without introducing new free-paths. As the proof of Claim 2 uses a bottom-up induction on T_P, we will need to use Claim 1 regarding the subtrees of T_P. We use the same notation as in the proof of Lemma 5.34: For every vertex $v \in V(T_P)$, we denote by $T_P(v)$ the subtree of T_P rooted in v.

Claim 1: Let $v \in V(T_P)$. There exists an ext-$var(T_P(v))$-connex tree T' for the hypergraph $\mathcal{H}' = (var(Q), E(\mathcal{H}(Q)) \cup V(T_P(v)))$.

Proof of Claim 1. Let $v = (z_a, z_b, z_c)$. By construction of T_P, we have that $var(T_P(v)) = \{z_a, z_{a+1}, \ldots, z_c\}$. Denote by R the subpath $(z_a, z_{a+1}, \ldots, z_c)$ of P. It is possible to show that since P is a chordless path, for every subpath of P, we have that Q is $var(R)$-connex. Let T be an ext-$var(R)$-connex tree for $\mathcal{H}(Q)$, and let $T_R \subseteq T$ be a connected subtree of T with $var(T_R) = var(R)$. We construct a new tree T' by first removing the edges among vertices of T_R, adding the tree $T_P(v)$ and then reconnecting the vertices of T_R to $T_P(v)$ as fellows: Let $u \in V(T_R)$. Since R is a chordless path, we have that $u = \{z_s\}$ for some $a \leq s \leq c$ or $u = \{z_s, z_{s+1}\}$ for some $a \leq s \leq c - 1$. Thus there exists some vertex $w \in T_P(v)$ with $u \subseteq w$. Chose some arbitrary $w \in T_P(v)$ with this property and add an edge (u, w).

Since T is a tree, removing the edges of T_R results in a forest. For every $u \in V(T_R)$, the connected component of this forest that contains u does not contain any other vertices in $V(T_R)$. Thus in every step of adding an edge, we attach a new tree to $T_P(v)$, and no such tree is attached to $T_P(v)$ more then once. Thus T' is again a tree and acyclic. $\square$

Claim 2: Q_1^+ is a union extension.

Proof of Claim 2. We prove via a bottom-up induction on T_P that every vertex $v \in V(T_P)$ is provided by some union extension of a CQ in Q.

For the base case, note that the leaves of T_P are triples of the form (z_i, z_{i+1}, z_{i+2}) for $0 \leq i \leq k-1$. Since P is a chordless path, for every subpath of P we have that Q is $\mathit{var}(R)$-connex. Therefore, Q is $\{z_i, z_{i+1}, z_{i+2}\}$-connex. Since also (z_i, z_{i+1}, z_{i+2}) is contained in some $\text{free}(Q_j)$, all leaves are provided.

Consider some vertex $v = (z_a, z_b, z_c)$ of T_P that is not a leaf, and assume that for every child v' of v, we have that every vertex in $T(v')$ is provided. We need to show that v is provided. Consider the ext-$\mathit{var}(T_P(v))$-connex tree T' of the hypergraph $(\mathit{var}(Q), E(\mathcal{H}(Q)) \cup V(T_P(v)))$, that was constructed in the proof for Claim 1. In this tree, we replace the node v by the two nodes $v_1 = \{z_a, z_b\}$ and $v_2 = \{z_b, z_c\}$, and then add the edge (v_1, v_2). For every edge $e = (u, v)$ that was lost when deleting v, we do the following: If u contains the variable z_c, add an edge (u, v_2), otherwise add the edge (u, v_1). Since no vertex in T' besides v contains both z_a and z_c, this is again a valid join tree. Moreover, both $\{z_a, z_b\}$ and $\{z_b, z_c\}$ are contained in vertices of $T' \setminus \{v_1, v_2\}$: If v has two children, then the children contain $\{z_a, z_b\}$ and $\{z_b, z_c\}$. Otherwise, if v has only one child node, we have that either $b = a + 1$ or $c = a + 1$. In both such cases, $\{z_a, z_b\}$ or $\{z_b, z_c\}$ is an edge of the path P and thus contained in a vertex in T'. Thus we have that T' is ext-$\{z_a, z_b, z_c\}$-connex acyclic. Let $Q_j \in Q$ such that $\{z_a, z_b, z_c\} \subseteq \text{free}(Q_j)$. Since every vertex in T' is provided, there exists a union extension Q_j^+ with $Q_j^+(\text{free}(Q)) \leftarrow R_{v_1}(\vec{v_1}), \ldots, R_{v_N}(\vec{v_N})$ and $\{v_1, \ldots, v_N\} = V(T')$. Therefore, Q_j^+ provides v. □

Claim 3: The set of free-paths in Q_1^+ equals the set of free-paths in Q_1 minus P.

Proof of Claim 3. Since $\{z_0, z_{k+1}\}$ is contained in a vertex of T by Lemma 5.34, it is also contained in the variables of some added atom $R_v(\vec{v})$, thus P is not a free-path of Q_1^+. For the sake of a contradiction, assume that there is some new free-path $P' = (z'_0, \ldots, z'_{m+1})$ in Q_1^+. The only new edges in $\mathcal{H}(Q_i^+)$ are between variables in $\mathit{var}(P)$, thus $|\mathit{var}(P) \cap \mathit{var}(P')| \geq 2$. Let z_s and z_t be the first and last elements in P' that are also in P. Note that the z_s and z_t are not neighbors in P. Replacing the path between z_s and z_t in P' with the path between z_s and z_t in P, we get the path $P'' = (z'_0, \ldots, z_s, z_{s+1}, \ldots, z_{t-1}, z_t, \ldots, z'_{m+1})$, which is a path in $\mathcal{H}(Q)$. This path contains a chordless sub-path P''' between z'_0 and z'_{m+1} containing at least one element in $\{z_s, z_{s+1}, \ldots, z_{t-1}, z_t\}$. Thus P''' is a free-path in Q_1 with a non-empty intersection with P, which is a contradiction to the assumption that every free-path is isolated. □

If Q_1^+ is free connex, then we are done. Otherwise, by Claim 3, every free-path P' in Q_1^+ is a free-path in Q_1 and thus union guarded and isolated in Q_1. Since the added atoms only contain variables of $var(P)$ and $var(P) \cap var(P') = \emptyset$, we have that P' is also union guarded and isolated in Q_1^+. The tree structure from Claim 1 over the root is a join-tree for the union extension. Since the extension consists of body-isomorphic acyclic queries, we can iteratively apply this process until Q_1 becomes free-connex. □

The lemmas of this section can be summarized to the following theorem:

Theorem 5.36. *Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of body-isomorphic acyclic CQs.*

- *If every free-path in Q is union guarded and isolated, then* $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$.
- *If there exists a free-path in Q that is not union guarded, then* $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$, *assuming* MAT-MUL.

Proof. For the positive case, we have that every CQ $Q_i \in Q$ is union-free-connex by Lemma 5.35, and thus $\textsc{Enum}\langle Q\rangle \in \mathsf{DelayC_{lin}}$ by Theorem 5.8. The negative case is given by Lemma 5.32. □

Following this theorem, it is left to handle cases like Example 5.29, of unions containing body-isomorphic acyclic CQs where some free-path is union guarded but not isolated.

5.3.2 Unions Containing Cyclic CQs

We close this chapter with a brief discussion on UCQs containing at least one cyclic query. We first mention that many of the observations we have regarding acyclic CQs also apply here. In the following examples, Q_1 is cyclic while Q_2 is free-connex. Example 5.37 shows that unions containing cyclic CQs may be tractable and covered by Theorem 5.8. Example 5.38 demonstrates that it is not enough to resolve the cyclic structures in cyclic CQs, but that we should also handle the free-paths. Finally, Example 5.39 shows that, much like Example 5.28 in the acyclic case, even if all intractable structures are unguarded, the original reductions showing the intractability of single CQs may not work.

Example 5.37. Let Q be a UCQ $Q = Q_1 \cup Q_2$ with

$$Q_1(x,y,z,w) \leftarrow R_1(y,z,w,x), R_2(t,y,w), R_3(t,z,w), R_4(t,y,z) \text{ and}$$
$$Q_2(x,y,z,w) \leftarrow R_1(x,z,w,v), R_2(y,x,w).$$

The query Q_2 provides $\{t,y,z,w\}$ to Q_1. Adding the virtual atom $R'(t,y,z,w)$ to Q_1 results in a free-connex union extension, so the union $Q_1 \cup Q_2$ is tractable according to Theorem 5.8. □

Example 5.38. Let Q be a UCQ $Q = Q_1 \cup Q_2$ with

$$Q_1(x,y,v) \leftarrow R_1(v,z,x), R_2(y,v), R_3(z,y) \text{ and}$$
$$Q_2(x,y,v) \leftarrow R_1(y,v,z), R_2(x,y).$$

The union $Q_1 \cup Q_2$ is intractable despite the fact that Q_2 provides the variables $\{v, y, z\}$ of the cycle in Q_1. This is due to the unguarded free-path (x, z, y) in Q_1. Indeed, similarly to Example 5.18, we can encode matrix multiplication to the variables x, z, y, thus $\textsc{Enum}\langle Q\rangle \notin \mathsf{DelayC_{lin}}$ assuming MAT-MUL. $\square$

Example 5.39. Let Q be a UCQ $Q = Q_1 \cup Q_2$ with

$$Q_1(x, z, y, v) \leftarrow R_1(x, z, v), R_2(z, y, v), R_3(y, x, v) \text{ and}$$
$$Q_2(x, z, y, v) \leftarrow R_1(x, z, v), R_2(y, t_1, v), R_3(t_2, x, v).$$

We do not know the complexity of this example. As Q_2 does not have a free variable that maps to y via a homomorphism, the CQ Q_1 is not union-free-connex, so we cannot use Theorem 5.8 to conclude that $Q_1 \cup Q_2$ is tractable. The only intractable structure in Q_1 is the cycle x, y, z, but encoding the problem TRIANGLE to this cycle in Q_1, like we did in Example 5.16, could result in n^3 answers to Q_2 in the worst case. This means that even if the input graph has triangles, we are not guaranteed to find one in $\mathcal{O}(n^2)$ time by evaluating the union efficiently. $\square$

In addition to these issues, in the cyclic case, even if the original intractable structures are resolved, the extension may introduce new ones. Resolving the following example in general is left for future work.

Example 5.40. We start with a special case of a more general example. For this, consider the union $Q = Q_1 \cup Q_2$ with

$$Q_1(x_2, x_3, x_4) \leftarrow R_1(x_2, x_3, x_4), R_2(x_1, x_3, x_4), R_3(x_1, x_2, x_4) \text{ and}$$
$$Q_2(x_2, x_3, x_4) \leftarrow R_1(x_2, x_3, x_1), R_2(x_4, x_3, v).$$

There exists a body-homomorphism from Q_2 to Q_1, but Q_1 is not contained in Q_2. The query Q_1 is cyclic, as it contains the cycle (x_1, x_2, x_3). This is the only intractable structure in Q_1, i.e. the hypergraph $\mathcal{H}(Q_1)$ does not contain any other hyperclique or a free-path. The query Q_2 is both free-connex and $\{x_2, x_3, x_4\}$-connex, and it provides $\{x_1, x_2, x_3\}$ to Q_1. Nevertheless, extending Q_1 with a virtual atom $R(x_1, x_2, x_3)$ does not result in a free-connex CQ. Even though the extension "removes" all intractable structures from Q_1, it is intractable as it introduces a new intractable structure, namely the hyperclique $\{x_1, x_2, x_3, x_4\}$.

In this case, we have that $\textsc{Enum}\langle Q_1 \cup Q_2\rangle \notin \mathsf{DelayC_{lin}}$ assuming 4-CLIQUE, with a reduction similar to that of Example 5.20. To see this, given an input graph, we can first compute all triangles and encode them to the three relations via a database $\mathcal{D}$. For every triangle $\{a, b, c\}$, add $((a, x_2), (b, x_3), (c, x_4))$ to $R_1^{\mathcal{D}}$, $((a, x_1), (b, x_3), (c, x_4))$ to $R_2^{\mathcal{D}}$ and $((a, x_1), (b, x_2), (c, x_4))$ to $R_3^{\mathcal{D}}$. By concatenating variable names, we can identify which solutions correspond to which CQ, and thus are able to ignore the answers to Q_2. The answers to Q_1 over $\mathcal{D}$ represent 3 vertices that appear in a 4-clique: Indeed, for every answer $((b, x_2), (c, x_3), (d, x_4))$ to Q_1, we know that $((b, x_2), (c, x_3), (d, x_4)) \in R_1^{\mathcal{D}}$, and there exists some value a such that $((a, x_1), (c, x_3), (d, x_4)) \in R_2^{\mathcal{D}}$ and $((a, x_1), (b, x_2), (d, x_4)) \in R_3^{\mathcal{D}}$. This means that $\{a, b, c, d\}$ is a 4-clique. In the opposite direction, for every 4-clique $\{a, b, c, d\}$ we have that $\{b, c, d\}$, $\{a, c, d\}$ and $\{a, b, d\}$ are triangles. By construction, $((b, x_2), (c, x_3), (d, x_4))$ is an answer to Q_1 over $\mathcal{D}$. As there are $\mathcal{O}(n^3)$ triangles and there can be at most $\mathcal{O}(n^3)$ solutions to Q_2, we solve 4-clique in $\mathcal{O}(n^3)$ time. This proves that the UCQ is intractable assuming 4-CLIQUE.

This example can be generalized to higher orders. There, we do not have a similar lower bound. Consider the union $Q = Q_1 \cup Q_2$ as follows:

$$\begin{aligned}
Q_1(x_2, \dots, x_k) &\leftarrow \{R_i(\{1, \dots, k\} \setminus \{i\}) \mid 1 \leq i \leq k-1\}\\
Q_2(x_2, \dots, x_k) &\leftarrow R_1(x_2, \dots, x_{k-1}, x_1), R_2(x_k, x_3, \dots, x_{k-1}, v).
\end{aligned}$$

Again, the query Q_1 is cyclic and Q_2 is free-connex. Even though Q_2 provides the set of variables $\{x_1, \dots, x_{k-1}\}$, adding a virtual atom with these variables does not result in a free-connex extension, as this extension is again cyclic. Just like in Example 5.29, we can encode k-clique to this example in general, but this does not imply a lower bound for any values of k larger than 4. □

CHAPTER 6

Well-Designed Pattern Trees

In this chapter, we turn our attention towards enumerating the answers to wdPTs. We will first establish that unlike before, constant delay enumeration w.r.t. data complexity is not possible even for some of the most basic pattern trees. Therefore, from Section 6.2 onwards, we will shift our focus to that of combined complexity and enumeration of the results with a longer delay. This also means that tractability has a different meaning in the context of wdPTs: In this chapter, we will call a decision problem *tractable* if and only it is solvable in polynomial time or in fixed-polynomial time, and an enumeration problem is tractable if and only if it is within OutputP or OutputFPT.

6.1 Constant Delay of wdPTs

We start with introducing some notation. Recall that a wdPT p is a tuple $p = (T, \lambda, \vec{x})$, where T is a rooted tree, λ maps the nodes of the tree to sets of relational atoms, and $\vec{x}$ are the free variables of the wdPT. Intuitively, such a pattern tree defines a query in the form of several CQs, one in each node. Given a relational database that should be retrieved, each CQ is evaluated over this database, and an answer is obtained by the combined answers of the CQs in a subtree of the wdPT which do not fail to give an answer. Formally, given a database $\mathcal{D}$, a solution μ of p over $\mathcal{D}$ is the projection of a mapping μ', that maps all relational atoms of a subtree T' of T into $\mathcal{D}$, such that there is no mapping μ'' with $\mu' \subset \mu''$ that maps T' into $\mathcal{D}$ as well. The set of all such solutions is denoted by $p(\mathcal{D})$. For a class $\mathcal{C}$ of wdPTs, we denote by $\text{Enum}(\mathcal{C})$ the following enumeration problem:

$\text{Enum}(\mathcal{C})$	
INSTANCE:	Query: wdPT $p \in \mathcal{C}$, Data: database $\mathcal{D}$.
Output:	The set $p(\mathcal{D})$.

In this section, we will evaluate the complexity of this problem w.r.t. data complexity, that is, we assume $p \in \mathcal{C}$ to be fixed. We start by discussing the case of wdPTs without projections, i.e. let $\mathcal{C}_{\mathsf{pf}}$ be the class of wdPTs $p = (T, \lambda, \vec{x})$ such that $\vec{x} = var(p)$. For this class, some of the most basic structures of wdPTs can already be an obstacle for constant delay enumeration:

Proposition 6.1. *Assuming* TRIANGLE, Enum($\mathcal{C}$) *is not in* $\mathsf{DelayC_{lin}}$ *for wdPTs in* $\mathcal{C} \subseteq \mathcal{C}_{\mathsf{pf}}$*, which consist of only a root node with a single child where the root node contains a single binary atom, the child node contains two binary atoms, and the two nodes share two variables.*

Proof. Constructing a pattern tree p with only two nodes R and N and three binary atoms and an appropriate database $\mathcal{D}$, we can decide whether a graph has a triangle by checking whether there is a $\mu \in p(\mathcal{D})$ such that μ is a mapping on the whole pattern tree. Using the fact that $|p|$ is constant, constant delay enumeration with linear preprocessing thus leads to an algorithm detecting a triangle in $\mathcal{O}(n^2)$. So let G be a graph for which we want to solve the triangle problem. We set $\mathcal{D} = E(G)$, and define the following wdPT $p = (T, \lambda, \vec{x})$:

- $V(T)$ consists of a single root node R with child N
- $\lambda(R) = \{e(x, y)\}$, $\lambda(N) = \{e(x, z), e(z, y)\}$
- $\vec{x} = \{x, y, z\}$

First note that this wdPT fulfills the requirements of the proposition. Every mapping $\mu \in p(\mathcal{D})$ is an element of either Ω_1 or Ω_2, where

$$\Omega_1 = \{\{(x, v_1), (y, v_2)\} \mid v_1, v_2 \in V(G)\},$$
$$\Omega_2 = \{\{(x, v_1), (y, v_2), (z, v_3)\} \in \Omega \mid v_1, v_2, v_3 \in V(G)\}.$$

Moreover, we have that $p(\mathcal{D}) \cap \Omega_2 \neq \emptyset$ if and only if G contains a triangle. Assume now that there is an algorithm $\mathcal{A}$ that enumerates all $\mu \in p(\mathcal{D})$ with constant delay and a linear preprocessing. As the pattern tree has a constant size, the asymptotic data complexity of the runtime of the enumeration algorithm equals the combined complexity. The algorithm outputs either only $\mu \in \Omega_1$ and thus halts in $\mathcal{O}(|n|^2 + |\Omega_1|) = O(|n|^2)$, or the algorithm outputs some $\mu \in \Omega_2$ in at most $\mathcal{O}(|n|^2 + |\Omega_1|) = O(|n|^2)$ many steps. Therefore $\mathcal{A}$ allows us to decide in $\mathcal{O}(|n|^2)$ whether G contains a triangle. □

For wdPTs with projection, we need to restrict the class of CQs in the pattern tree for a chance to achieve constant delay with linear preprocessing: Recall that a wdPT with a single node amounts to a CQ, thus a restriction to free-connex acyclic CQs is needed by Theorem 3.1. However, we show below that Enum($\mathcal{C}$) is not in $\mathsf{DelayC_{lin}}$ even for wdPTs where the restriction to free-connex CQs imposed locally on each set of atoms. As in Proposition 6.1, constant delay and linear preprocessing would lead to a contradiction to one of our computational hypothesis.

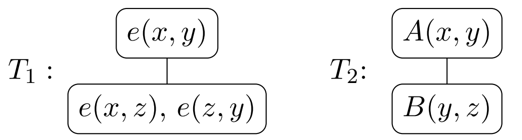


Figure 6.1: T_1 and T_2 form the tree structures of the wdPTs from the proofs of Proposition 6.1 respectively Proposition 6.2.

Proposition 6.2. *Assuming* MAT-MUL*,* ENUM($\mathcal{C}$) *is not in* $\mathsf{DelayC_{lin}}$ *already for wdPTs consisting of only two nodes, each containing a single binary atom and sharing a single variable.*

Proof. Let A, B be Boolean matrices. We define relations $a^{\mathcal{D}} = \{(i, j) \mid A(i, j) = 1\}$ and $b^{\mathcal{D}} = \{(i, j) \mid B(i, j) = 1\}$, and also define the following wdPT $p = (T, \lambda, \vec{x})$:

- $V(T)$ consists of a single root node R with child N
- $\lambda(R) = \{a(x, y)\}$, $\lambda(N) = \{b(y, z)\}$
- $\vec{x} = \{x, z\}$

Assume that we can enumerate $p(\mathcal{D})$ with linear preprocessing and constant delay. By the definition of boolean matrix multiplication, we can construct the product $C = AB$ in the following way:

$$C(i, j) = \begin{cases} 1 & : \{(x, i), (z, j)\} \in p(\mathcal{D}), \\ 0 & : \text{otherwise.} \end{cases}$$

Since $|p(\mathcal{D})| = \mathcal{O}(n^2)$, we can compute C in $\mathcal{O}(n^2)$ which is a contradiction. □

6.1.1 Classes of wdPTs

As we have seen above, even the most basic wdPTs are not suitable for constant delay enumeration after a linear preprocessing. Allowing a slightly longer preprocessing phase however, the solutions can be output with this delay. To achieve this result, we need to introduce classes of wdPTs which have been used to achieve tractability results for evaluation. Not only will we be able to show a positive results for constant delay enumeration in Theorem 6.5 by introducing these classes, but we will use complexity results for the evaluation problem as a tool to achieve bounds for the enumeration problem in Section 6.3.

When looking for classes of wdPT that allow for fast enumeration, a natural starting point are tractable classes of CQs. In the past, several classes of CQs have been successfully used for defining classes of wdPTs achieving tractability for several computational problems, see [BPS15]. There are two different such approaches: Either by locally restricting the CQs defined by each node of the wdPT to be from some tractable class (*"local* *tractability"*), or by globally requiring that for every subtree (containing the root) the corresponding CQ is tractable (*"global tractability"*).

Definition 6.3. Let $\mathcal{C}$ be a class of CQs. A wdPT $p = (T, \lambda, \vec{x})$ is *locally in* $\mathcal{C}$ if for every node $N \in V(T)$ the CQ $Q() \leftarrow \lambda(N)$ is in $\mathcal{C}$. Also, p is *globally in* $\mathcal{C}$ if for every subtree T' of T rooted in R the CQ $Q_{T'}$ is in $\mathcal{C}$. We denote with ℓ-$\mathcal{C}$ and g-$\mathcal{C}$ the sets of all wdPTs that are locally and globally in $\mathcal{C}$, respectively.

In the following, when restricting a wdPT locally or globally to a class of CQs, we will use one of the most fundamental classes of tractable CQs: those of bounded treewidth [CR00, DKV02]. Recall that we denote by $\mathsf{TW}(k)$ the class of all CQs of treewidth at most k, for $k \geq 1$.

For the evaluation problem over projection-free wdPTs, it was shown that local tractability is sufficient to achieve tractability [LPPS13]. However, in the presence of projection, which amounts to the class of wdPTs that we study here, this problem becomes NP-complete [LPPS13]. The same completeness result holds when resorting to global tractability. Thus, just restricting the structure of the CQs defined by a wdPT is not sufficient, and in order to achieve positive results for evaluation or enumeration, we have to restrict the class of wdPTs even further.

A natural source of complexity when dealing with wdPTs, which was identified in [BPS15], comes from the information shared between different nodes. Thus another approach to achieve tractability results is to restrict the number of variables nodes may have in common.

Definition 6.4. let $p = (T, \lambda, \vec{x})$ be a wdPT. For two nodes $N, M \in V(T)$, we define the interface $\mathcal{I}(N, M) = \mathit{var}(N) \cap \mathit{var}(M)$. Similarly, for a node $N \in V(T)$ define $\mathcal{I}(N) = \bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M)$. For $c \geq 0$, we say that p has *c-bounded interface* if $|\mathcal{I}(N)| \leq c$ for all $N \in V(T)$. Similarly, we say that p has *c-semi-bounded interface* if for any two distinct nodes $N, M \in V(T)$ we have $|\mathcal{I}(N, M)| \leq c$. We denote with $\mathsf{BI}(c)$ and $\mathsf{SBI}(c)$ the classes of wdPTs of c-bounded interface and c-semi-bounded interface, respectively.

Condition (2) in the definition of wdPTs says that for a wdPT $p = (T, \lambda, \vec{x})$, for every variable y mentioned in T, the set of nodes of T where y occurs is connected. This means that the interface of a node N can be completely determined by just looking at its neighbors, i.e. for every $N \in V(T)$ we have $\bigcup_{M \in (V(T) \setminus \{N\})} \mathcal{I}(N, M) = \bigcup_{\{M \mid \{N, M\} \in E(T)\}} \mathcal{I}(N, M)$.

6.1.2 Longer Preprocessing Phase

The newly defined classes of wdPTs let us go back and re-evaluate Proposition 6.1 as well as Proposition 6.2. Intuitively, the negative results in the two propositions is mainly caused by the restriction of the preprocessing to linear time. Below, we show that for the class $\mathcal{C}_{\mathsf{pf}}$ of wdPTs without projection, relaxing this restriction leads to a constant delay algorithm. That is, after a preprocessing phase of time $\mathcal{O}(|D|^m f(|p|))$ in terms of combined complexity for a constant $m > 0$ and some computable function f, we can enumerate all answers with a constant delay for wdPTs in class ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$. Note that it is important to require that m be a constant in order to exclude preprocessing of time $\mathcal{O}(|\mathcal{D}|^{|p|})$, which in most cases would suffice to simply compute all of the solutions.

Theorem 6.5. *Let $c, k \geq 1$ be positive integers. Then there exists an enumeration algorithm $\mathcal{A}$ for* $\textsc{Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c) \cap \mathcal{C}_{\mathsf{pf}})$ *with delay*(0) *in* $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^{c+k+1})$ *and delay*(i) *in* $\mathcal{O}(1)$ *for* $i > 0$.

Proof. The idea of the enumeration algorithm $\mathcal{A}$ is as follows: In the preprocessing, we construct a global tree decomposition of all atoms, which is consistent with the structure of the pattern tree (cf. the discussion preceding Theorem 6.8). Then by partitioning the corresponding relations of the nodes of the decomposition and eliminating tuples which are not part of some solution, a repeated top-down traversal through the tree yields all solutions with a delay only in the size of the pattern tree.

So let $p = (T, \lambda, \vec{x})$ be a pattern tree with $\vec{x} = var(T)$, $p \in l$-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$ and $\mathcal{D}$ be a database. We will first describe the precomputation phase and then the enumeration phase of $\mathcal{A}$ for $p(\mathcal{D})$ in several steps.

- For every pattern tree node N, let (S_N, π_N) be the tree decomposition with treewidth of at most k of the Gaifman graph H_N corresponding to $\lambda(N)$. We define a new tree decomposition (S_N, ν_N) with $\nu_N(s) = \pi_N(s) \cup I(N)$ for every $s \in S_N$. The treewidth of this decomposition is at most $c + k$ as the interface is bounded by c, so all relations R_s of the nodes $s \in S_N$ can be computed in total time $|\mathcal{D}|^{c+k+1}$. Thus we can identify the tree decomposition of each N with $\Gamma_N := (S_N, \rho_N)$, where $\rho_N(s) = R_s$ for all $s \in S_N$.
- For every Γ_N, pick one node r_N of S_N as a root and modify the relations R_s by applying a bottom-up semi-join through the tree, followed by a top-down semi-join through the tree of the relations.
- Define a pair (Γ, ρ) as follows: Γ is a tree with

$$\begin{aligned} V(\Gamma) &= \bigcup\{V(S_N) \mid (S_N, \rho_N) = \Gamma_N \text{ for } N \in T\}, \\ E(\Gamma) &= \bigcup\{E(S_N) \mid (S_N, \rho_N) = \Gamma_N \text{ for } N \in T\} \\ &\quad \cup \{e(r_{N_1}, r_{N_2}) \mid (N_1, N_2) \in E(T) \text{ and } r_N \text{ is the root of } S_N\}. \end{aligned}$$

  and ρ is a map on $V(\Gamma)$ such that for all $t \in V(\Gamma)$ we have $\rho(t) = \rho_N(t)$ if $t \in S_N$. This is well-defined, since for all $N, M \in T$ the sets S_N and S_M are disjoint. The pair (Γ, ρ) corresponds to a tree decomposition of the pattern tree p, as for all $v \in var(T)$, the set $\{t \in V(\Gamma) \mid v \in var\rho(t)\}$ is a connected subset of Γ and for every $N \in T$, every edge of H_N is contained in some $\rho(t)$, $t \in V(\Gamma)$. Figure 6.2 illustrates the idea of this tree structure.

- Define an order $\prec_{pt}$ on the pattern tree nodes by a depth-first traversal in pre-order. Further define the following functions on the pattern tree nodes:

$$\text{PTNEXT}(N) = \begin{cases} M & : M \text{ is the successor of } N \text{ according to } \prec_{pt}, \\ \emptyset & : N \text{ is the last pattern tree node according to } \prec_{pt}. \end{cases}$$

$$\text{PTPARENT}(M) = \begin{cases} J & : J \text{ is the parent node of } M, \\ \emptyset & : M \text{ is the root node.} \end{cases}$$

  Analogously we define an order $\prec_N$ on the nodes $V(S_N)$ of every tree decomposition Γ_N, and with a slight abuse of notation we also define the functions tdnext(t) and tdparent(t) on the nodes of each tree decomposition.

- Consider two nodes $s, t \in \Gamma$ such that s is the parent node of t and let R_s and R_t be the corresponding relations. We partition the relation R_t in the following way: For all $\alpha \in R_s$, let α' be the projection of α to . We call tuples $\delta_1 \in R_s$ and $\delta_2 \in R_t$ *compatible*, if they are the same on the shared variables $var(s) \cap var(t)$. Then define $R_t^{\alpha'} = \{\gamma \in R_t \mid \alpha' \text{ is compatible with } \gamma\}$.

The algorithm above describes the precomputation phase of the enumeration algorithm $\mathcal{A}$, which takes $\mathcal{O}(f(|\varphi|)|\mathcal{D}|^{c+k+1})$ steps. The enumeration phase of $\mathcal{A}$ is given by Algorithm 2. For the algorithm, assume that the pattern tree nodes are given as $N_1 \prec_{pt} N_2 \prec_{pt} \cdots \prec_{pt} N_J$ with local roots $r_{N_1}, \ldots, r_{N_J}$. We need to show that Algorithm 2 enumerates $p(\mathcal{D})$ with a constant delay. For this, denote Algorithm 2 by $\mathcal{A}_e$ and by $\mathcal{P}$ the output of $\mathcal{A}_e$.

- $\mathcal{A}_e$ makes a first output after constant time: The algorithm starts by choosing an element $\alpha \in R_{r_1}$. As this tuple is in the root N_1 of the pattern tree and can be extended to a solution on the local tree decomposition of N_1 (by the application of the semi-joins), it can be extended to a full solution in $p(\mathcal{D})$ by the definition of wdPTs. For each pattern tree node N, the precomputation phase added $\mathcal{I}(N)$ to each bag of the local tree decomposition in N. Thus for every tuple t in the root r_N of N, it is fully determined to which of the pattern tree nodes that are children of N t can be extended. In algorithm $\mathcal{A}_e$, the procedure PT-Extension sets the α_j to either the tuples compatible with $\{\alpha_1, \ldots, \alpha_{j-1}\}$ if such a tuple exists, or to α_j otherwise. Then, again by the local tree decomposition in every N and the application of the top-down and bottom-up semi-joins, every tuple t_N can be locally extended to a solution t'_N on the tree decomposition of N. The procedure Local-Extension constructs all such local extensions. Then, by the well-designed property, all such chosen tuples t_N can be combined to a solution. By the construction of the relations R_t^{α}, we only need one top-down traversal of the pattern tree nodes and one top down traversal on every local tree decomposition, which allows the output of the first solution in constant time in terms of data complexity.

- The delay is constant: Let μ_1 and μ_2 be two consecutive outputs of $\mathcal{A}_e$. Then the number of computational steps in $\mathcal{A}_e$ between the output of μ_1 and μ_2 is the

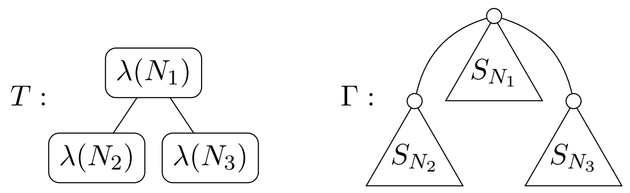


Figure 6.2: T is the tree structure of a wdPT p with nodes N_1, N_2 and N_3. For $1 \leq i \leq 3$, S_{N_i} represents the tree structure of a tree decomposition of $Q() \leftarrow \lambda(N_i)$, and Γ is the tree structure of the constructed tree decomposition of p.

same as to output the first solution. Indeed, let $t_1, \ldots, t_l$ be tuples in the roots of some $N_{i_1}, \ldots, N_{i_l}$ that have been used to extend to the local tree decompositions and thus to μ_1. Then either there is at least one new local extension of some t_i for some $1 \leq i \leq l$, or a new tuples t_j are chosen by at most $|p|$ many calls of the PT-Extension procedure.

- There are no duplicates in $\mathcal{P}$: Assume that μ_1 is in the output of $\mathcal{A}_e$. Then, for any other output μ_2, at least one tuple in some local tree decomposition is chosen differently for μ_1 and μ_2. Since there is no projection, it follows that $\mu_1 \neq \mu_2$.
- $p(\mathcal{D}) = \mathcal{P}$. First let $\mu \in p(\mathcal{D})$. Let N be a pattern tree node such that μ is a solution on N. This means that in the local tree decomposition of N, there is a tuple in the relation of every bag such that the composition of those tuples corresponds to μ on N. If μ is not a solution on some pattern tree node N', then $R_r^\beta = \emptyset$, where β is the projection of α_M to $I(M, N)$, M is the parent of N and α_M is either the empty tuple or the tuple in the root of M which corresponds to μ on M. It follows that $\mu \in \mathcal{P}$, as such compatible tuples are chosen by $\mathcal{A}_e$. So assume that we have some $\mu \in \mathcal{P}$. Then in a top down traversal, compatible tuples in the roots of the local tree decompositions of the pattern tree are chosen. Moreover, such roots are extended to solutions on the local tree decompositions. By construction, a solution on such a local tree decomposition for a pattern tree node N corresponds so some $\mu_N : \vec{x} \cap \mathit{var}(\lambda(N)) \to \mathit{dom}(\mathcal{D})$, and since the solutions on the local tree decompositions are compatible for any output of $\mathcal{A}_e$, the union of all such μ_N, which is also the join of all α_N and thus μ, equals a mapping on the pattern tree p and thus $\mu \in p(\mathcal{D})$.

□

We note that the proof in fact allows to show that in the above theorem, $f(|p|)$ is actually polynomial w.r.t. $|p|$. Thus, given an exponential number of solutions, it is impossible to compute all of them during the preprocessing step.

Theorem 6.5 gives a constant delay algorithm for wdPTs without projection. Aiming for a constant delay algorithm after a polynomial preprocessing phase in the presence of

Algorithm 2 Enumerate all solutions of $p(\mathcal{D})$.

```
Set $\alpha_{N_1}, \dots, \alpha_{N_J}$ to be global variables
procedure Enumerate
    for all $\alpha \in R_{r_1}$ do
        $\alpha_1 = \alpha$
        PT-Extension($N_1$)
    end for
end procedure

procedure PT-Extension(Pattern tree node $N$)
    $M = \text{PTNEXT}(N)$
    if $M = \emptyset$ then
        Local-Extension($N_1$, $r_1$)
    end if
    $J = \text{PTPARENT}(M)$
    $\beta$ = Project $\alpha_J$ down to $I(J, M)$
    if $R^{\beta}_{r_M} = \emptyset$ then
        $R^{\beta}_{r_M} = \{\emptyset\}$
    end if
    for all $\alpha \in R^{\beta}_{r_M}$ do
        $\alpha_M = \alpha$
        PT-Extension($M$)
    end for
end procedure
procedure Local-Extension(Pattern tree node $N$, Tree decomposition node $t$)
    $s = \text{tdnext}(t)$
    if $s = \emptyset$ or $\alpha_N = \emptyset$ then
        $M = \text{PTNEXT}(N)$
        if $M = \emptyset$ then
            Output $\alpha_1 \bowtie \dots \bowtie \alpha_J$ and stop the current Local-Extension procedure
        end if
        Local-Extension($M$, $r_M$)
    end if
    $r = \text{tdparent}(s)$
    $\beta$ = Project $\alpha_N$ down to $var(r) \cap var(s)$
    for all $\alpha \in R^{\beta}_{s}$ do
        $\alpha_N = \alpha_N \bowtie \alpha$
        Local-Extension($N$,$s$)
    end for
end procedure
```

projection, it stands to reason to consider the same question in the setting of CQs, as they are a special case of wdPTs. In [BDG07], Bagan et al. also introduce a weaker version of free-connex acyclicity, namely bounded free-connex treewidth. This class of CQs allows for an enumeration algorithm with polynomial preprocessing phase and constant delay for the enumeration phase, see [BDG07, Theorem 29].

Definition 6.6. Let $G = (V, E)$ be a graph. For $S \subseteq V$, an S-connex tree decomposition of G is a tuple (T, ν, A), where (T, ν) is a tree decomposition of G and A is a connected subset of vertices of T such that $\cup_{v \in V(T)} \nu(v) = S$. The S-connex treewidth of a graph G is the smallest treewidth of any S-connex tree decomposition of G. For a CQ Q, the free-connex treewidth of Q is the free(Q)-connex treewidth of the Gaifman graph of Q.

Let $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Then, even if we allow polynomial time preprocessing instead of a linear one as in Theorem 6.5, there is no constant delay enumeration algorithm for pattern trees in $\{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ assuming that $\mathsf{W[1]} \neq \mathsf{FPT}$. This is due to the fact that enumeration in these classes is not in $\mathsf{OutputFPT}$ under this complexity assumption.

Proposition 6.7. *Let $c \geq 1$ and $\mathcal{C}^*$ be the class of conjunctive queries with bounded free-connex treewidth. Further let $p \in \{g\text{-}\mathcal{C}^*, l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)\}$ and $\mathcal{D}$ be a database. Then there is no positive integer m and computable function f such that $p(\mathcal{D})$ can be enumerated with delay(0) in $\mathcal{O}(|D|^m \cdot f(|p|))$ and delay(i) in $\mathcal{O}(1)$ for $i \geq 1$ unless $\mathsf{FPT} = \mathsf{W[1]}$.*

This proposition depends on results for the evaluation complexity and the induced connection to the enumeration theory, specifically on Theorem 6.16. Thus we will give the proof after the proof of that theorem.

6.2 Parameterized Complexity of Evaluation

As we have established in the previous section, constant delay enumeration with linear (or even polynomial) preprocessing is only possible for some special cases of wdPTs, even when assuming that the size of the query is fixed. Thus the best we can hope for is to achieve enumeration results within the class DelayP when assuming combined complexity. Furthermore, evaluating the enumeration problem in a parameterized setting and taking the size of the query as a parameter, gives us in some sense an intermediate state between data complexity and combined complexity. For parameterized complexity, we aim for enumeration within $\mathsf{DelayFPT}$ or even $\mathsf{OutputP}$.

We start by considering the parameterized complexity of the evaluation problem for wdPTs. As we will see in Theorem 6.16, evaluation and enumeration have in fact a strong connection in this setting.

While there has been some research on the classical complexity of fragments of wdPTs [BPS15], the parameterized complexity is still open. We thus recall two variants of the evaluation problem, namely $\textsc{Eval}(\mathcal{C})$ and $\textsc{Max-Eval}(\mathcal{C})$ and introduce the parameterized versions p-Eval($\mathcal{C}$) and p-Max-Eval($\mathcal{C}$). Here, the size of the query $|p|$ is taken as parameter. Recall that $|p|$ denotes the size of a query $p = (T, \lambda, \vec{x})$, which is defined as the input size of the query Q_T, and $p_m(\mathcal{D})$ is the set of all maximal solutions, which is set of answers in $p(D)$ not subsumed by any other answer.

Eval($\mathcal{C}$) / Max-Eval($\mathcal{C}$)	
INSTANCE:	Query: wdPT $p \in \mathcal{C}$, mapping μ,
	Data: database $\mathcal{D}$.
QUESTION:	Is $\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

p-Eval($\mathcal{C}$) / p-Max-Eval($\mathcal{C}$)	
INSTANCE:	wdPT $p \in \mathcal{C}$, mapping μ, database $\mathcal{D}$.
PARAMETER:	$|p|$
QUESTION:	Is $\mu \in p(\mathcal{D})$ / $\mu \in p_m(\mathcal{D})$?

Observe that in order to talk about the common notions of data- and query complexity, for Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we distinguish which parts of the input are considered to be part of the query, and which are part of the data. For the parameterized case, because of the explicit parameter, such a distinction does not make sense.

Considering the classes of wdPTs introduced in Section 6.1.1, the NP-hardness proof for Eval(g-TW(k)) provided in [BPS15] reveals that hardness even holds for Eval(g-TW(k) $\cap$ SBI(c)), but this class is newly introduced in this work and was not considered in [BPS15]. The introduction of the notion of a semi-bounded interface allows us to define the classes ℓ-TW(k) $\cap$ SBI(c) and g-TW(k) $\cap$ SBI(c). Below, we show that a parameterized complexity analysis of these classes helps to explore the gap between tractability of Eval(ℓ-$\mathcal{C}$ $\cap$ BI(c)) and the NP-completeness of Eval(g-TW(k)). In fact, we show that the parameterized evaluation problem for the different classes of wdPTs is in PTIME (for ℓ-$\mathcal{C}$ $\cap$ BI(c)), in FPT (for g-TW(k) $\cap$ SBI(c)), W[1]-complete (for ℓ-TW(k) $\cap$ SBI(c)), and W[2]-hard (for g-TW(k)), respectively.

Evaluation in case of bounded vs. semi-bounded interface

Before we present our FPT-result for wdPTs in g-TW(k) $\cap$ SBI(c), we first discuss the main difference compared with the restriction to ℓ-TW(k) $\cap$ BI(c). In the latter case, it is easy to construct a global tree decomposition of width $k + c$, s.t. for any two neighboring nodes N, M in the wdPT, the interface $\mathcal{I}(N, M)$ is covered by some bag in the tree decomposition. Indeed, we can take a local tree decomposition for every node N (of width at most k) and add all interface variables to every bag. A global tree decomposition of p is then obtained by gluing together the local tree decompositions. With this global tree decomposition, the FPT-membership (actually, even the PTIME-membership) is easily established [BPS15]. In contrast, for semi-bounded interface, the variables in $\mathcal{I}(N, M)$ for neighboring nodes N, M in p can be arbitrarily scattered over the global tree decomposition. Since the total number of interface variables of a single node in p

with all its neighbors is unbounded, we cannot apply the above trick to construct a tree decomposition where all interfaces are covered by some bag. Hence, a different approach is needed in the following theorem to obtain FPT-membership also in this case.

We first introduce the decision problem EXTENDSOLUTION($\mathcal{C}$) (a similar problem was introduced in [CH97, CV15] to study enumeration problems), defined as follows: given a database $\mathcal{D}$, a wdPT $p \in \mathcal{C}$, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p), does there exist some mapping $\mu' \in p(\mathcal{D})$ such that $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$? Observe that the last property $dom(\mu') \cap \vec{x}' = \emptyset$ is essential, since it allows us to explicitly specify variables $\vec{x}'$ which we do not want to be bound by the desired solutions. Clearly, the problem EVAL($\mathcal{C}$) corresponds to the special case of the EXTENDSOLUTION($\mathcal{C}$) problem where we set $\vec{x}' \cup dom(\mu) = \vec{x}$, i.e., we ask if μ itself is the desired mapping μ'.

Theorem 6.8. *The problem p-EVAL(g-TW(k) $\cap$ SBI(c)) is in FPT for every $k, c \geq 1$.*

Since the full proof of this theorem is very long and technical, we only present the main ideas of the proof here. The full proof can be found in the appendix.

Proof idea. We prove FPT-membership for the EXTENDSOLUTION(g-TW(k) $\cap$ SBI(c)) problem, from which the desired FPT-result follows. Let p be a wdPT and $\mathcal{D}$ a database. Further let $\mathcal{T}$ denote a global tree decomposition of p of width k, let $\vec{x}$ denote the set of free variables in p, and let μ be a mapping with $dom(\mu) \subseteq \vec{x}$. Let $\mathcal{N}$ denote a set of nodes in p with $dom(\mu) \subseteq var(\mathcal{N})$ and let $\mathcal{M} = \{M_1, \dots, M_\beta\}$ denote the set of nodes outside $\mathcal{N}$ whose parent is in $\mathcal{N}$. We have to test if there exists an extension ν of μ on the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$.

The key idea is, for all $M_i \in \mathcal{M}$, to define *critical subsets* $\mathcal{C}(N_i, M_i)$ of $\mathcal{I}(N_i, M_i)$, where $N_i \in \mathcal{N}$ is the parent of M_i. Intuitively, $\mathcal{C}(N_i, M_i)$ is defined in such a way that the existence of an extension ν of μ to the variables in M_i only depends on the values for μ on each of the critical subsets. Our FPT-algorithm relies on several crucial properties of $\mathcal{C}(N_i, M_i)$:

First of all, for each critical subset $\vec{v} \subseteq \mathcal{I}(N_i, M_i)$, we can efficiently determine the "good" (resp. "bad") value combinations, i.e., value combinations such that an extension (resp. no extension) of a mapping ν to M_i is possible, namely: $good(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists an extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$ and $bad(\vec{v}) = \{\eta \mid dom(\eta) = \vec{v}$ and there exists no extension ν of η to $var(M_i)$ with $\nu(M_i) \subseteq \mathcal{D}\}$. It can be shown that, for an arbitrary mapping μ with $\mathcal{I}(M_i, N_i) \subseteq dom(\mu)$, there exists an extension ν of μ with $var(M_i) \subseteq dom(\nu)$ and $\nu(M_i) \subseteq \mathcal{D}$ if and only if for every $\vec{v} \in \mathcal{C}(M_i, N_i)$, we have $\mu_{|\vec{v}} \in good(\vec{v})$.

Consider μ from our arbitrary instance of EXTENDSOLUTION(g-TW(k) $\cap$ SBI(c)). We have to test if there exists an extension ν of μ to the existentially quantified variables in $\mathcal{N}$, s.t. ν cannot be further extended to any of the nodes M_i in $\mathcal{M}$. In other words, such

ν has to satisfy the following two conditions: (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) for every $i \in \{1, \dots, \beta\}$, there exists a critical subset $\vec{v}_i \in \mathcal{C}(M_i, N_i)$, s.t. $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$.

Hence, our decision procedure for ExtendSolution(g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) just checks if such a combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ of critical subsets exists. We can search for such a combination by nested loops over all $\vec{v}_i \in \mathcal{C}(M_i, N_i)$ with $i \in \{1, \dots, \beta\}$. Since $\vec{v}_i \subseteq \mathcal{I}(M_i, N_i)$ and $|\mathcal{I}(M_i, N_i)| \leq c$, there are at most 2^c elements in each $\mathcal{C}(M_i, N_i)$. Moreover, β is bounded by the size of p. Hence, we have to check at most $f(p) = (2^c)^{|p|}$ combinations $(\vec{v}_1, \dots, \vec{v}_\beta)$. To prove the algorithm to be in FPT, it suffices to show that, for a given combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ of critical subsets, one can test in polynomial time if there exists an extension ν of μ with (1) $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}$ and (2) $\nu_{|\vec{v}_i} \in bad(\vec{v}_i)$ for every $i \in \{1, \dots, \beta\}$.

The second crucial property of the critical subsets $\mathcal{C}(N_i, M_i)$ with $i \in \{1, \dots, \beta\}$ is that for any combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ with $\vec{v}_i \in \mathcal{C}(N_i, M_i)$, we can transform the given global tree decomposition $\mathcal{T}$ of p of width k into a tree decomposition $\mathcal{T}^*$ of width $\leq (k+1) \cdot (c+1)$, s.t. every $\vec{v}_i$ is covered by the bag of some vertex t in $\mathcal{T}^*$. Analogously to the PTIME-membership proof in [BPS15], the tree decomposition $\mathcal{T}^*$ guarantees that the check for the existence of an extension ν of μ with the desired properties (1) and (2) is feasible in polynomial time. □

What happens if, instead of the restriction to g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$, we consider ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$? Below we show that, with this relaxation, fixed-parameter tractability is lost.

Theorem 6.9. *The problem p-*Eval*(ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) is $\mathsf{W[1]}$-complete for $k \geq 1$ and $c \geq 2$.*

Proof. Membership is shown by reduction to the $\mathsf{W[1]}$-complete problem Pos-Eval, which is the evaluation problem for FO-queries built from relational atoms using $\exists, \wedge, \vee$ [PY99] (see also [PY99] for the problem definition). The idea is to create one positive query which is a big disjunction over all possible CQs $Q_{T'}$ corresponding to subtrees of p that potentially have μ as an answer. The maximality of answers is enforced by introducing "interface relations" which, for every child node, only contain those tuples which cannot be extended to the child.

Let an instance of p-Eval(ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) be given by a wdPT $p = (T, \lambda, \vec{x})$ with root node R, a database instance $\mathcal{D}$ and a mapping μ. It is convenient to introduce some notation first: Let $\mathcal{T}'$ be the set of all subtrees T' of T containing R s.t. $var(T') \cap \vec{x} = dom(\mu)$, i.e. whose free variables are exactly the variables on which μ is defined. Also, for a subtree T' of T containing R, let the set $\mathcal{N}_{T'}$ contain exactly all nodes in $V(T) \setminus V(T')$ that are adjacent to some node in $V(T')$ (i.e., the "child nodes" of T').

Next, for every $T' \in \mathcal{T}'$, define the set of atoms

$$\phi_{T'} = \bigcup_{N \in V(T')} \lambda(N) \cup \bigcup_{N \in \mathcal{N}_{T'}} a_n(x_{i_1}, \dots, x_{i_\alpha})$$

where $\{x_{i_1}, \ldots, x_{i_\alpha}\} = \mathcal{I}(N, M)$, N is the parent of M, and $a_{T',N}$ is a new relation symbol not occurring in p. For $N \in \mathcal{N}_{T'}$ with parent M and $\mathcal{I}(N, M) = \{x_{i_1}, \ldots, x_{i_\alpha}\}$, let Q_N be the CQ $Q_N(x_{i_1}, \ldots, x_{i_\alpha}) \leftarrow \lambda(N)$ and τ_N the set of tuples

$$\tau_N = \{(\mu(x_{i_1}), \ldots, \mu(x_{i_\alpha})) \mid \mu \in Q_N(\mathcal{D})\}.$$

Finally, for a $T' \in \mathcal{T}'$ and $N \in \mathcal{N}_{T'}$, let $a^I_{T',N}$ be the set of atoms

$$a^I_{T',N} = \{a_{T',N}(\tau) \mid \tau \in (|dom(\mathcal{D})|^\alpha) \setminus \tau_N)\}.$$

We can now define a positive query Q, a database instance $\mathcal{D}'$ and a tuple t as follows:

$$\mathcal{D}' = \mathcal{D} \cup \bigcup_{T' \in \mathcal{T}'} \bigcup_{N \in \mathcal{N}_{T'}} a^I_{T',N},$$

$$Q = \bigvee_{T' \in \mathcal{T}'} \exists \vec{y}_{T'} \bigwedge_{\beta \in \phi_{T'}} \beta$$

where $\vec{y}_{T'} = (\bigcup_{N \in (V(T') \cup \mathcal{N}_{T'})} var(\lambda(N)) \setminus dom(\mu)$, and

$$t = (\mu(x_{\gamma_1}), \ldots, \mu(x_{\gamma_k}))$$

where $\{x_{\gamma_1}, \ldots, x_{\gamma_k}\} = dom(\mu)$.

The above reduction indeed is an fpt-reduction (in fact, the size of Q is indeed in $\mathcal{O}(2^{|p|})$). Moreover, it follows by the construction that $\mu \in p(\mathcal{D})$ if and only if $t \in Q(\mathcal{D}')$, which proves membership.

Hardness is shown by an fpt-reduction from p-CLIQUE. Given a graph $G = (V, E)$ and $d \in \mathbb{N}$, we construct a wdPT p and a database $\mathcal{D}$ with the following intuition: The root R of p contains d variables $x_1, \ldots, x_d$. Mapping the root into $\mathcal{D}$ assigns one node from V to each x_i. In addition, R contains one child for each pair of distinct variables $x_\alpha, x_\beta \in \{x_1, \ldots, x_d\}$. Each of these children can be mapped into $\mathcal{D}$ if and only if there is an edge between the nodes assigned to x_α, x_β. Thus G contains a clique of size d if and only if there exists a mapping $\mu \in p(\mathcal{D})$ that maps all children of R into $\mathcal{D}$.

Thus, let $G = (V, E)$ be a graph and $d \in \mathbb{N}$. We define a wdPT $p = (T, \lambda, \vec{x})$, a mapping μ, and a database $\mathcal{D}$ as follows:

- T consists of a root R with child nodes $N_{i,j}$ for $1 \leq i < j \leq d$,
- $\lambda(R) = \{v(x_1), \ldots, v(x_k), b(z_o)\}$,
- for $1 \leq i < j \leq d$, $\lambda(N_{i,j}) = \{e(x_i, x_j), b(z_{i,j})\}$,
- $\vec{x} = \{z_{i,j} \mid 1 \leq i < j \leq d\}$,
- $e^{\mathcal{D}} = \{(v_i, v_j) \mid \{v_i, v_j\} \in E\}, v^{\mathcal{D}} = \{v_i \mid v_i \in V\}, b^D = \{1\}$, and

- $\mu = \{(z_0, 1)\} \cup \{(z_{i,j}, 1) \mid 1 \leq i < j \leq d\}$.

It is easy to check that indeed $p \in \ell\text{-}\mathsf{TW}(1) \cap \mathsf{SBI}(2)$, the reduction is an fpt-reduction, and that G has a clique of size d if and only if $\mu \in p(\mathcal{D})$. Indeed, bindings to the variables $x_1, \ldots, x_d$ by some $\nu \in p(\mathcal{D})$ encode a selection of d nodes. Moreover, each child node checks the existence of an edge between two distinct nodes from $x_1, \ldots, x_d$. Thus the selection on $x_1, \ldots, x_d$ is a clique if and only if all child nodes of R can be mapped into $\mathcal{D}$. □

We now consider another relaxation of the $g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ restriction from Theorem 6.8 by considering wdPTs in $g\text{-}\mathsf{TW}(k)$ but without any bound on the interfaces.

Theorem 6.10. *The problem p-*Eval*($g\text{-}\mathsf{TW}(k)$) is $\mathsf{W}[2]$-hard for $k \geq 1$.*

Proof. The proof is by reduction from p-Dominating Set. Thus, let $G = (V, E)$ be a graph and $d \in \mathbb{N}$. The idea of the constructed wdPT p and database $\mathcal{D}$ is to have variables $x_1, \ldots, x_d$ in the root R of p such that a mapping of R into $\mathcal{D}$ assigns one node from V to each x_i. Observe that $S \subseteq V$ is a dominating set if and only if there does not exist some $u \in V \setminus S$ that is not adjacent to any node in S. This is tested in the single child node of R: It contains an additional variable x_0 which also encodes nodes in V. Now the child node is mapped into $\mathcal{D}$ if and only if there exists a value for x_0 that differs from those of all x_i's, and there does not exist an edge between the nodes mapped to x_0 and any of the x_i's. I.e., there exists a solution that only maps the root into $\mathcal{D}$ if and only if G contains a dominating set of size d.

So we construct an instance of Eval($g\text{-}\mathsf{TW}(k)$) consisting of a wdPT $p = (T, \lambda, \vec{x})$, a mapping μ, and a database $\mathcal{D}$ as follows:

- T consists of a root node R with a single child node N,
- $\lambda(R) = \{v(x_1), \ldots, v(x_d), b(z_0)\}$,
- $\lambda(N) = \{\mathit{diff}(x_0, x_i), e^c(x_0, x_i) \mid 1 \leq i \leq d\} \cup \{b(z_1)\}$,
- $\vec{x} = \{z_0, z_1\}$,
- $v^{\mathcal{D}} = \{v_i \mid v_i \in V\}$, $\mathit{diff}^{\mathcal{D}} = \{(v_i, v_j) \mid v_i, v_j \in V \text{ with } v_i \neq v_j\}$, $(e^c)^{\mathcal{D}} = \{(v_i, v_j) \mid \{v_i, v_j\} \in ((V \times V) \setminus E)\}$, $b^{\mathcal{D}} = \{1\}$, and
- $\mu = \{(z_0, 1)\}$.

It is easy to check that the reduction is indeed an fpt-reduction and that $p \in g\text{-}\mathsf{TW}(1)$. It remains to show that $\mu \in p(\mathcal{D})$ if and only if G has a dominating set of size d. We show the two directions separately.

Assume that $\mu \in p(\mathcal{D})$. I.e., there exists some maximal homomorphism φ from p to $\mathcal{D}$ with $\mu \subseteq \varphi$ and s.t. $\varphi \in Q_R(\mathcal{D})$ (where $Q_R = Q_{T'}$ for the subtree containing R only). Thus, for all values $\alpha \in \mathit{dom}(\mathcal{D})$ we have that $(\alpha, \varphi(x_i)) \notin \mathit{diff}^{\mathcal{D}}$ or $(\alpha, \varphi(x_i)) \notin (e^c)^{\mathcal{D}}$ for some $1 \leq i \leq k$ since φ can clearly be extended to z_1 otherwise. Consider the set S of nodes defined as $S := \{\varphi(x_i) \mid 1 \leq i \leq k\}$ (identifying nodes in V with the respective elements in $\mathit{dom}(\mathcal{D})$). It can now be checked that S is a dominating set: Indeed, for an arbitrary node $v \in V$, either $v \in S$ (in case that $(v, \varphi(x_i)) \notin \mathit{diff}^{\mathcal{D}}$ for some i) or for some $v' \in S$ there is an edge $\{v, v'\} \in E$ (if $(v, \varphi(x_i) \notin (e^c)^{\mathcal{D}}$ for some i). In case that $|S| < k$, adding arbitrary nodes from V to S s.t. $|S| = k$ gives a dominating set of size k.

Next assume that G has a dominating set $S = \{s_1, \ldots, s_k\}$ of size k. Then the mapping φ defined as $\{(x_i, s_i) \mid 1 \leq i \leq k\} \cup \mu$ is a maximal homomorphism from p to $\mathcal{D}$. Assume to the contrary that this is not the case, and let φ' be an extension of φ. Then the node $\varphi(x_0)$ is neither in S (because of the atoms $\mathit{diff}(x_0, x_i)$) nor adjacent to any node in S (because of the atoms $e^c(x_0, x_i)$), which gives the desired contradiction. ☐

It was shown in [BPS15] that the problem Max-Eval(g-$\mathsf{TW}(k)$) is in PTIME. In other words, for wdPTs with globally bounded treewidth, there is no need to also restrict the interface. Moreover, as recalled above, restricting wdPTs to ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$ also yields a restriction of the global treewidth. The only case remaining is therefore the restriction to ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$.

Proposition 6.11. *The problem p-Max-Eval(ℓ-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) is $\mathsf{W[1]}$-hard for $k \geq 1$ and $c \geq 2$.*

Proof. The reduction used to prove the hardness in the proof of Theorem 6.9 also proves this case, since clearly $\mu \in p(\mathcal{D})$ if and only if $\mu \in p_m(\mathcal{D})$. ☐

6.3 Classical and Parameterized Complexity of Enumeration

We now turn our attention back to the enumeration problem. Analogously to the evaluation problem in Section 6.2, we will study four variants of the enumeration problem, namely enumerating *all* vs. the *maximal* solutions and *parameterized* vs. *non-parameterized* problems.

Enum($\mathcal{C}$) / Max-Enum($\mathcal{C}$)	
INSTANCE:	Query: wdPT $p \in \mathcal{C}$, Data: database $\mathcal{D}$.
Output:	Is $p(\mathcal{D})$ / $p_m(\mathcal{D})$?

p-Enum($\mathcal{C}$) / p-Max-Enum($\mathcal{C}$)	
INSTANCE:	wdPT $p \in \mathcal{C}$, database $\mathcal{D}$.
PARAMETER:	$\lvert p \rvert$
OUTPUT:	Is $p(\mathcal{D})$ / $p_m(\mathcal{D})$?

	ℓ-TW(k) $\cap$BI(c)	g-TW(k) $\cap$SBI(c)	ℓ-TW(k) $\cap$SBI(c)	g-TW(k)
p-Eval$(\mathcal{C})$	PTIME*	FPT Theorem 6.8	W[1]-complete Theorem 6.9	W[2]-hard Theorem 6.10
p-Max-Eval$(\mathcal{C})$	PTIME*	PTIME*	W[1]-hard Theorem 6.11	PTIME*
Enum$(\mathcal{C})$	SDelayP Corollary 6.14	*open*	not OutputP Corollary 6.17	not OutputP Corollary 6.19
Max-Enum$(\mathcal{C})$	not OutputP Corollary 6.22	not OutputP Corollary 6.22	not OutputP Corollary 6.22	not OutputP Corollary 6.22
p-Enum$(\mathcal{C})$	SDelayP Corollary 6.14	DelayFPT Corollary 6.14	not OutputFPT Corollary 6.17	not OutputFPT Corollary 6.17
p-Max-Enum$(\mathcal{C})$	DelayFPT Theorem 6.24	DelayFPT Theorem 6.24	not OutputFPT Proposition 6.25	DelayFPT Theorem 6.24

Table 6.1: Summary of the main results on evaluation and enumeration of wdPTs. Entries marked with a * were shown in [BPS15]

As we have discussed in Section 6.1, for wdPTs, analogously to the evaluation problem, additional restrictions are necessary in order to achieve tractability. However, for enumeration we get a more diverse picture than for evaluation: When we are interested in all solutions, the additional restrictions used for evaluation are sufficient also for enumeration, while this is not the case if we are only interested in the maximal solutions. It will turn out that the techniques required to analyze the enumeration of *all* vs. the *maximal* solutions differ significantly. We thus treat the enumeration and the max-enumeration problems in separate subsections below. Table 6.1 gives a summary of the results for both the enumeration as well as the evaluation complexities derived in this chapter.

6.3.1 Enumeration

For our study of the enumeration problem, the decision problem ExtendSolution$(\mathcal{C})$, which we introduced in Section 6.2, again plays an important role. In fact, this method, which is sometimes also called backtrack method for enumeration problems, has been used frequently [MS16, RT75, CH97, AF96]. We will give an in-depth explanation adapted to the case of wdPTs.

Lemma 6.12. *Let $\mathcal{C}$ be a class of pattern trees, $p = (T, \lambda, \vec{x}) \in \mathcal{C}$, and $\mathcal{D}$ a database. Assume that there is a computable function f such that for every partial mapping μ and every subset $\vec{x}'$ of $\vec{x}$, the problem* ExtendSolution$(\mathcal{C})$ *can be decided in $\mathcal{O}(f(|p|, |\mathcal{D}|))$.*

Then there exists an algorithm enumerating $p(\mathcal{D})$ with delay(i) in $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$ for $i \geq 0$.

Proof. The general idea is to create solutions by iteratively testing if certain variable bindings can be extended to solutions. Observe that for CQs, it would suffice to test (in nested loops) for each variable if it can be bound to a certain domain element. In contrast, for wdPTs, we now have to consider an additional option, namely not binding a variable at all. Thus, we look for extensions (= solutions) that bind some variables to specific domain values (expressed by μ) and leave some variables unbound (expressed by $\vec{x}'$).

So let $p \in \mathcal{C}$ be a wdPT with free variables $\vec{x} = \{x_1, \ldots, x_n\}$, $\mathcal{D}$ a database, and denote Algorithm 3 by $\mathcal{A}$.

Algorithm 3 Enumerate all solutions of $p(\mathcal{D})$ for $p = (T, \lambda, \{x_1, \ldots, x_n\})$.

```
 1: Initialize a global vector $\vec{v} \in (dom(\mathcal{D}) \cup \{\bot\})^n$
 2: $V = dom(\mathcal{D}) \cup \{\bot\}$
 3: procedure Enumerate
 4:     Ext-Sol(1)
 5: end procedure
 6:
 7: procedure Ext-Sol($i$)
 8:     for all $\nu \in V$ do
 9:         $\vec{v}(i) = \nu$
10:         $\mu = \{(x_j, a) \mid \vec{v}(j) = a, a \neq \bot, 1 \leq j \leq i\}$
11:         $\vec{x}' = \{x_j \mid \vec{v}(j) = \bot, 1 \leq j \leq i\}$
12:         if SP-Check($p, \mathcal{D}, \mu, \vec{x}'$) then
13:             if $i = n$ then
14:                 Output $\mu$
15:             else
16:                 Ext-Sol($i + 1$)
17:             end if
18:         end if
19:     end for
20: end procedure
```

Assume that for a partial mapping μ and a subset $\vec{x}'$ of the free variables of p, the procedure SP-Check$(p, \mathcal{D}, \mu, \vec{x}')$ decides whether $(p, \mathcal{D}, \mu, \vec{x}') \in$ ExtendSolution$(\mathcal{C})$ in time $\mathcal{O}(f(|p|, |\mathcal{D}|))$ for some computable function f. We need to show that $\mathcal{A}$ enumerates $p(\mathcal{D})$ with a delay polynomial in $f(|p|, |\mathcal{D}|), |p|$ and $|\mathcal{D}|$. First consider a possible first solution that is in the output. On the very first call of Ext-Sol(1), assume that there is no $\nu \in V$ such that SP-Check$(p, \mathcal{D}, \{(x_1, \nu)\}, \emptyset)$ or SP-Check$(p, \mathcal{D}, \emptyset, \{x_1\})$ evaluates to true. By the definition of ExtendSolution$(\mathcal{C})$, $p(\mathcal{D}) = \emptyset$ and $\mathcal{A}$ stops in $\mathcal{O}(|V| \cdot f(|p|, |\mathcal{D}|)) =$

$\mathcal{O}(|\mathcal{D}| \cdot f(|p|, |\mathcal{D}|)$ with no output. So assume that there is some $\nu \in V$ with SP-Check$(p, \mathcal{D}, \{(x_1, \nu)\}, \emptyset)$= true (the case SP-Check$(p, \mathcal{D}, \emptyset, \{x_1\})$= true can be shown analogously). Hence the partial solution $\{(x_1, \nu)\}$ can be extended to some solution in $p(\mathcal{D})$, and $\mathcal{A}$ constructs such a solution, where for each variable x_i, $1 \leq i \leq n$, we need at most $|V| = \mathcal{O}(|\mathcal{D}|)$ many steps to to find an element ν' such that either the current partial solution can be extended by $\{(x_i, \nu')\}$, or to determine that the current partial solution can be extended to some solution $\mu \in p(\mathcal{D})$ with $x_i \notin dom(\mu)$. Thus we can output a first solution in $n \cdot |V| \cdot \mathcal{O}(f(|p|, |\mathcal{D}|)) = \mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$ many steps. Next assume that the functions μ_1 and μ_2 are output consecutively by $\mathcal{A}$, and let l be minimal such that $\mu_1(x_l) \neq \mu_2(x_l)$. This means that for all k with $l \leq k \leq n$, algorithm $\mathcal{A}$ makes at most $|V| = \mathcal{O}(|\mathcal{D}|)$ many failed SP-Check calls at Ext-Sol(k) before $\mathcal{A}$ finds some μ and some $\vec{x}'$ such that SP-Check$(p, \mathcal{D}, \mu, \vec{x}')$ returns true. As with the first output, we need at most $n \cdot |V| \cdot \mathcal{O}(f(|p|, |\mathcal{D}|))$ many steps in the algorithm to extend μ to some μ' (in this case μ_2 that is output, thus the delay between the outputs μ_1 and μ_2 is indeed $\mathcal{O}(f(|p|, |\mathcal{D}|) \cdot |\mathcal{D}| \cdot |p|)$.

Next we need to show that $\mathcal{A}$ produces no duplicates. Assume that μ_1 is in the output. For any another output μ_2, the algorithm chooses at least one new value $\nu \in V$ at some for-loop for some procedure call Ext-Sol(j), $1 \leq j \leq n$, therefore $\mu_1(x_j) \neq \mu_2(x_j)$.

Now let $\mathcal{P}$ be the output of $\mathcal{A}$. To show that $\mathcal{P} = p(\mathcal{D})$, let $\mu \in \mathcal{P}$. Then at a procedure call Ext-Sol(n), SP-Check$(p, \mathcal{D}, \mu, \vec{x}')$ = true for some $\vec{x}'$. Since $dom(\mu) \cup \vec{x}' = \vec{x}$, there is no $\mu' \in p(\mathcal{D})$ such that $\mu \subset \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$. Thus $\mu \in p(\mathcal{D})$. Now assume that $\mu \in p(\mathcal{D})$ with $\vec{x}' = \vec{x} \setminus dom(\mu)$. Then $\mu \in \mathcal{P}$, as for any $\mu' \subseteq \mu$ and any $\vec{x}'' \subseteq \vec{x}$ we have SP-Check$(p, \mathcal{D}, \mu', \vec{x}'')$ = true, and hence μ will be output at some point in the execution of the algorithm. □

To make use of this lemma, we identify classes of wdPTs that meet the requirements:

Proposition 6.13. *The following complexity results hold for* ExtendSolution$(\mathcal{C})$*:*

1. *Let* $c \geq 1$ *and* $\mathcal{C}$ *be a class of CQs for which* CQ-Eval$(\mathcal{C})$ *is in* PTIME*. Then* ExtendSolution$(\ell\text{-}\mathcal{C} \cap \mathsf{BI}(c))$ *is in* PTIME*.*
2. ExtendSolution$(g\text{-}\mathsf{TW}(k))$ *is* NP*-complete for every* $k \geq 1$*.*
3. *Let* $k, c \geq 1$*. Then* ExtendSolution$(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ *parameterized by* $|p|$ *is in* FPT*.*

Proof. We only need to show (1): Indeed (2) follows immediately from [BPS15]. For (3) we note that in the previous section, Theorem 6.8 was stated for p-Eval$(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$. One can actually show the stronger result of FPT membership for the parameterized version of ExtendSolution$(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$.

So to show part 1 of the proposition, we describe the polynomial time algorithm for deciding the problem ExtendSolution$(\ell\text{-}\mathcal{C} \cap \mathsf{BI}(c))$, given that $\mathcal{C}$ is a class of CQs for which CQ-EVAL($\mathcal{C}$) is in PTIME and $c \geq 1$ a positive integer. The algorithm is a slight adaptation of the algorithm in the proof of [BPS15, Theorem 6].

Thus assume that we are given a wdPT $p = (T, \lambda, \vec{x})$, a database $\mathcal{D}$, a partial mapping $\mu : \mathbf{X} \to \mathbf{U}$ and a set $\vec{x}' \subseteq \vec{x}$.

1. As a preprocessing step, first of all check if there exists some subtree T' of T s.t. $dom(\mu) \subseteq \text{fvars}(T')$ and $\text{fvars}(T') \cap \vec{x}' = \emptyset$. If this is not the case, then clearly $\mu \notin p(\mathcal{D})$.

2. Let T' be the *minimal* subtree of T s.t. $dom(\mu) \subseteq \text{fvars}(T')$, and T'' be the *maximal* subtree of T s.t. $dom(\mu) \subseteq \text{fvars}(T'')$ and $\text{fvars}(T'') \cap \vec{x}' = \emptyset$.

   Observe that because of p being well-designed, T' and T'' are uniquely identified.

3. The goal of the algorithm is to construct an acyclic Boolean CQ Q and a database $\mathcal{D}'$, s.t. Q contains one atom for every node in T', and that $Q(\mathcal{D}') \neq \emptyset$ if and only if there exists some $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$.

   Towards this goal, first substitute in p every $x \in dom(\mu)$ by $\mu(x)$. With a slight abuse of notation, call this pattern tree again p.

4. For every node $N \in V(T'')$, introduce a new relation symbol R_N of arity r_N, where $r_N = |\mathcal{I}(N)|$.

5. Let Q be the acyclic Boolean CQ

$$Q() \leftarrow \bigwedge_{N \in V(T')} R_N(\vec{v}_N)$$

   Thus, Q contains exactly one atom for each node in T' (and not in T'').

6. Next, we have to define the database $\mathcal{D}'$.

   Again, this requires to settle some notation first: Consider a node $N \in V(T'')$ and a tuple $\alpha \in dom(\mathcal{D})^{r_N}$ (where again $r_N = |\mathcal{I}(N)|$). Then we may interpret α as a mapping $\vec{v}_n \to dom(\mathcal{D})$ defined as $\{(\vec{v}_n(i), \alpha(i)) \mid 1 \leq i \leq r_N\}$, where $\vec{v}_n(i)$ and $\alpha(i)$ refer to the i^{th} element of $\vec{v}_n$ and α, respectively. Thus, to simplify notation, in the following we may identify tuples α with the corresponding mappings.

7. Towards defining database $\mathcal{D}'$, we define an intermediate database $\mathcal{D}''$ as follows:

   - For every $N \in V(T'')$, and every tuple $\alpha \in dom(\mathcal{D})^{r_N}$, let $R_N^{\mathcal{D}''}$ contain the tuple α if there exists some mapping ν s.t. $\mu_{|\,\text{fvars}(\lambda(N))} \cup \alpha \cup \nu(\vec{v}_i)) \in R_i^{\mathcal{D}}$ for every atom $R_i(\vec{v}_i) \in \lambda(N)$. I.e., if the mapping $\mu_{|\,\text{fvars}(\lambda(N))} \cup \alpha$ can be extended to a solution to the CQ built from the atoms in $\lambda(N)$.

- If we would now evaluate Q already on $\mathcal{D}''$, we had $Q(\mathcal{D}'') \neq \emptyset$ whenever μ is contained in some solution in $p(\mathcal{D})$, i.e. if either μ is indeed some solution to p over $\mathcal{D}$, or if it can be extended to some solution. Thus, in order to test whether μ is contained in some μ' such that $\mu' \in p(\mathcal{D})$ and $dom(\mu') \cap (\vec{x}') = \emptyset$, we have to restrict the number of atoms in $\mathcal{D}''$.
- To do so, for all atoms of the form $R_N(\alpha)$ for all nodes $N \in \mathcal{N} = (V(T'') \setminus V(T')) \cup L$ where L are all leaf nodes of T', we compute a label $l(R_N(\alpha)) \in \{\mathit{safe}, \mathit{unsafe}\}$ in a bottom-up traversal of T'' as follows:
- In a leaf node N of T'', we have $l(R_N(\alpha)) = \mathit{unsafe}$ if there exists some child node N_i of N and an extension φ of $\mu \cup \alpha$ s.t. for all $R_i(\vec{v}_i) \in \lambda(N_i)$ we have $(\varphi_{|var(\lambda(N_i))}(\vec{v}_i)) \in R_i^{\mathcal{D}}$. Otherwise, i.e. if $\mu \cup \alpha$ cannot be extended to any child node N_i of N, then $l(R_N(\alpha)) = \mathit{safe}$
- For a non-leaf node $N \in \mathcal{N}$ such that for all child nodes $N_1, \dots, N_k$ of N all labels $l(R_{N_i}(\alpha_j))$ have already been computed, the labels are defined as follows:

  $l(R_N(\alpha)) = \mathit{safe}$ if for every child node N_i of N, either (a) for all tuples $\beta \in R_{N_i}^{\mathcal{D}''}$ we have that α and β are not compatible (i.e., they do not agree on the shared variables), or (b) if there exist compatible tuples, then there exists at least one compatible tuple $\beta \in R_{N_i}^{\mathcal{D}''}$ with $l(R_{N_i}(\beta)) = \mathit{safe}$.

  $l(R_N(\alpha)) = \mathit{unsafe}$ otherwise, i.e. if there exists some child node N_i of N s.t. there exists at least one compatible tuple $\beta \in R_{N_i}^{\mathcal{D}''}$ and for all such compatible tuples $\beta_j \in R_{N_i}^{\mathcal{D}''}$ we have $R_{N_i}(\beta_j) = \mathit{unsafe}$.

8. We define $\mathcal{D}'$ from $\mathcal{D}''$ as follows:
   - For every non-leaf node N of T', $R_N^{\mathcal{D}'}$ contains all tuples α from $R_N^{\mathcal{D}''}$.
   - For every leaf node N of T', $R_N^{\mathcal{D}'}$ contains only those tuples α from $R_N^{\mathcal{D}''}$ s.t. $l(R_N(\alpha)) = \mathit{safe}$.

We now have that $q(\mathcal{D}') \neq \emptyset$ if and only if there exists a $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$.

Clearly, the above construction is feasible in polynomial time, and also $q(\mathcal{D}') \neq \emptyset$ can be decided in polynomial time, since q is an acyclic CQ. $\square$

Now the following corollary follows immediately from the previous results.

Corollary 6.14. *Let $k, c \geq 1$.*

- *Let $\mathcal{C}$ be a class of CQs for which* CQ-Eval($\mathcal{C}$) *is in* PTIME. *Then* Enum(ℓ-$\mathcal{C}$$\cap$BI($c$)) *is in* SDelayP.
- *The problem* p-Enum(g-TW(k) $\cap$ SBI(c)) *is in* DelayFPT.

We now move to negative results for the enumeration problem. As an important tool and an interesting result in its own right, we establish a close relationship between enumeration and the parameterized complexity of evaluation. We introduce some terminology first:

Definition 6.15. A class $\mathcal{C}$ of wdPTs is *robust* if for every wdPT $p = (T, \lambda, \vec{x}) \in \mathcal{C}$ the following two conditions hold:

1. For every $\mathcal{N} = \{N_1, \dots, N_m\} \subseteq V(T)$ the wdPT $(T, \lambda_{\mathcal{N}}, \vec{z})$ is in $\mathcal{C}$, where $\vec{z} = \{z_1 \dots, z_m\}$ is a set of new variables, $\lambda_{\mathcal{N}}(N) = \lambda(N)$ for all $N \in V(T) \setminus \mathcal{N}$ and $\lambda_{\mathcal{N}}(N_i) = \lambda(N_i) \cup \{b(z_i)\}$ for $1 \leq i \leq m$ and some new relation symbol b.
2. For every variable $x \in \mathit{var}(p)$, let p' be the wdPT retrieved from p by replacing every occurrence of x by the same constant c. Then $p' \in \mathcal{C}$.

The notion of "robust" classes of wdPTs is important in the following theorem, to make sure that wdPTs do not fall out of their class when certain transformations are performed.

Theorem 6.16. *Let $\mathcal{C}$ be a robust class of wdPTs. If p-*E*NUM*$(\mathcal{C})$ *is in* OutputFPT*, then p-*E*VAL*$(\mathcal{C})$ *is in* FPT.

Proof. Assume that we have given a pattern tree $p = (T, \lambda, \vec{x}) \in \mathcal{C}$ with $\vec{x} = \{x_1, \dots, x_n\}$, a database $\mathcal{D}$ and a mapping $\mu = \{(x_1, a_1), \dots, (x_r, a_r)\}$. We will use the fact that p-ENUM$(\mathcal{C})\in$ OutputFPT to check whether $\mu \in p(D)$ in time $\mathcal{O}(f(|p|) \cdot |D|^m)$ for some positive integer m.
Let T' (T'') be the minimal (maximal) subtree of T that contains $\{x_1, \dots, x_r\}$ and does not contain $\{x_{r+1}, \dots, x_n\}$. If no such trees exist, then $\mu \notin p(\mathcal{D})$. Further let $B(T') \subseteq T'$ be the set of all pattern tree nodes with a child in $T \setminus T'$ and let $B(T'') \subseteq (T \setminus T'')$ be the set of all pattern tree nodes with a parent in T''. Define a pattern tree $q = (T_q, \lambda_q, \vec{x}_q)$ as follows:

- $T_q = T$
- Let $A \in \lambda(N)$ be an atom for some $N \in T$. Then denote by A_μ the atom where every occurrence of some $X_i \in \{x_1, \dots, x_r\}$ is replaced by a_i. Define λ_q as follows:

$$\lambda_q(N) = \begin{cases} \{A_\mu \mid A \in \lambda(N)\} & : N \in T \setminus \{B(T') \cup B(T'')\}, \\ \{A_\mu \mid A \in \lambda(N)\} \cup \{b(z_N)\} & : N \in B(T') \\ \{A_\mu \mid A \in \lambda(N)\} \cup \{b(\bar{z}_N)\} & : N \in B(T''), \end{cases}$$

  where for all $N \in T$ the variables z_N and $\bar{z}_N$ are new variables and b is a new unary relation symbol. Denote by Φ the set of all such z_N and by Ψ the set of all such $\bar{z}_N$.
- $\vec{x}_q = \Phi \cup \Psi$.

Further extend the database $\mathcal{D}$ to a database $\mathcal{D}_q$ by adding the relation $b^{\mathcal{D}_q} = \{1\}$. There are at most $2^{|q|}$ mappings in $q(\mathcal{D}_q)$ and the size of every mapping is bounded by $|q|$. Therefore, since $\mathcal{C}$ is robust and thus $q \in \mathcal{D}$, we can enumerate $q(\mathcal{D}_q)$ in time $\mathcal{O}((|\mathcal{D}_q| + |q| + |q|2^{|q|})^m f(|q|))$ for some function f and some positive integer m.

Claim. *Let $\nu := \{(z,1) | z \in \Phi\}$. Then $\mu \in p(\mathcal{D})$ if and only if $\nu \in q(\mathcal{D}_q)$.*

Proof of the Claim. We have $\mu \in p(\mathcal{D})$ if and only if there is a variable assignment of $var(T'')$, such that $\lambda_q(N)$ is true for all $N \in B(T')$ and this variable assignment can not be extended to $var(T)$ such that $\lambda_q(N')$ is true for any N' in $B(T'')$. This is equivalent to $\nu \in q(\mathcal{D}_q)$. □

So to decide whether $\mu \in p(\mathcal{D})$, it suffices to decide whether $\nu \in q(\mathcal{D}_q)$. This can be done in time $\mathcal{O}((|\mathcal{D}_q| + |q| + |q|2^{|q|})^m f(|q|)) = \mathcal{O}((|\mathcal{D}_p| + |p| + |p|2^{|p|})^m f(|p|)) = \mathcal{O}(|\mathcal{D}_p|^m \cdot (|p|^2 2^{|p|})^m f(|p|))$. □

As already mentioned in Section 6.1, we are now ready to give the proof of Proposition 6.7:

Proof of Proposition 6.7. Let $\mathcal{C}^*$ be a class of conjunctive queries with free-connex bounded treewidth. It is easy to see that in the proofs for Theorem 6.9 and Theorem 6.10 , the wdPTs used for the reductions in the hardness results are in the classes $l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)$ respectively $g\text{-}\mathcal{C}^*$ for some $c > 0$. So by assuming that $\mathsf{FPT} \neq \mathsf{W[1]}$, it follows that neither p-Enum($l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)$) nor p-Enum($g\text{-}\mathcal{C}^*$) is in OutputFPT by Theorem 6.16. For the sake of a contradiction, assume that for every pattern tree $p \in l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)$ and database $\mathcal{D}$, there is positive integer m and function f such that $p(\mathcal{D})$ can be enumerated with $delay(0)$ in $\mathcal{O}(|D|^m \cdot f(|p|))$ and $delay(i)$ in $\mathcal{O}(g(|p|))$ for some function g and $i \geq 1$ (the case $p \in g\text{-}\mathcal{C}^*$ can be shown analogously). Then we can output $p(\mathcal{D})$ in time $\mathcal{O}(|D|^m f(|p|) + |p(\mathcal{D})| g(|p|)) = \mathcal{O}((|p| + |\mathcal{D}| + |p(\mathcal{D})|)^m (f+g)(|p|))$, which is a contradiction to the fact that p-Enum($l\text{-}\mathcal{C}^* \cap \mathsf{SBI}(c)$) is not in OutputFPT. □

Exploiting this relationship between the evaluation and the enumeration problem, the next results are immediate consequences of the W[1]- and W[2]-hardness, respectively, shown in the previous section and the fact that all the classes mentioned in this chapter are robust.

Corollary 6.17. *If* FPT $\neq$ W[1]*, then the following holds:*

- *The problem p-*Enum*(ℓ-TW(k) $\cap$ SBI(c)) is not in* OutputFPT *for $k \geq 1$ and $c \geq 2$. Thus also the problem* Enum*(ℓ-TW(k) $\cap$ SBI(c)) is not in* OutputP *for $k \geq 1$ and $c \geq 2$.*
- *The problem p-*Enum*(g-TW(k)) is not in* OutputFPT *for $k \geq 1$.*

It follows immediately from Theorem 6.16 (in combination with Theorem 6.10) that $\textsc{Enum}(g\text{-}\mathsf{TW}(k))$ is not in $\mathsf{OutputP}$ if $\mathsf{FPT} \neq \mathsf{W}[1]$. However, we can show the same result under an even stronger assumption, namely assuming that $\mathsf{PTIME} \neq \mathsf{NP}$, by making use of the following NP-hardness result.

Proposition 6.18. *The following problem is* NP*-hard for every* $k \geq 1$*: Given a wdPT* $p \in g\text{-}\mathsf{TW}(k)$ *and a database* $\mathcal{D}$*, decide whether* $|p(\mathcal{D})| = 2$*.*

Proof. The proof is by reduction from the well-known NP-complete problem Dominating Set, using the same construction already described in the proof of Theorem 6.10. W.l.o.g. we consider only graphs $G = (V, E)$ that contain at least one node N that is not already a dominating set. Then we clearly always have $\mu' = \{(z_0, 1), (z_1, 1)\} \in p(\mathcal{D})$ (just map all variables $x_1, \dots, x_d$ to N). However, it still remains that $\mu = \{(z_0, 1)\} \in p(\mathcal{D})$ if and only if G contains a dominating set of size d. We thus get $|p(\mathcal{D})| = 2$ if and only if G contains such a dominating set. $\square$

The next result follows immediately. Indeed, an algorithm solving $\textsc{Enum}(g\text{-}\mathsf{TW}(k))$ in polynomial time w.r.t. the size of the input plus the output would provide a polynomial time decision procedure for Dominating Set, a problem well-known to be NP-hard.

Corollary 6.19. *If* $\mathsf{PTIME} \neq \mathsf{NP}$*, then* $\textsc{Enum}(g\text{-}\mathsf{TW}(k))$ *is not in* $\mathsf{OutputP}$ *for every* $k \geq 1$*.*

6.3.2 Max-Enumeration

We next consider the problem of enumerating only the *maximal solutions* of a wdPT. In fact, we show that for none of the classes of wdPTs considered in this work, the problem $\textsc{Max-Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c))$ is in $\mathsf{OutputP}$. After establishing this result, we therefore turn towards the parameterized problem, where for all but one of the classes we can show a positive result, namely $\mathsf{DelayFPT}$ membership.

For the negative result on the non-parameterized problem, we show intractability for $\textsc{Max-Enum}(\ell\text{-}\mathsf{TW}(k) \cap g\text{-}\mathsf{TW}(k') \cap \mathsf{BI}(c))$, which immediately implies all other results for non-parameterized maximal enumeration. Again, we do this via an intractability result for a suitable decision problem.

Proposition 6.20. *The following problem is* NP*-hard for every* $k, k', c \geq 1$*: Given a wdPT* $p \in \ell\text{-}\mathsf{TW}(k) \cap g\text{-}\mathsf{TW}(k') \cap \mathsf{BI}(c)$*, a database* $\mathcal{D}$*, and an integer* $s \geq 1$ *encoded in unary, decide if* $|p_m(\mathcal{D})| > s$*.*

Proof. The proof is by reduction from the well-known NP-complete problem 3-SAT: Let ϕ be a Boolean formula in 3-CNF, i.e. $\phi = \bigwedge_{i=1}^{m}(l_{i,1}, l_{i,2}, l_{i,3})$ where all $l_{i,j}$ ($1 \leq i \leq m$, $j \in \{1, 2, 3\}$) are literals over the variables $x_1, \dots, x_n$.

We define $p = (T, \lambda, \vec{z})$, $\mathcal{D}$, and s as follows. For an example of this construction, see Figure 6.3.

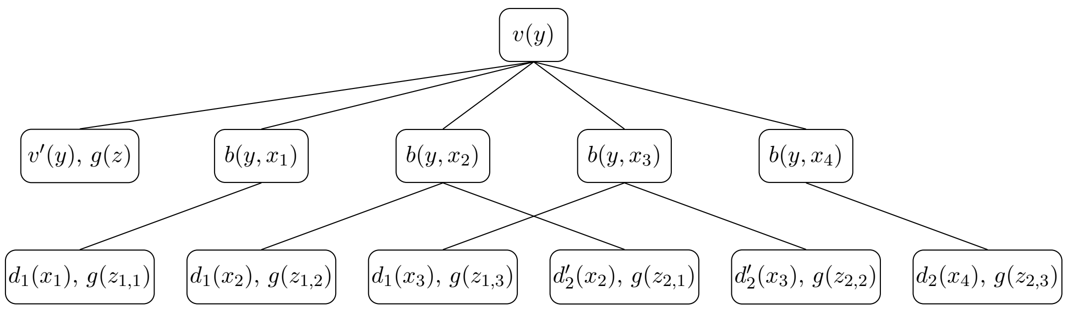


Figure 6.3: Example of a tree structure of the wdPT p constructed from the 3-CNF formula $\phi = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_2 \wedge \neg x_3 \wedge x_4)$ by the reduction given the proof of Proposition 6.20

- $V(T) = \{R\} \cup \{N_i \mid 0 \leq i \leq n\} \cup \{N'_{\alpha,\beta} \mid 1 \leq \alpha \leq m, 1 \leq \beta \leq 3\}$,
- $E(T) = \{\{R, N_i\} \mid 0 \leq i \leq n\} \cup \{\{N_i, N'_{\alpha,\beta}\} \mid l_{\alpha,\beta} = x_i$ or $l_{\alpha,\beta} = \neg x_i\}$.
- I.e., T contains a root R with $n+1$ children (intuitively, one child N_i for every variable x_i, and one additional child). Furthermore, each of the n children of R representing a variable x_i contains one distinct child for every occurrence $l_{\alpha,\beta}$ of x_i in ϕ.
- $\lambda(R) = \{v(y)\}$,
- $\lambda(N_0) = \{v'(y), g(z)\}$,
- $\lambda(N_i) = \{b(y, x_i)\}$ for $1 \leq i \leq n$,
- $\lambda(N'_{\alpha,\beta}) = \{l^*_{\alpha,\beta}, g(z_{\alpha,\beta})\}$ for $1 \leq \alpha \leq m$, $1 \leq \beta \leq 3$ where
  $l^*_{\alpha,\beta} = d_\alpha(x_i)$ if $l_{\alpha,\beta} = x_i$ and $l^*_{\alpha,\beta} = d'_\alpha(x_i)$ if $l_{\alpha,\beta} = \neg x_i$, respectively,
- $g^{\mathcal{D}} = \{1\}, v^{\mathcal{D}} = \{a_i \mid 0 \leq i \leq m\}, (v')^{\mathcal{D}} = \{a_i \mid 1 \leq i \leq m\}$,
  $b^{\mathcal{D}} = \{(a_0, 0), (a_0, 1)\} \cup \{(a_i, a_i) \mid 1 \leq i \leq m\}$,
- For all $1 \leq i \leq m : d_i^{\mathcal{D}} = \{1\} \cup \{a_j \leq 1 \leq i \neq j \leq m\}$,
  $(d'_i)^{\mathcal{D}} = \{0\} \cup \{a_j \leq 1 \leq i \neq j \leq m\}$,
- $\vec{z} = \{z\} \cup \{z_{\alpha,\beta} \mid 1 \leq \alpha \leq m, 1 \leq \beta \leq 3\}$, and
- $s = m$.

The local treewidth of 1, the global treewidth of 1 and the interface of 1 can be easily checked, thus $p \in \ell\text{-TW}(1) \cap g\text{-TW}(1) \cap \text{BI}(1)$. Also, the reduction is feasible in polynomial time.

It remains to show that $|p_m(\mathcal{D})| > s = m$ if and only if ϕ is satisfiable. In order to do so, we start with the following two Claims:

Claim 1: There cannot exist a mapping $\mu \in p(\mathcal{D})$ (and thus in $p_m(\mathcal{D})$) such that $z \in dom(\mu)$ and for each $1 \leq i \leq m$, at least one of $z_{i,1}, z_{i,2}, z_{i,3}$ is in $dom(\mu)$.

Claim 2: For $1 \leq i \leq m$ let μ_i be the mapping $\mu_i = \{(z, 1)\} \cup \{(z_{\alpha,1}, 1), (z_{\alpha,2}, 1), (z_{\alpha,3}, 1) \mid 1 \leq \alpha \neq i \leq m\}$. Then $\mu_i \in p_m(\mathcal{D})$.

To see that the first Claim holds, consider an arbitrary mapping $\mu \in p(\mathcal{D})$. By definition, there exists a mapping ν and a subtree T' of T (containing R), such that $\mu \cup \nu \in Q_{T'}(\mathcal{D})$. Consider $\nu(y)$. Clearly, $y \in dom(\nu)$ and $\nu(y) \in \{a_0, \ldots, a_m\}$. We distinguish two cases: if $\nu(y) = a_0$, then $x \notin dom(\mu)$ since $a_0 \notin (v')^{\mathcal{D}}$. If $\nu(y) = a_j$ for some $1 \leq j \leq m$, then obviously $\nu(x_i) = a_j$ for all $1 \leq i \leq n$. However, since $a_j \notin d_j^{\mathcal{D}}$ and $a_j \notin (d'_j)^{\mathcal{D}}$, we get that $z_{j,1}, z_{j,2}, z_{j,3} \notin dom(\mu)$, what proves the claim.

For the second Claim, we first have to show that for each $1 \leq j \leq m$, there exists some ν_j and a subtree T_j of T containing the root s.t. $\mu_j \cup \nu_j \in Q_{T_j}(\mathcal{D})$. Towards this goal, consider $\nu_j = \{(e, a_j)\} \cup \{(x_i, a_j) \mid 1 \leq i \leq n\}$. For T_j, we consider the complete tree T except the nodes $N_{j,1}, N_{j,2}, N_{j,3}$. It can now be easily checked that $\mu_j \cup \nu_j \in Q_{T_j}(\mathcal{D})$. Indeed, $\nu_j(y) \in v^{\mathcal{D}}$ (node R), $\nu(y) \in (v')^{\mathcal{D}}$ as well as $\mu_j(z) \in g^{\mathcal{D}}$ (node N_0). Also $(\nu_j(e), \nu_j(x_i)) = (a_j, a_j) \in b^{\mathcal{D}}$ (nodes N_i, $1 \leq i \leq n$), and finally, $\nu(x_i) = a_j \in d_\alpha^{\mathcal{D}}, \nu(x_i) = a_j \in (d'_\alpha)^{\mathcal{D}}$ for all $1 \leq \alpha \neq j \leq m$ as well as $\mu(z_{\alpha,1}) = \mu(z_{\alpha,2}) = \mu(z_{\alpha,3}) = 1 \in g^{\mathcal{D}}$ (nodes $N'_{\alpha,\beta}$). The maximality of both, $\mu_j \cup \nu_j$ as well as μ_j follows from the first claim.

We thus have shown that there are at least m maximal solutions to p over $\mathcal{D}$. Next, we show that there at least $m + 1$ maximal solutions if and only if ϕ is satisfiable. We show both directions separately.

Assume that ϕ is satisfiable, i.e. that there exists some truth assignment I on the propositional variables $x_1, \ldots, x_n$ s.t. ϕ evaluates to 1 (true). We show that there exists at least one additional maximal solution for p on $\mathcal{D}$: Consider the mappings μ defined as $\mu = \{(z_{\alpha,\beta}, 1) \mid I(l_{\alpha,\beta}) = 1\}$ and $\nu = \{(y, a_0)\} \cup \{(x_i, I(x_i)) \mid 1 \leq i \leq n\}$ (by slight abuse of notation, we use the same symbols to denote both, the propositional variables in ϕ as well as the variables in p; from the context, it is however always clear which element is actually referred to by x_i). We note that for every $1 \leq j \leq m$, the domain of μ contains at least one of the variables $z_{j,1}, z_{j,2}, z_{j,3}$. This is the case, because for every clause C_j, at least one of the literals $l_{j,1}, l_{j,2}, l_{j,3}$ must evaluate to 1 under I.

Thus, by the first claim we have that μ cannot be extended to a solution in $p(\mathcal{D})$ that is defined on z. At the same time, μ is not contained in any of the mappings μ_j. Thus, in order to show that there exists at least one additional maximal solution, it suffices to show that $\mu \cup \nu \in Q_{T'}(\mathcal{D})$ for some subtree T' of T containing R. In this case, either μ already is the desired solution, or it can be extended to one.

We claim that $\mu \cup \nu \in Q_{T'}(\mathcal{D})$ for the subtree T' of T consisting of the nodes R, $N_1, \ldots, N_n$, as well as at least one node $N'_{\alpha,\beta}$ for each $1 \leq \alpha \leq m$. In fact, this can be easily checked. We have that $\nu(y) \in v^{\mathcal{D}}$ (node R) as well as $(\nu(y), \nu(x_i)) \in b^{\mathcal{D}}$ (node N_i) for every $1 \leq i \leq n$. Finally, consider an arbitrary value α from $\{1, \ldots, m\}$. Since I is a model for ϕ, the clause C_α evaluates to 1 under I. Thus, for at least one $\beta \in \{1,2,3\}$ we have that $I(l_{\alpha,\beta}) = 1$. Thus, by definition of μ we have $\mu(z_{\alpha,\beta}) = 1$ as well. Hence $\mu(z_{\alpha,\beta}) \in g^{\mathcal{D}}$, and it only remains to show that ν also maps $l^*_{\alpha,\beta}$ into $\mathcal{D}$. Towards this goal, we distinguish two cases. If $l_{\alpha,\beta} = x_i$, then $I(x_i) = 1$, and thus also $\nu(x_i) = 1$ by the definition of ν as well as $l^*_{\alpha,\beta} = d_\alpha(x_i)$ by definition. Thus $\nu(x_i) \in d_\alpha^{\mathcal{D}}$. If $l_{\alpha,\beta} = \neg x_i$, then $I(x_i) = 0$, and thus also $\nu(x_i) = 0$ by the definition of ν as well as $l^*_{\alpha,\beta} = d'_\alpha(x_i)$ by definition. Thus $\nu(x_i) \in (d'_\alpha)^{\mathcal{D}}$, which proves the case.

Now assume that there exists at least one additional maximal solution μ in $p_m(\mathcal{D})$. Then for every $1 \leq j \leq m$ it must be the case that $\{z_{j,1}, z_{j,2}, z_{j,3}\} \cap dom(\mu) \neq \emptyset$, since otherwise μ would be contained in at least one μ_j. Thus, by the first claim this means that $z \notin dom(\mu)$. Let ν be the mapping witnessing $\mu \in p(\mathcal{D})$, i.e. the mapping s.t. $\mu \cup \nu \in Q_{T'}(\mathcal{D})$ for some subtree T' of T. Because of $z \notin dom(\mu)$, we have $\nu(y) = a_0$. Thus we have $\nu(x_i) \in \{0,1\}$. We define a truth assignment I on the propositional variables $x_1, \ldots, x_n$ as $I(x_i) = \nu(x_i)$. To see that ϕ evaluates to 1 under I, consider an arbitrary clause C_α. Choose β s.t. $z_{\alpha,\beta} \in \{z_{\alpha,1}, z_{\alpha,2}, z_{\alpha,3}\} \cap dom(\mu)$. To see that $I(l_{\alpha,\beta}) = 1$, distinguish two cases: If $l_{\alpha,\beta} = x_i$, then by definition $d_\alpha(x_i) \in \lambda(N'_{\alpha,\beta})$. We thus get $\nu(x_i) = 1$, which proves the case. If $l_{\alpha,\beta} = \neg x_i$, then by definition $d'_\alpha(x_i) \in \lambda(N'_{\alpha,\beta})$. We thus get $\nu(x_i) = 0$, hence $I(l_{\alpha,\beta}) = I(\neg x_i) = 1 - I(x_i) = 1$. This concludes the proof. $\square$

To show that Max-Enum(ℓ-TW(k) $\cap$ g-TW(k') $\cap$ BI(c)) is not in OutputP using this result, we recall a relationship from [Str10] (also [KSS00]). The lemma below is actually a slight reformulation of [Str10, Lemma 2.11]. However, it holds by the same arguments as the original statement.

Lemma 6.21 ([Str10])**.** *Let E be an enumeration problem. If E is in* OutputP*, then the following problem is in* PTIME*: Given an instance x of E and an integer s encoded in unary, decide if $|E(x)| > s$.*

The intractability of Max-Enum(ℓ-TW(k) $\cap$ g-TW(k') $\cap$ BI(c)) now is an immediate consequence of the previous two results. As ℓ-TW(k) $\cap$ g-TW(k') $\cap$ BI(c) is contained in all classes of wdPTs considered in this work, we obtain the following corollary:

Corollary 6.22. *If* PTIME $\neq$ NP*, then* Max-Enum($\mathcal{C}$) *is not in* OutputP *for every* $\mathcal{C} \in \{\ell\text{-TW}(k) \cap \text{BI}(c), g\text{-TW}(k) \cap \text{SBI}(c), \ell\text{-TW}(k) \cap \text{SBI}(c), g\text{-TW}(k)\}$ *and* $k, c \geq 1$*.*

After having seen only intractability results in the non-parameterized case, we now show that the parameterized problem p-Max-Enum($\mathcal{C}$) is tractable in all but one cases. We first establish the tractability for $\mathcal{C} = g$-TW(k) and $\mathcal{C} = \ell$-TW(k) $\cap$ BI(c), respectively. Of course, this immediately implies tractability for $\mathcal{C} = g$-TW(k) $\cap$ SBI(c).

The overall idea behind the DelayFPT-algorithm can be summarized as follows: Assume that for a pattern tree $p = (T, \lambda, \vec{x})$ and a database $\mathcal{D}$, we can compute for every subtree T' of T (containing the root) the set $Q_{T'}(\mathcal{D})$ efficiently. Since at most $2^{|p|}$ non-maximal solutions can be extended to the same maximal solution, this allows us to enumerate all maximal solutions in DelayFPT.

We start by formulating a crucial lemma, that formalizes the main part of the idea outlined above. It will be convenient to recall the problem PARTIAL-EVAL($\mathcal{C}$) from the literature (cf. [BPS15]): Given a wdPT p, a database $\mathcal{D}$ and a mapping μ, does there exist some $\mu' \in p(\mathcal{D})$ s.t. $\mu \subseteq \mu'$?

Lemma 6.23. *Let $\mathcal{C}$ be a class of wdPTs such that* PARTIAL-EVAL($\mathcal{C}$) *is in* PTIME. *Assume that there exists an enumeration algorithm $\mathcal{A}$, such that for every $p \in \mathcal{C}$ and every database $\mathcal{D}$, $\mathcal{A}$ enumerates some set P of partial solutions with $p_m(\mathcal{D}) \subseteq P$ with delay(i) in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$ for some computable function f, some positive integer m and $0 \leq i \leq |P|$. Then p-*MAX-ENUM($\mathcal{C}$) *is in* DelayFPT.

Proof. Given the enumeration algorithm $\mathcal{A}$ for P, the idea is to construct an extension $\mathcal{A}'$ of $\mathcal{A}$ that, after initializing a set $M = \emptyset$, performs the following steps: (1) Retrieve the next output μ of $\mathcal{A}$ and extend it to a maximal solution μ_m. If μ_m has not been created before, add μ_m to M; (2) If the previous step was repeated $2^{|p|}$ times since the last output, output and delete a mapping from M. If $\mathcal{A}$ halts, $\mathcal{A}'$ outputs M and halts as well.

So let $p = (T, \lambda, \vec{x}) \in \mathcal{C}$ with database $\mathcal{D}$ and P be a set of partial solutions containing all mappings from $p_m(\mathcal{D})$. Further let $\mathcal{A}$ be an enumeration algorithm such that *delay*(i) is in $\mathcal{O}(f(|p|) \cdot |\mathcal{D}|^m)$ for some computable function f, some positive integer m and $0 \leq i \leq |P|$. We will extend $\mathcal{A}$ to some algorithm $\mathcal{A}'$ which enumerates the maximal solutions $p_m(\mathcal{D})$. To do so, we will need the following:

Claim. *Let $\mu : \mathbf{X} \to \mathbf{U}$ be a partial solution. Then μ can be extended to a mapping $\mu' \in p_m(\mathcal{D})$ in polynomial time.*

Proof of the Claim. If $\vec{x} = dom(\mu)$ we can test by PARTIAL-EVAL($\mathcal{C}$) whether μ is maximal, so let $x \in \vec{x} \setminus dom(\mu)$. Let D be the domain of $\mathcal{D}$. By partial evaluation, we can test for all $a \in D$ if $\mu' = \mu \cup \{(x, a)\} \subseteq \nu$ for some $\nu \in p(\mathcal{D})$. If there is no $x \in \vec{x} \setminus dom(\mu)$ and no $a \in D$ such that $\mu' \subseteq \nu$ for some $\nu \in p(\mathcal{D})$, then μ has to be maximal, since it is a partial mapping and thus contained in some maximal one.

If there is an a with $\mu \cup \{(x, a)\} \subseteq \nu$, we can extend μ to μ' and test the maximality of μ'. This procedure terminates after polynomial time. □

The algorithm $\mathcal{A}'$ extends $\mathcal{A}$ as follows: Initialize a set $M = \emptyset$ and a counter $j = 0$.

- If there is an output μ in $\mathcal{A}$, then extend μ to a maximal solution μ_m. If μ_m has not been output before, set $M \leftarrow M \cup \{\mu_m\}$. Further set $i \leftarrow i + 1$.
- If $i = 2^{|p|}$, then set $i = 0$ and output some $\nu_m \in M$. Further set $M \leftarrow M \setminus \{\nu_m\}$.

- If $\mathcal{A}$ halts, output M.

Since $p_m(\mathcal{D}) \subseteq P$, $\mathcal{A}'$ outputs $p_m(\mathcal{D})$. Thus it remains to show that we can do this with a delay of $\mathcal{O}(g(|p|) \cdot |\mathcal{D}|^k)$ for some function g and some positive integer k. With the RAM machine model, we can build an exponentially large search tree for the output (or the set of maximal mapping already created earlier) and check for any μ_m if it is in the output at any given time in polynomially many steps, which we denote by the polynomial function $\rho(|\mathcal{D}|, |p|)$. If M is never empty, the delay between any two outputs is bounded by $\mathcal{O}(2^{|p|} \cdot \rho(|\mathcal{D}|, |p|) \cdot f(|p|) \cdot |\mathcal{D}|^m)$. Thus it suffices to show that M is not empty at any output-step of algorithm $\mathcal{A}'$. For the sake of a contradiction assume that we are at the K-th output step and that $M = \emptyset$. This means that $K - 1$ maximal solutions have been output so far, and $K(2^{|p|} + 1)$ partial solutions have been extended to maximal solutions. As every maximal solution corresponds to at most $2^{|p|}$ subsumed solutions, at least K different maximal mappings have been found by extending partial solutions, which contradicts $M = \emptyset$. □

For a class $\mathcal{C}$ of wdPTs to show that Max-Enum($\mathcal{C}$) is in DelayFPT it thus remains to identify a suitable set of partial solutions. We do this for the classes mentioned before.

Theorem 6.24. *Let $\mathcal{C}'$ be a class of CQs s.t.* CQ-Eval($\mathcal{C}'$) *is in* PTIME*, $k, c \geq 1$, and $\mathcal{C} \in \{g\text{-}\mathcal{C}', l\text{-}\mathcal{C}' \cap \mathsf{BI}(c)\}$. Then the problem p-*Max-Enum($\mathcal{C}$) *is in* DelayFPT*.*

Proof. First let $p = (T, \lambda, \vec{x})$ be a pattern tree in g-$\mathcal{C}'$ with database $\mathcal{D}$. For all $\vec{x}' \subseteq \vec{x}$, let $T_{\vec{x}'}$ be the minimal subtree of T rooted in R such that the variables of $T_{\vec{x}'}$ are exactly $\vec{x}'$ (if no such subtree exists set $T_{\vec{x}'} = \emptyset$). Further set $Q_{T_{\vec{x}'}} = \{\lambda(N) \mid N \in V(T_{\vec{x}'})\}$ and $Q = \{Q_{T_{\vec{x}'}} \mid T_{\vec{x}'} \subseteq T$ rooted in $R\}$. Since CQs in $\mathcal{C}'$ can be enumerated in polynomial delay (cf. [BDGM12]) and for any two queries $Q_1, Q_2 \in Q$ we have $var(Q_1) \neq var(Q_2)$, there exists an algorithm $\mathcal{A}$ that outputs $L = \bigcup_{q \in Q} q(\mathcal{D})$ with polynomial delay (just enumerate the partial solutions corresponding to each subtree T'). Every partial solution with domain $\{x_1, \ldots, x_n\}$ is a solution on the minimal subtree of $\{x_1, \ldots, x_n\}$, thus $\mathcal{A}$ enumerates all partial solution, and hence all maximal solutions with polynomial delay. Thus p-Max-Enum(g-$\mathcal{C}'$) is in DelayFPT by Lemma 6.23.
Next let $p = (T, \lambda, \vec{x})$ be a pattern tree in $l - \mathcal{C}' \cap \mathsf{BI}(c)$ with database $\mathcal{D}$. Then by Corollary 6.14 we can enumerate $p(\mathcal{D})$ with polynomial delay, and since $p_m(\mathcal{D}) \subseteq p(\mathcal{D})$ we are done by Lemma 6.23. □

Algorithm $\mathcal{A}'$ sketched in the proof of Lemma 6.23 crucially depends on the choice of RAMs as the model of computation: $\mathcal{A}'$ may need to store an exponential number of maximal solutions. A Turing Machine (TM) cannot access these solutions efficiently, while a RAM can. However, these algorithms could be easily adapted to run in *incremental delay* on a TM, i.e. for some $m \in \mathbb{N}$ and computable function f, *delay*(i) is in $\mathcal{O}(f(|p|) \cdot (|p| + |\mathcal{D}| + \sum_{j=1}^{i} |y_i|)^m)$ for $i \geq 0$ (where $y_1, \ldots, y_i$ are the first i solutions returned by the algorithm).

We have thus shown fixed-parameter tractability for three out of the four classes we consider. We conclude this section by showing that for the remaining class the problem is not in $\mathsf{OutputFPT}$.

Proposition 6.25. *If* $\mathsf{FPT} \neq \mathsf{W}[1]$*, then* p-$\textsc{Max-Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ *is not in* $\mathsf{OutputFPT}$ *for every* $k, c \geq 1$*.*

Proof. Recall again the proof of Theorem 6.9, which shows the $\mathsf{W}[1]$-hardness of p-$\textsc{Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$, as well as of p-$\textsc{Max-Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ in Proposition 6.11. The proof is by reduction from the p-$\textsc{Clique}$-problem. Let p and $\mathcal{D}$ be the wdPT and database defined by the reduction. We observe that $|p(\mathcal{D})| \leq 2^{|p|}$, and thus also $|p_m(\mathcal{D})| \leq 2^{|p|}$. We prove the proposition by contradiction.

Assume to the contrary that there exists some algorithm A in $\mathsf{OutputFPT}$ that returns all mappings $p'_m(\mathcal{D}')$ for any wdPT $p' \in \ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ and database $\mathcal{D}'$. For p and $\mathcal{D}$ the algorithm thus had a runtime in $\mathcal{O}((2^{|p|} \cdot |p| + (|\mathcal{D}| + |p|))^m \cdot f(|p|)$ (for some $m \in \ell$ and some computable function f). This would give rise to a decision procedure for p-$\textsc{Max-Enum}(\ell\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ in FPT, which gives the desired contradiction. $\square$

CHAPTER 7

Conclusion

7.1 Summary

Starting from a dichotomy on the enumeration complexity for acyclic self-join free-CQs [BDG07], we have embarked on a complexity analysis of the enumeration problem for three query languages based on CQs: CQs themselves in the presence of dependencies, unions of CQs and well-designed pattern trees.

Previous hardness results as well as the classification for CQs given in [BDG07] regarding the enumeration complexity of CQs no longer hold in the presence of dependencies. We have shown that some of the queries that where previously classified as hard become tractable in the presence of FDs, and that the others remain intractable. We have classified the enumeration complexity of self-join-free CQs according to their FD-extension. Under reasonable assumptions for lower bounds in decision complexity, we have the following: A query is in $\mathsf{DelayC_{lin}}$ if its extension is free-connex, it is not in $\mathsf{DelayC_{lin}}$ if its extension is acyclic but not free-connex, and it is not even decidable whether there exists an answer to the query over a database in linear time if the schema has only unary FDs and its extension is cyclic. We also have shown that these results apply for CQs in the presence of CDs. In addition to our results on constant delay enumeration, we have shown that this work has consequences in other enumeration classes such as $\mathsf{DelayLin}$.

Regarding UCQs, we have observed how one set of CQs within a union can influence the enumeration complexity of another set of CQs by providing variables, and formalized this observation by introducing union extensions. Then, we generalized the notion of free-connexity to the setting of UCQs, and showed that these queries are indeed tractable. In particular, we demonstrated that UCQs containing only intractable CQs may be tractable. Towards lower bounds for UCQs, we showed that in case of a union of two intractable CQs or two acyclic body-isomorphic CQs, free-connexity fully captures the tractable cases.

The two approaches for extending a CQ in order to encode additional information, either by provided variables in case of union extensions or by certain variables on the right-hand side of a dependency in case of FD-extensions, are by no means mutually exclusive. In fact, we can find even more tractable classes of UCQs by simply starting with a UCQ, first extending all of the CQs within the union according to the FDs, and then computing the union extension.

When introducing partial matchings to CQs, we have seen that enumeration within $\mathsf{DelayC_{lin}}$ is only possible for very restricted structures of well-designed pattern trees. We have thus evaluated the complexity of enumerating all solutions and of enumerating the maximal solutions of wdPTs in terms of combined complexity. Due to the close relationship with the parameterized complexity of the corresponding evaluation problem, we have also revisited the evaluation and max-evaluation problems. A summary of the main results is given in Table 6.1 on page 100. For the problems Eval($\mathcal{C}$) and Max-Eval($\mathcal{C}$) we have identified fixed-parameter tractable and intractable cases. Likewise, for the two variants of the enumeration problem, we have identified tractable and intractable cases – both in terms of classical and parameterized complexity. More precisely, for the classical complexity, we have established tractability by showing a strong form of tractability (i.e., polynomial delay) and we have established intractability by ruling out even a weaker form of tractability (i.e. output polynomial time). We have proved analogous results from a parameterized complexity point of view.

7.2 Future Work

The results presented in this work open up quite a few directions for future work. Throughout the thesis, we have used several well-established hypotheses for lower bounds in decision complexity, such as mat-mul or hyperclique. Since proving unconditional lower bounds for such problems is highly unlikely in the current state of complexity theory, it would we very interesting to base our lower bounds on alternative assumptions in complexity theory, such as $\mathsf{PTIME} \neq \mathsf{NP}$, the ETH [LMS$^+$13] or even separation assumptions in parameterized or counting complexity.

Regarding CQs in the presence of dependencies, our proof for the hardness of FD-cyclic CQs assumes that all FDs are unary. The question remains open whether this result holds for general FDs, along with the classification of Example 4.22. In addition, to show that enumerating CD-free-connex CQs can be done in $\mathsf{DelayC_{lin}}$, we store all printed results. The required space has the size of the output, which may be polynomial in that of the input. It is unclear whether a solution that requires less space exists. Moreover, it would be interesting if our method of FD-extensions can also be used for other integrity constraints of CQs.

For UCQs, a full classification for the general case remains an open problem. In Section 5.3, we described the next steps we plan to tackle in this vein, and provided examples with unknown complexity. Resolving these examples is a necessary step on the way to a future dichotomy. As with cardinality dependencies, the memory we used in our techniques for the tractable UCQs that do not contain only tractable CQs may increase in size by a constant with every new answer. An interesting question is whether we can achieve the same time bounds when restricting the memory to only be allocated in the preprocessing step.

For wdPTs, even though we have provided quite a comprehensive picture of the complexities in various settings, Table 6.1 still calls for further work. Above all, the Enum($\mathcal{C}$) problem with $\mathcal{C} = g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c)$ is open. Also, for two of our W[1]- and W[2]-hardness results, a matching upper bound is missing. Finally, the search for tractable classes both for evaluation and enumeration of wdPTs should be continued. Note that none of the restrictions studied in this thesis sufficed to ensure tractability of Max-Enum($\mathcal{C}$). Hence, further restrictions should be an issue addressed in future research as well.

APPENDIX A

Full Proof of Theorem 6.8

In this appendix, we will give the full proof of Theorem 6.8. It is convenient to introduce some additional notation and to provide a formal definition for some of the notions used.

For a wdPT $p = (T, \lambda, \vec{x})$, we introduce the additional notation of fvars(T) as fvars$(T) = var(T) \cap \vec{x}$, and similar for fvars$(p)$ and fvars(N) for $N \in V(T)$. For graphs, we recall the notion of an *induced (sub)graph*. Let $G = (V, E)$ and $S \subseteq V$. Then the subgraph $G[S] = (S, E')$ where $E' = \{e \in E \mid e \cap S = e\}$ is called the *induced (sub)graph*. We also say that $G[S]$ is the subgraph of G induced by S. Furthermore, we say that two nodes $v, w \in V$ are *adjacent* if $\{v, w\} \in E$.

Instead of proving Theorem 6.8 directly, it is convenient to prove a slightly stronger result, namely: *The problem* ExtendSolution$(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ *parameterized by the query size is in* FPT *for every* $k, c \geq 1$.

Recall from Section 6.3 that the ExtendSolution$(\mathcal{C})$ problem is defined as follows: we are given a database $\mathcal{D}$, a wdPT $p \in \mathcal{C}$, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p). The question is if there exists some mapping $\mu' \in p(\mathcal{D})$ such that $\mu \subseteq \mu'$ and $dom(\mu') \cap \vec{x}' = \emptyset$. Clearly, the problem Eval$(\mathcal{C})$ corresponds to the special case of the ExtendSolution$(\mathcal{C})$ problem where we set $\vec{x}' \cup dom(\mu) = \vec{x}$, i.e., we ask if μ itself is the desired mapping μ'.

Throughout this section, let $D = dom(\mathcal{D})$. Also, in order to avoid confusion, we use upper case letters to denotes nodes in a wdPT, and lower case letters for vertices in the Gaifman graph or in tree decompositions.

The proof proceeds in several steps.

Critical subsets of the interface variables.

Consider two nodes N and M in the wdPT $p = (T, \lambda, \vec{x})$, s.t. N is the parent of M. Let μ be a mapping that binds the interface variables between N and M, i.e.,

$\mathcal{I}(M, N) \subseteq \mathit{dom}(\mu)$. Our first goal is to establish a criterion to decide whether μ can be extended to a mapping ν on $\mathit{var}(M)$, s.t. $\nu(\lambda(M)) \subseteq \mathcal{D}$, i.e. whether ν maps M into $\mathcal{D}$.

Definition A.1. Let $\mathcal{G}(M)$ denote the Gaifman graph of the CQ $Q() \leftarrow \lambda(M)$ and let $\mathcal{G}^L(M)$ denote the subgraph of $\mathcal{G}(M)$ induced by the variables in $\mathit{var}(M) \setminus \mathcal{I}(M, N)$ (i.e., the "local" variables of M). Let CONN denote the set of connected components of $\mathcal{G}^L(M)$. Then the set LOCAL of *locally connected subsets* of $\mathcal{I}(M, N)$ is defined as follows:

$\mathit{LOCAL} = \{\vec{v} \mid \exists E \in \mathit{CONN},\ \vec{v} = \{v \in \mathcal{I}(M, N) \mid \exists y \in E, \text{ s.t. } v, y \text{ are adjacent in } \mathcal{G}\}\}.$

Definition A.2. The set $\mathcal{C}(M, N)$ of *critical subsets* of $\mathcal{I}(M, N)$ contains precisely the subsets $\vec{v} \subseteq \mathcal{I}(M, N)$, s.t.

- either there exists an atom A in M with $\vec{v} = \mathit{var}(A)$,
- or $\vec{v} \in \mathit{LOCAL}$, where LOCAL denotes the locally connected subsets of $\mathcal{I}(M, N)$.

Definition A.3. Let $\vec{v} \in \mathcal{C}(M, N)$ be a *critical subset* of $\mathcal{I}(M, N)$. We define the "good" and "bad" values for mappings μ with $\mathit{dom}(\mu) = \vec{v}$ as follows:

- $\mathit{good}(\vec{v}) = \{\mu \mid \mathit{dom}(\mu) = \vec{v}$ and there exists an extension ν of μ to $\mathit{var}(M)$ with $\nu(M) \subseteq \mathcal{D}\}$.
- $\mathit{bad}(\vec{v}) = \{\mu \mid \mathit{dom}(\mu) = \vec{v}$ and there exists no extension ν of μ to $\mathit{var}(M)$ with $\nu(M) \subseteq \mathcal{D}\}$.

We observe that, for every $\vec{v} \in \mathcal{C}(M, N)$, the sets $\mathit{good}(\vec{v})$ and $\mathit{bad}(\vec{v})$ can be computed in polynomial time w.r.t. the input $\mathcal{D}$ and p. Indeed, there are at most $|D|^{|\vec{v}|}$ choices of μ that we have to test, with $|\vec{v}| \leq |I| \leq c$. Moreover, since $p \in g\text{-}\mathsf{TW}(k)$ (actually, even $p \in \ell\text{-}\mathsf{TW}(k)$ would suffice) we can test in polynomial time for each μ if μ is in $\mathit{good}(\vec{v})$ or in $\mathit{bad}(\vec{v})$.

The following lemma gives a necessary and sufficient criterion for mappings μ defined on $\mathcal{I}(M, N)$ to decide if μ can be extended to M or not.

Lemma A.4. *Let μ be a mapping with $\mathcal{I}(M, N) \subseteq \mathit{dom}(\mu)$. Then μ can be extended to a mapping ν on $\mathit{var}(M)$, s.t. $\nu(M) \subseteq \mathcal{D}$ if and only if for every $\vec{v} \in \mathcal{C}(M, N)$, we have $\mu_{|\vec{v}} \in \mathit{good}(\vec{v})$.*

Proof. The "only if"-direction follows immediately from the definition of $\mathit{good}(\vec{v})$. For the "if"-direction, assume that $\mu_{|\vec{v}} \in \mathit{good}(\vec{v})$ holds for every $\vec{v} \in \mathcal{C}(M, N)$. In particular, $\mu_{|\vec{v}} \in \mathit{good}(\vec{v})$ holds for every locally connected subset $\vec{v}$ of $\mathcal{C}(M, N)$. Hence, for every such $\vec{v}$, there exists an extension $\nu^{[\vec{v}]}$ to $\mathit{var}(M)$ with $\nu^{[\vec{v}]}(M) \subseteq \mathcal{D}$.

Now consider an arbitrary connected component E of the graph $\mathcal{G}^L(M)$ from Definition A.1. By definition of LOCAL, there exists a unique critical subset $\vec{v} \in \mathcal{C}(M, N)$, s.t.

$\vec{v}$ consists precisely of the interface variables adjacent in the Gaifman graph $\mathcal{G}(M)$ to some variable in E. For all variables $y \in \mathit{var}(E)$, we set $\nu(y) = \nu^{[\vec{v}]}(y)$.

We claim that this mapping ν is the desired extension of μ to $\mathit{var}(M)$ with $\nu(M) \subseteq \mathcal{D}$. This claim is proved by inspecting each atom τ in the pattern of M:

1. If $\mathit{var}(\tau) \subseteq \mathcal{I}(M,N)$, then $\mathit{var}(\tau)$ itself is a critical subset $\vec{v}$ of $\mathcal{I}(M,N)$ by the definition of $\mathcal{C}(M,N)$. Since we are assuming $\mu_{|\vec{v}} \in \mathit{good}(\vec{v})$, we clearly have $\nu(\tau) = \mu_{|\vec{v}}(\tau) \in \mathcal{D}$.
2. Now suppose that $\mathit{var}(\tau) \not\subseteq \mathcal{I}(M,N)$, i.e., $\mathit{var}(\tau) \setminus \mathcal{I}(M,N) \neq \emptyset$. By the definition of the Gaifman graph, all variables in $\mathit{var}(\tau) \setminus \mathcal{I}(M,N)$ are contained in the same connected component of $\mathcal{G}^L(M)$. Let $\vec{v}$ denote the corresponding locally connected subset of $\mathcal{I}(M,N)$. Clearly, all variables in $\mathit{var}(\tau) \cap \mathcal{I}(M,N)$ are contained in $\vec{v}$. Then, since $\nu(\tau) = \nu^{[\vec{v}]}(\tau)$ and $\nu^{[\vec{v}]}(\tau) \in \mathcal{D}$, we have $\nu(\tau) \subseteq \mathcal{D}$ as desired.

□

Blocking the extension of a mapping to several nodes.

The critical subsets $\mathcal{C}(M,N)$ above allowed us to provide a criterion for deciding if some mapping μ with $\mathit{dom}(\mu) \subseteq \mathcal{I}(M,N)$ can be extended to a mapping ν on $\mathit{var}(M)$ with $\nu(M) \subseteq \mathcal{D}$. The theorem requires to verify that a mapping ν cannot be extended to any node from a possibly big subset of p, namely the set of nodes containing at least one variable from $\vec{x}'$. Before we exhibit an FPT-algorithm for this verification task, we introduce a plausibility check for the set $\vec{x}'$ of free variables to which we must not extend the given mapping μ:

Consider two arbitrary variables $x_i, x_j \in \vec{x}$ and let N_i (resp. N_j) denote the node in wdPT p where x_i (resp. x_j) first occurs in top-down direction. Now suppose that $x_i \in \mathit{dom}(\mu)$. If $N_i = N_j$ or N_i is a descendant of N_j, then also $x_j \in \mathit{dom}(\mu)$ must hold. Intuitively, if some variable is in the domain of μ, then all other variables introduced in the same node or in some ancestor node must also be in the domain of μ. If this condition (which is easy to check) is violated by μ, then we can immediately reject μ. In the sequel, we may therefore assume w.l.o.g. that this condition is fulfilled.

Given a database $\mathcal{D}$, a wdPT p, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p), we now define two sets of nodes $\mathcal{N}_\mu$ and $\mathcal{M}_\mu$, s.t. we search for a solution $\mu' \in p(\mathcal{D})$, which sends all nodes in $\mathcal{N}_\mu$ into $\mathcal{D}$ and which does not bind any of the variables introduced in any of the nodes in $\mathcal{M}_\mu$. For node N with parent N', we write $\mathit{nvar}(N)$ do denote the "new" variables in N, i.e., $\mathit{nvar}(N) = \mathit{var}(N) \setminus \mathcal{I}(M,N)$.

$\mathcal{N}'_\mu = \{N \mid N \text{ is a node in } p \text{ and } \mathit{nvar}(N) \cap \mathit{dom}(\mu) \neq \emptyset\}$.

$\mathcal{N}_\mu = \mathcal{N}'_\mu \cup \{N \mid N \text{ is the ancestor of some node in } \mathcal{N}'_\mu\}$

$\mathcal{M}'_\mu = \{M \mid M \text{ is a node in } p \text{ and } \mathit{nvar}(M) \cap \vec{x}'\}$

$\mathcal{M}_\mu = \{M \mid M \in \mathcal{M}'_\mu$ and no ancestor of M is in $\mathcal{M}'_\mu\}$.

Intuitively, $\mathcal{N}_\mu$ contains the nodes which a desired solution $\mu' \in p(\mathcal{D})$ is requested to send into $\mathcal{D}$. The two-stage definition via $\mathcal{N}'_\mu$ was needed to add those nodes in p which do not introduce new variables but which are on the path from the root of p to some node N with $nvar(N) \cap dom(\mu) \neq \emptyset$. The two-stage definition of $\mathcal{M}_\mu$ via $\mathcal{M}'_\mu$ is based on the intuition that if we block μ' from an extension to some node M, then we implicitly of course also block the extension to any descendant of M. Hence, $\mathcal{M}_\mu$ only retains the top-most node in case $\mathcal{M}'_\mu$ contains ancestor-descendant pairs.

By our plausibility check above, we can be sure that a node in $\mathcal{N}_\mu$ can never be a descendant of a node in $\mathcal{M}_\mu$. Since $\mathcal{N}_\mu$ is closed under the ancestor relation, we can in fact be sure that, for every node $M \in \mathcal{M}_\mu$, some ancestor (but not necessarily the parent) of M is contained $\mathcal{N}_\mu$. In our decision procedure for the ExtendSolution(g-$\mathsf{TW}(k) \cap \mathsf{SBI}(c)$) problem, it will be important that there is no "gap" between the nodes in $\mathcal{N}_\mu$ and in $\mathcal{M}_\mu$. Therefore, we have to add nodes to $\mathcal{N}_\mu$ and/or $\mathcal{M}_\mu$ to ensure that for every node $M \in \mathcal{M}_\mu$, the parent of M is in $\mathcal{N}_\mu$. In principle, exponentially many (w.r.t. the size of p) possible combinations of $\mathcal{N}_\mu$ and $\mathcal{M}_\mu$ are thus possible. Since we are only interested in an FPT upper bound, we can afford to try all these possibilities. Hence, from now on, we may assume that for every node $M \in \mathcal{M}_\mu$, the parent of M is in $\mathcal{N}_\mu$.

The following lemma shows how to test for the existence of a solution μ' of $p(\mathcal{D})$ that sends all nodes in $\mathcal{N}_\mu$ into D but which cannot be extended to any of the nodes in $\mathcal{M}_\mu$.

Lemma A.5. *Given a database $\mathcal{D}$, a wdPT p, a partial mapping μ and a set $\vec{x}' \subseteq \vec{x}$ (where $\vec{x}$ are the free variables of p). Let $\mathcal{N}_\mu$ and $\mathcal{M}_\mu$ be as defined above. Moreover, let $\vec{v}_\mathcal{N}$ denote* all *variables occurring in $\mathcal{N}_\mu$, i.e., $\vec{v}_\mathcal{N} = \bigcup_{N \in \mathcal{N}_\mu} var(N)$. The following equivalence holds:*

There exists an extension μ' of μ with $\mu' \in p(\mathcal{D})$ if and only if there exists an extension ν of μ to $\vec{v}_\mathcal{N}$, s.t.

1. *$\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}_\mu$ and*
2. *for every $M \in \mathcal{M}_\mu$ with parent $N \in \mathcal{N}_\mu$, there exists $\vec{v} \in \mathcal{C}(M, N)$, s.t. $\nu_{|\vec{v}} \in bad(\vec{v})$.*

Proof. The "only if" direction is immediate. Just set $\nu = \nu'_{|\vec{v}_\mathcal{N}}$. For the "if" direction, suppose that there exists an extension ν of μ to $\vec{v}_\mathcal{N}$, s.t. conditions (1) and (2) of the lemma hold. It suffices to show that, for every $M \in \mathcal{M}_\mu$, ν cannot be extended to ν' on $nvar(M)$, s.t. $\nu'(M) \subseteq \mathcal{D}$. Note that it may be the case that ν is not yet the desired solution $\mu' \in p(\mathcal{D})$ because ν can be extended to further variables in $var(p)$. However, if it is guaranteed that no extension of ν can send any of the nodes $M \in \mathcal{M}_\mu$ into $\mathcal{D}$, then we can be sure that a solution $\mu' \in p(\mathcal{D})$ with the desired properties indeed exists. By Lemma A.4, we conclude that condition (2) of the lemma expresses the condition that for every $M \in \mathcal{M}_\mu$, ν cannot be extended to ν' on $nvar(M)$, s.t. $\nu'(M) \subseteq \mathcal{D}$. □

In principle, a decision procedure for the problem $\textsc{ExtendSolution}(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ can be obtained by a direct application of Lemma A.5: let $\mathcal{M}_\mu = \{M_1, \dots, M_\beta\}$ and, for every $i \in \{1, \dots, \beta\}$, let N_i denote the parent of M_i. By Lemma A.5, we know that there exists an extension μ' of μ with $\mu' \in p(\mathcal{D})$ and $dom(\mu) \neq \vec{x}' = \emptyset$ if and only if there exists a combination $(C_1, \dots, C_\beta)$ of critical subsets $C_i \in \mathcal{C}(M_i, N_i)$, s.t. there exists an extension ν of μ to $\vec{v}_{\mathcal{N}}$ with

1. $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}_\mu$ and
2. $\nu_{|\vec{v}_i} \subseteq bad(\vec{v}_i)$ for every $i \in \{1, \dots, \beta\}$.

Hence, our decision procedure for the problem $\textsc{ExtendSolution}(g\text{-}\mathsf{TW}(k) \cap \mathsf{SBI}(c))$ just needs to check if such a combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ of critical subsets exists. We can search for such a combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ by nested loops over all $\vec{v}_i \in \mathcal{C}(M_i, N_i)$ with $i \in \{1, \dots, \beta\}$. Since $\vec{v}_i \subseteq \mathcal{I}(M_i, N_i)$ and $|\mathcal{I}(M_i, N_i)| \leq c$, there are at most 2^c elements in each $\mathcal{C}(M_i, N_i)$. Moreover, β is bounded by the size of p. Hence, we have to check at most $f(p) = (2^c)^{|p|}$ s combinations $(\vec{v}_1, \dots, \vec{v}_\beta)$. To prove the algorithm to be in FPT, it suffices to show that, for a given combination $(\vec{v}_1, \dots, \vec{v}_\beta)$ of critical subsets, one can test in polynomial time (w.r.t. the size of the input $\mathcal{D}$ and p) if there exists an extension ν of μ to $\vec{v}_{\mathcal{N}}$ with

1. $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}_\mu$ and
2. $\nu_{\vec{v}_i} \subseteq bad(\vec{v}_i)$ for every $i \in \{1, \dots, \beta\}$.

Recall that $bad(\vec{v}_i)$ for every critical subset $\vec{v}_i$ can be computed in polynomial time. Our final goal is, therefore, to transform the given tree decomposition $\mathcal{T}$ of p into a tree decomposition $\mathcal{T}^*$, s.t. every $\vec{v}_i$ is covered by some bag in $\mathcal{T}^*$, i.e., for every $i \in \{1, \dots, \beta\}$, there exists a node t_i in $\mathcal{T}^*$, such that $\vec{v}_i$ is a subset of the bag at t_i and such that the width of $\mathcal{T}^*$ is still bounded by a constant. Clearly, if we have such a tree decomposition $\mathcal{T}^*$, then our test for the existence of an extension ν of μ to $\vec{v}_{\mathcal{N}}$ with

1. $\nu(N) \subseteq \mathcal{D}$ for every $N \in \mathcal{N}_\mu$ and
2. $\nu_{\vec{v}_i} \subseteq bad(\vec{v}_i)$ can be done in polynomial time (w.r.t. the size of $\mathcal{D}$ and p).

Transformation of the global tree decomposition

We transform the global tree decomposition $\mathcal{T}$ of p into the tree decomposition $\mathcal{T}^*$ by the following algorithm:

```
$\mathcal{T}^* := \mathcal{T}$;
for every $i \in \{1, \dots, \beta\}$ {
        if $\vec{v}_i$ is covered by some bag in $\mathcal{T}^*$ then do nothing;
        else {
            // i.e., $\vec{v}_i$ is a locally connected subset of $\mathcal{I}(M, N)$
            let $\vec{e}_i$ be a connected component in $\mathcal{G}^L(M_i)$, s.t.
                the variables in $\vec{v}_i$ are connected via variables in $\vec{e}_i$;
            choose an arbitrary variable $v_i \in \vec{e}_i$;
            choose an arbitrary node $t_i$ in $\mathcal{T}^*$, s.t. $v_i$ is in the bag $B_i$ of $t_i$;
            add all of $\vec{v}_i$ to the bag $B_i$;
            // restore the connectedness condition of $\mathcal{T}^*$, i.e.:
            for every node $t_j$ in $\mathcal{T}^*$ with $y$ in bag $B_i$ of $t_i$ for some $y \in \vec{v}_i$ do
                add $y$ to the bag of every $t$ along the path from $t_i$ to $t_j$;
        }
    }
```

The following lemma states that $\mathcal{T}^*$ resulting from the above algorithm indeed has the desired properties.

Lemma A.6. *Let $\mathcal{T}^*$ be the tree decomposition resulting from the above transformation of the tree decomposition $\mathcal{T}$ of wdPT p. Then $\mathcal{T}^*$ fulfills the following properties:*

1. *for every $i \in \{1, \ldots, \beta\}$, there exists a node t_i in $\mathcal{T}^*$, s.t. $\vec{v}_i \subseteq B_i$, where B_i denotes the bag of t_i and*
2. *the width of $\mathcal{T}^*$ is at most $(k+1) \cdot (c+1)$.*

Proof. Property (1) follows immediately from the construction of $\mathcal{T}^*$. It remains to show that the width of $\mathcal{T}^*$ has the desired upper bound. To this end, consider an arbitrary node s in the tree decomposition $\mathcal{T}^*$ and suppose that the bag B of node s is augmented during the construction of $\mathcal{T}^*$. Note that, in each iteration of the loop, we add at most $|\vec{v}_i|$ variables to B with $|\vec{v}_i| \leq c$. Moreover, we add $\vec{v}_i$ or some variable $y \in \vec{v}_i$ to the bag B of s only if B contains some "local variable" $u_i \in \vec{e}_i$ (this is ensured by the connectedness condition of tree decompositions). We assume that $\mathcal{T}$ has treewidth k, i.e., the size of each bag in $\mathcal{T}$ is bounded by k. Hence, each vertex s in $\mathcal{T}$ can be augmented in at most $k+1$ iterations of the loop (it is ensured by the well-designedness condition of wdPTs that no "local variable" u_i can also occur in a different node $M_j \in \mathcal{M}_\mu$ with $j \neq i$). In total, we thus start with bag size at most $k+1$ and we add at most $(k+1)$ times up to c variables to each bag. Hence, the bag size of $\mathcal{T}^*$ is bounded by $k + (k+1) \cdot c \leq (k+1) \cdot (c+1)$. □

This concludes our proof of Theorem 6.8.

Bibliography

[ABW18] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. If the current clique algorithms are optimal, so is valiant's parser. *SIAM Journal on Computing*, 47(6):2527–2555, 2018.

[AF96] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21 – 46, 1996. First International Colloquium on Graphs and Optimization.

[AFG16] Myrto Arapinis, Diego Figueira, and Marco Gaboardi. Sensitivity of counting queries. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, Rome, Italy*, pages 120:1–120:13, 2016.

[AFP$^+$15] Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Simkus, and Sebastian Skritek. Towards reconciling SPARQL and certain answers. In *Proc. WWW 2015*, pages 23–33. ACM, 2015.

[AU16] Marcelo Arenas and Martín Ugarte. Designing a query language for RDF: marrying open and closed worlds. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 225–236. ACM, 2016.

[AW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.

[Bag06] Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *International Workshop on Computer Science Logic*, pages 167–181. Springer, 2006.

[BB13] Johann Brault-Baron. *De la pertinence de l'énumération: complexité en logiques propositionnelle et du premier ordre.* PhD thesis, Université de Caen, 2013.

[BDG07] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *International Workshop on Computer Science Logic*, pages 208–222. Springer, 2007.

[BDGM12] Andrei A. Bulatov, Víctor Dalmau, Martin Grohe, and Dániel Marx. Enumerating homomorphisms. *J. Comput. Syst. Sci.*, 78(2):638–650, 2012.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[BKPS18] Pablo Barceló, Markus Kröll, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and static analysis for well-designed pattern trees with projection. *ACM Trans. Database Syst.*, 43(2):8:1–8:44, 2018.

[BKS17] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*, pages 303–318, 2017.

[BKS18] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering UCQs under updates and in the presence of integrity constraints. In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, pages 8:1–8:19, 2018.

[BPS15] Pablo Barceló, Reinhard Pichler, and Sebastian Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *Proc. PODS 2015*, pages 131–144. ACM, 2015.

[CFWY14] Yang Cao, Wenfei Fan, Tianyu Wo, and Wenyuan Yu. Bounded conjunctive queries. *PVLDB*, 7(12):1231–1242, 2014.

[CH97] Nadia Creignou and J-J Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique théorique et applications*, 31(6):499–511, 1997.

[CK] Nofar Carmeli and Markus Kröll. On the enumeration complexity of unions of conjunctive queries. To appear in the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems.

[CK18] Nofar Carmeli and Markus Kröll. Enumeration complexity of conjunctive queries with functional dependencies. In Benny Kimelfeld and Yael Amsterdamer, editors, *21st International Conference on Database Theory (ICDT 2018)*, volume 98 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[CKP+17] Nadia Creignou, Markus Kröll, Reinhard Pichler, Sebastian Skritek, and Heribert Vollmer. On the complexity of hard enumeration problems. In *Language and Automata Theory and Applications - 11th International Conference, LATA 2017, Umeå, Sweden*, pages 183–195, 2017.

[CM77] Ashok K Chandra 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*, pages 77–90. ACM, 1977.

[CMM+17] Nadia Creignou, Arne Meier, Julian-Steffen Müller, Johannes Schmidt, and Heribert Vollmer. Paradigms for parameterized enumeration. *Theory Comput. Syst.*, 60(4):737–758, 2017.

[CR00] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

[CS18] Florent Capelli and Yann Strozecki. Incremental delay enumeration: Space and time. *Discrete Applied Mathematics*, 2018.

[CV15] Nadia Creignou and Heribert Vollmer. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundamenta Informaticae*, 136(4):297–316, 2015.

[Dam06] Peter Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction. *Theoretical Computer Science*, 351(3):337–350, 2006.

[DG07] Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Logic*, 8(4), August 2007.

[DKV02] Víctor Dalmau, Phokion G. Kolaitis, and Moshe Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proc. CP*, pages 310–326, 2002.

[DM15] Arnaud Durand and Stefan Mengel. Structural tractability of counting of solutions to conjunctive queries. *Theory Comput. Syst.*, 57(4):1202–1249, 2015.

[EG02] Thomas Eiter and Georg Gottlob. Hypergraph transversal computation and related problems in logic and AI. In *European Workshop on Logics in Artificial Intelligence*, pages 549–564. Springer, 2002.

[Fer02] Henning Fernau. On parameterized enumeration. In *International Computing and Combinatorics Conference*, pages 564–573. Springer, 2002.

[FFG02] Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *J. ACM*, 49(6):716–752, 2002.

[FG10] Jörg Flum and Martin Grohe. Parameterized complexity theory. Berlin/Heidelberg/New York, 2010.

[FRU$^+$18] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, pages 165–177. ACM, 2018.

[GLS01] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

[Gra96] Etienne Grandjean. Sorting, linear time and the satisfiability problem. *Ann. Math. Artif. Intell.*, 16:183–236, 1996.

[IUV17] Muhammad Idris, Martin Ugarte, and Stijn Vansummeren. The dynamic yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 1259–1274, New York, NY, USA, 2017. ACM.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[Kaz13] Wojciech Kazana. *Query evaluation with constant delay.* Theses, École normale supérieure de Cachan - ENS Cachan, September 2013.

[Kim12] Benny Kimelfeld. A dichotomy in the complexity of deletion propagation with functional dependencies. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '12, pages 191–202, New York, NY, USA, 2012. ACM.

[KLMN14] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.

[KPS16] Markus Kröll, Reinhard Pichler, and Sebastian Skritek. On the complexity of enumerating the answers to well-designed pattern trees. In Wim Martens and Thomas Zeume, editors, *19th International Conference on Database Theory, ICDT 2016, Bordeaux, France, March 15-18, 2016*, volume 48 of *LIPIcs*, pages 22:1–22:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[KPW17] Markus Kröll, Reinhard Pichler, and Stefan Woltran. On the complexity of enumerating the extensions of abstract argumentation frameworks. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 1145–1152. ijcai.org, 2017.

[KRU15] Egor V. Kostylev, Juan L. Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *Proc. ICDT 2015*, volume 31 of *LIPIcs*, pages 212–229. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[KS13a] Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proc. PODS 2013*, pages 297–308. ACM, 2013.

[KS13b] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic (TOCL)*, 14(4):25, 2013.

[KSS00] Dimitris J Kavvadias, Martha Sideri, and Elias C Stavropoulos. Generating all maximal models of a boolean expression. *Information Processing Letters*, 74(3):157–162, 2000.

[Lat08] Matthieu Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 407(1):458–473, 2008.

[LG14] François Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

[LM14] Katja Losemann and Wim Martens. MSO queries on trees: enumerating answers under updates. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, page 67. ACM, 2014.

[LMS$^+$13] Daniel Lokshtanov, Dániel Marx, Saket Saurabh, et al. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013.

[LPPS13] Andrés Letelier, Jorge Pérez, Reinhard Pichler, and Sebastian Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.

[LWW18] Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.

[MS16] Arnaud Mary and Yann Strozecki. Efficient enumeration of solutions produced by closure operations. In Nicolas Ollinger and Heribert Vollmer, editors, *33rd Symposium on Theoretical Aspects of Computer Science, STACS 2016, February 17-20, 2016, Orléans, France*, volume 47 of *LIPIcs*, pages 52:1–52:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[NS18] Matthias Niewerth and Luc Segoufin. Enumeration of MSO queries on strings with constant delay and logarithmic updates. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 179–191. ACM, 2018.

[PAG09] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[Pap03] Christos H Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the forty-second ACM symposium on Theory of computing*, pages 603–610. ACM, 2010.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, W3C, January 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.

[PS13] Reinhard Pichler and Sebastian Skritek. Tractable counting of the answers to conjunctive queries. *Journal of Computer and System Sciences*, 79(6):984–1001, 2013.

[PS14] Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed SPARQL. In *Proc. PODS 2014*, pages 39–50. ACM, 2014.

[PY99] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

[RT75] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.

[Seg15] Luc Segoufin. Constant delay enumeration for conjunctive queries. *SIGMOD Rec.*, 44(1):10–17, May 2015.

[SSV18] Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. In Jan Van den Bussche and Marcelo Arenas, editors, *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pages 151–163. ACM, 2018.

[Str10] Yann Strozecki. *Enumeration complexity and matroid decomposition.* PhD thesis, Université Paris Diderot – Paris 7, December 2010. Available at `http://www.prism.uvsq.fr/~ystr/these_strozecki`.

[SV17] Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In Michael Benedikt and Giorgio Orsi, editors, *ICDT 2017*, volume 68 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 137–146, 1982.

[VW15] Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 43. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[WW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the Seventh International Conference on Very Large Data Bases - Volume 7*, VLDB '81, pages 82–94. VLDB Endowment, 1981.